Е. М. Лаврищева

ПРОЕКТЫ РФФИ

Монография

№16-01-00352
(2016-2018)

«ТЕОРИЯ И МЕТОДЫ РАЗРАБОТКИ ПРОГРАММНЫХ И ОПЕРАЦИОННЫХ СИСТЕМ»

№19-01-00206
(2019-2021)

«МОДЕЛИ, МЕТОДЫ И СРЕДСТВА ОБЕСПЕЧЕНИЯ НАДЕЖНОСТИ, БЕЗОПАСНОСТИ И ЗАЩИТЫ ДАННЫХ В ПРОГРАММНЫХ, ТЕХНИЧЕСКИХ И ИНФОРМАЦИОННЫХ СИСТЕМАХ»

Москва, 2023

Институт системного программирования
имени В. П. Иванникова РАН

Е. М. Лаврищева

Проекты РФФИ

№16-01-00352 (2016-2018)
«Теория и методы разработки программных и операционных систем»

№19-01-00206 (2019-2021)
«Модели, методы и средства обеспечения надежности, безопасности и защиты данных в программных, технических и информационных системах»

Соисполнители проектов
В. В. Кулямин, Л. Е. Карпов,
В. С. Мутилин, С. В. Козин, С. В. Зеленов, А. Г. Рыжов

Москва, 2023

УДК 681.1.001.

Теория и методы разработки программных и операционных систем. Модели, методы и средства обеспечения надежности, безопасности и защиты данных в программных, технических и информационных системах – Е.М. Лаврищева, В. В. Кулямин, Л. Е. Карпов, В. С. Мутилин, С. В. Козин, С. В. Зеленов, А. Г. Рыжов. – 2016-2021. – 400 с. DOI: 10.15514/LAVR-2023-400.

Аннотация. В монографии представлены методы моделирования и программирования программно-технических, информационных и операционных систем с обеспечением качества, безопасности и защиты, исследованные в проектах РФФИ №16-01-00352 (2016-2018) и РФФИ №19-01-00206 (2019-2021). Основу моделирования и программирования прикладных систем составляет: отечественный метод сборки (связывания–linking) прикладных ресурсов в более сложные системы и 64 функции преобразования обмениваемых типов данных ресурсов с учетом стандарта ISO/IEC 11404 GDT-1996 в общесистемных средах IBM, MS, OMG, Oberon, INTEL, UNIX и др., сделанных в 80-е годы XX века. Представлены формальные средства спецификации ресурсов (VDM, Z, Clear, Monadic, ОКМ, GDM), технологии программирования ОС и трансляторов; системных и сервисных средств моделирования программных и технических систем (SOA, SCA, SCM, WSDL, Grid, Etics и др.). Описана онтология в Semantic Web для представления задач вычислительной математики, геометрии, биологии и Жизненного цикла стандарта ISO/IEC 12207, представленного на Конференции в Лондоне «Sciences and Information – 2015. Определены методы верификации и тестирования изменяемых ресурсов и создаваемых систем из них с обеспечением качества, безопасности и защиты данных. В рамках проектов РФФИ используется стандарт метода сборки IEEE 828-Configuration – 2001 и типов данных, задаваемых стандартом данных 11404 GPD-2007 для структурных и неструктурированных типов данных Big Data и Cloud Computing в современной среде Интернет. Предложен новый вариант сборщика информационных, интеллектуальных и операционных ресурсов с использованием новых операций конфигурационной сборки: config, assembly в .Net, Grid; build, waver в Etics: make в BSD, cmake в GNU и JavaEE. С помощью этих операций создан экспериментальный вариант ядра OS Linux и Веб-система с использованием сервисных ресурсов SOA, SCA, SOAP WWW3C Интернет. Вариант ядра OS Linux отработан с помощью средств верификации, тестирования, безопасности и защиты данных. Рекомендуется студентам старших курсов, аспирантам и научным работникам.

Книга обсуждена на Ученом совете ИСП РАН в ноябре 2022 г.

Предисловие

Широкое распространение современных вычислительных средств и парад программирования ставит задачи по эффективному их применению в процессе моделировании современных задач предметных областей знаний (физики, биологии, матетатики, медицины, генетики и др.) на современном этапе информатизации и интеллектуализации знаний во Всемирном сообществе.

Анализ истории развития таких средств показывает, что программирование прошло путь с 50-х годов XXвека от моделирования математических методов решения физико-технических, прикладных задач. На международных ЭВМ созданы общесистемные ОС (IBM, MS, Intel, Oberon, Unix и др.) для реализации задач в разных областях. На отечественных ЭВМ создавались ОС, ПО, ОМО и реализовывались математические методы в виде математических подпрограмм и в библиотеках (ИС-2, MatLab, Demral, Reuses и др.). Они использовались в авиационной, автомобильной, космической областях, а также при строительстве железных дорог и домов, в торговой и экономической деятельности. Широкое применение получили функциональные компоненты многоразового применения (КПИ) в разных областях знаний и метод конвейерной сборки технических средств (устройств, приборов) и робототехнических изделий в биотехнике, медицине, химии и др. Каждый ресурс (объект, компонент) и системы из них проверялись на безопасность, надежность, качество и защиту обрабатываемых данных. В мировых компьютерных организациях сообществе ISO, IEC, IEEE разработано множество стандартов по разработке, оцениванию и управлению созданием коллективами сложных программных и технических и вариантов ОС.: ПТС, ПС, АИС, АСНИ, АСУ, АСУТП и др. Созданы и действуют робототехнические устройства для проведения физиологических, кардиологических, травмотических, онкологических исследований, и проведения хирургических операций и т.п. В университетах, институтах и школах страны сформировались программы обучения студентов современным основам программирования и инженерни реализации разных математических, физических, информационных, биологических, медицинских и интнллектуальных задач на ЭВМ.

Основу создания технических и программных систем (ПТС) составляет теория математического моделирования сложных систем из готовых ресурсов, отражающая сформировавшиеся знания в области интеллектуальных, информационных, операционных и вычислительных задач. В мировом сообществе в рамках информатизации с 1992 созданы специальные информационные технологические инструменты (E-science, Semantic Web, Grid, Etics и др.), обеспечивающие приобретение и извлечение знаний E-science, накопленных в разных предметных областях знаний для их использования. Одним из средств концептуализации знаний Semantic Web является онтологический аппарат представления знаний в научных, технических, физических, математических и биологических областях знаний. В рамках Международного Европейкого проекта создается международный физический инструмент «Коллайдер», объединяющий 10 тысяч учёных и инженеров из более 100 физических институтов мира для проведения исследований о фундаментальных частицах и их взаимодействиях. В рамках этого проекта образованы системы Grid, Etics для компьютерной и программной поддержки физических экспериментов мирового научного сообщества E-sciences. Прниводится их использование при обработки физических и биологических задач.

Начиная с 2001 года проводились международные конференции (VAMOS, Modeling, Reusebility, Ontology и др.), сформировавшие новые методы моделирования задач из технических и программных областей: физический эксперимент Коллайдер, медицинские, торговые, финансовые онтологии и системы. Одним из путей достижения таких задач

является технология производства программных продуктов (ПП), удовлетворяющая определенным экономическим критериям по организации моделирования, разработки и изготовления продуктов высокого качества и производительности с использованием накопленных готовых ресурсов - ГОР, КПИ (компонентов повторного использования) со стандартными или унифицированными интерфейсами (CALL, RPC, RMI, WSDL), способствующими производить формализованное интеграционное связывание (сборку) ресурсов в ЯП (C++, Java, Python, Ruby, Java, Basic, Snobol и др.) в общесистемных средах (IBMSphere, ,Net, OS Linux, Intel, Unix, JavaEE, Grid, Etics, Semantic Beb,, Visial Studio и др.).

Сборочный конвейерный процесс производства ПП в априори основывается на контроле деятельности исполнителей ресурсов с оценкой показателей качества отдельных КПИ, так и вариантов продукта (например, экспериментального варианта ядра OS Linux). В мировой практике сформировались новые тенденции в современной технологии программирования и моделирования:

– конвейерная сборка с планированием трудозатрат, учетом, контролем результатов и др.) в автомобильной, авиационной, космичесой, строительной, медицинской промышленности и др.;

– анализ сущности решаемых задач в предметных областях и формализованное их описание в виде КПИ, REUSES, ГОР сохранения в Библиотеках для многократного использования в системах представления знаний, ПТС;

– развитие математического моделирования сложных изменяемых систем с обеспечением изменяемости (вариабельности), безопасности, защиты данных и качества систем;

– Европейская система информатизации E-science с системами Semantic Web, Grid, Etics обеспечивают проведение физических экспериментов и создание информационных и интеллектуальных областей знаний.

Наиболее быстрый переход к такой организации работ состоял в систематизации и формализация операций интеграции, сборки многократно используемых готовых информационных ресурсов, в том числе со стандартными и нестандартными типами данных Big Data в Cloud Computing вычислениях в Интернет.

Проблематика математического моделирования вариабельных (изменяемых) сложных ПТС начали развиваться в ИСП РАН (2014-2022) под руководством академика В.П. Иванникова. Методы моделирования ПТС с обеспечением качества, безопасности и защиты данных были представлены в проектах:

РФФИ №16-01-00352 (2016-2018) «Теория и методы разработки программных и операционных систем»

РФФИ №19-01-00206 (2019-2021) «Модели, методы и средства обеспечения надежности, и безопасности и защиты данных в программных, технических и информационных системах».

Данная монография представляет собой сборник основных материалов, которые вошли в научные отчет по результатам исследований в перечисленных выше проектах.

В проекте РФФИ №16-01-00352) успешно завершен в 2018 и зафиксирован в ЦИТИС и E-Library. В нем принимали участие специалистов ИСП - Карпов Л.Е, Томилин А.Н., Кулямин В.В., Митулин В.С., Рыжов А.Г., Петров И.Б (МФТИ).

По проекту РФФИ №19-01-00206 зафиксированы научные тематические наработки по безопасности, надежности и качества ПП в 2019-2021 в ЦИТИС и публикациях. В нем принимали участие специалисты ИСП - Митулин В.С., Рыжов А.Г., Зеленов С.И. и др. Проведено исследование и обработка отдельных методов надежности применительно к специализированным бортовым устройствам и программам Исследован набор международных методов по проблематике безопасности и защите данных программ и устройств от аварий, узроз и контратак. Методы и средства обеспечения безопасности прошли апробацию на отдельных технических, программных и операционных средствах –

эксперименальгом варианте ядра ОС Linux. Методы надежности, качества с обеспечением безопасности и защиты данных представлены в подразделах проекта РФФИ 19-01-00206 (2019 -2021).

В исследованиях и в разработке данных проектов принимали участие более 10 ведущих специалистов ИСП РАН отдела «Технология разработки систем» А.К. Петренко, а также студенты МФТИ, которые на практике отрабатывали отдельные аспекты теории моделирования сложных систем и сделали 7 магистерских диссертаций (Р. Губайдулина, А.Тараймович, В.Левин, А.Иванов, А.Павловская, В.Козин и др.). Научные аспекты моделирования систем и обеспечения надежности, безопасности и защиты данных отображены в докладах и статьях на Всероссийских, Международных конференциях и в Трудах ИСП РАН.

В докладах и статьях отображены научные аспекты поставленных прикладных задач по 2-проектов РФФИ:

– моделирование программно-технических, информационных и операционных систем, начиная с появления компьютеров;

– верификация и тестирование готовых ресурсов (КПИ, reuses, services), используемых при моделировании программных, операционных и технических средств;

– сборка (конфигурационная сборка) готовых ресурсов в сложные системы с доказательством их правильности и безопасности в созданной конфигурационной структуре прикладных и операционных систем OS Linux с помощью операций config, assembly, make, cmake, building, waver;

– теория преобразования разнородных типов данных КПИ, REUSES и систем общих типов данных стандарта ISO/IEC 11404 GDT -1996; GPD-2007 для неструктурированных типов данных Big Data и Cloud Computing;

– логическое специфицирование системных и сервисно-компонентных моделей SOA, SCA, SCM WWW3C и создаваемых новых функциональных элементов прикладных систем;

– обеспечение безопасности, защиты данных, надежности и качества ПТС в клиент-серверной среде Интернет.

Данные методы и средства математического моделирования и программирования ресурсов и ПТС из них представлены в подразделах отчетов РФФИ 2016 -2021.

Раздел 1. Форма 501. Проект № 16-01-00352-2016. Теория и методы разработки вариабельных программных и операционных систем

Лаврищева Е.М. Карпов Л.Е, Кулямин В.В., Мутилин В.С., Рыжов А.Г., Томилин А.Н.

Аннотация. Дано определение теории и методов моделирования изменяемых сложных систем (программных, информационных, операционных, технических) из накопленных готовых ресурсов (ГОР), КПИ, сервисов Интернет и из вновь создаваемых ресурсов (модулей, объектов, компонентов, сервисов, артефактов и др.), записанных в ЯП (Fortran, Cobol, PL-1, C++, Java и др.).

В 2016 году согласно приведенной выше главной задачи проекта проведен системный анализ проблемы вариабельности систем разного назначения и дан обзор современных подходов и методов создания вариантов систем, а также представлена теория моделирования вариабельных операционных и программных систем (ПС), метод генерации отдельных вариантов операционных систем (ОС) из элементов ядра ОС Linux и Web-систем Интернет из сервисных ресурсов. Выполнены следующие научно-технические разработки:

1) Проведен анализ, дана характеристика современных подходов к моделированию и программированию вариабельных ПС и разработаны базовые основы теории и метода объектно-компонентного (ОКМ) моделирования ПС в виде графической объектной модели (ОМ) и модели характеристик FM (Model Feature) с использованием ГОР и артефактов ПС. Основу ОКМ моделирования составляет логико-алгебро-математический аппарат определения ОМ на четырех уровнях проектирования (обобщенном, структурном, характеристическом и поведенческом) по графу систем. На каждом уровне определяются и уточняются характеристики функций объектов ОМ и FM, устанавливаются их отношения и логика поведения. Выделенные объекты и их интерфейсы группируются в классы и могут отображаться в вершинах графа ОМ. Функции объектов трансформируются к виду программных компонентов с сохранением интерфейсов в компонентной модели. На основе ОМ, FM и компонентной модели проводится конфигурационная сборка элементов этих моделей в требуемый вариант вариабельной структуры, способной функционировать и изменяться по требованиям заказчика.

2) Проанализированы существующие в рамках VAMOS (Variability Managment OS) подходы к генерации вариантов ОС из извлекаемых с готового кода систем (OS Linux, Intel, WebSphere IBM, VS.MS и др.) необходимых функциональных артефактов и ГОР. Сформировался язык конфигурирования новых заказываемых вариантов ОС. К методам извлечения артефактов и знаний о них относятся Variability Minning, Data Mining и др. Основу метода Variability Minning составляет нахождение из ядра старого Legacy-кода ОС функциональных артефактов для их повторного использования при формировании из них модели вариабельности MF и системы. Поведение элементов модели MF описывается в языке линейной темпоральной логики (Linear Temporal Logic, LTL) Model Checking для проведения верификации и конфигурации из верифицированных элементов варианта новой ОС средствами языка Konfig. Исследования этой проблематике проводились с участием стулентов МФТИ (Р.Гузазулиной, А Иванова, А.Левина, Козина В. и др.). Ими сделаны магистерские диссертации по использованию Data Minimg Reusebility Variability Minning,

assembly components of OS Linux. Более подробно сборка экспериментального ядра OS Linux будет приведена в Проекте 206 с участием Козина В.

3) Рассмотрены новые механизмы Object-Component Stile на платформах Onion, Spring (Java) и Semantic Web с целью построения моделей OM, MF с помощью объектно-компонентного метода (ОКМ), Konfig и генерации вариантов (версий) систем из композитных сервисов с помощью средств Inversion of Control в среде инфраструктуры Интернет.

4) Предложены механизмы верификации функциональных элементов ОС, входящих в модель вариабельности и в архитектуру системы с отметкой вариантными точками этих элементов. Механизмы верификации включают набор инструментов статической верификации (software model checking).

5) Отработан метод тестирования по методу анализа деревьев ошибок (*FCTA - Fault Contribution Tree Analys)* заданных объектами в вершинах ОМ средствами булевой логики. Проводится тестирование элементов систем с помощью специальных тестов, учитывающих разные комбинации характеристик элементов модели MF на основе концептуальной модели для процесса тестирования систем путем конкретизации ЖЦ ISO/IEC 12207 со специальными механизмами тестирования вариабельной системы и элементов MF. Данная модель тестирования из артефактов системы уточняется при создании экспериментального вариантов ОС Linux и ПС в 2017 г. для применения в заданной предметной области (физики, математики, биологии, медицины).

РАЗВЕРНУТЫЙ НАУЧНЫЙ ОТЧЕТ по проекту 16-01-00352

Согласно принятых целей проекта «Теория и методы разработки вариабельных программных, технических и операционных систем» для разработки в 2016 году поставлены задачи:

– теория и методы моделирования программных систем (ПС), технических (ПТС) и операционных систем (ОС) с помощью готовых ресурсов – ГОР (reuses, assets, artifacts, objects, components, aspects, services и др.) и создание новых вариантов согласно заданных требований;

– теория извлечения знаний об элементах готовых действующих систем (Linux, Intel, Legacy Systems, Real time systems, и др.) и построения на их основе модели характеристик FM - модели создаваемой системы при формировании требуемых вариантов систем в прикладной области знаний.

Основные задачи проекта РФФИ 352 в 2016г. состояли в разработке:

– метода моделирования архитектуры ПС, ОС и Веб-систем, а также модели вариабельности FM для ПС и ОС, создаваемых из готовых артефактов;

– метода верификации моделей FM как для отдельных ГОР, так и ПС, Веб-систем и ОС, функционирующих в общесистемной среде в мировой инфраструктуре Интернет;

– апробации стандартного метода сборки систем из ГОР, включая контроль изменений в систему и управления характеристиками модели FM для эффективного сопровождения системы;

– выбора средств тестирования созданных вариабельных систем из ГОР и оценки надежности их функционирования;

– проведения сборки готовых модулей, объектов КПИ, ГОР в сложные ПС, СПС и ПТС.

В рамках поставленных целей и задач по их реализации в проекте РФФИ 352 проведен системный анализ проблематики вариабельности современных систем разного назначения в статьяхе:

Лаврищева Е.М. и Петренко А.К. Моделирование систем и семейств систем, Труды ИСП РАН, 2016, том 28, вып. 6, 2016, стр. 49-64. ISSN 2220-6426 (Online), ISSN 2079-8156 (Print). DOI: 10.15514/ISPRAS-2016-28(6)-4. Доклад на Научной конференция «Открытые системыы» ИСПРАН, 24-27 ноября 2016 (приводится ниже). Дается краткая характеристика системы АПРОП, обеспечивающей сборку модулей в сложную структуру и преобразованием обмениваемых данных операцией link в 1976 году для ОС ЕС ЭВМ по договору В.М.Глушкова с НИЦЕВТ (1975-1976) под руководством Райкова Л.Д.). Был сделан вариант системы АПРОП, автоматизирующей сложные программы из готовых ресурсов – КПИ, Reuses, Servises.

На данный момент приводится обзор подходов и описание современных методов разработки и моделирования систем, которые могут меняться, а также средств и методов, использующих специализированные инструменты, учитывающие вариабельность ОС. Выделены техники анализа, использующие выбор подходов к определению вариантов общесистемных средств с помощью модели FM и покрытия исходного и выходного кодов системы тестами из комбинации конфигурационных параметров стандарта конфигурационной сборки 1998, которые учитывают вариабельность и итеративность уточнения модели поведения готовой системы.

Определены новые разработки в рамках данного проекта РФФИ 352 в 2016 году, включая базовые основы метода моделированию и программирования ПС, ОС (Grid, Etics, MS, Product Line, Linux, Intel, Oberon и др.), Web-system, WebSphere IBM, Onion Architecture, Semantic Web, Grid, Etics, включающие средства:

формальное описание для ПС, ОС, ПТС модели Feature Model – FM, с помощью готовых ресурсов (ГОР) и новых артефактов системы, задаваемых в графической Объектной Модели – ОМ с помощью логико-алгебро-математического аппарат ОКМ на четырех уровнях проектирования (обобщенном, структурном, характеристическом и поведенческом). На каждом уровне моделируются и уточняются функции объектов, устанавливаются их интерфейсы и отношения, выявляются и определяются базовые характеристики функций объектов и логика их поведения. Выделенные объекты в ОМ и их интерфейсы группируются в классы и отображаются в вершинах графа ОМ. На основе базовых характеристик этих объектов создается модель MF, участвующая в проведении сборки необходимых вариантов системы. Методы и функции объектов трансформируются к виду программных компонентов с сохранением интерфейсов и отображаются в компонентной модели. На основе ОМ, компонентной и вариабельной модели проводится конфигурационная сборка требуемых элементов и ГОР в файл системы для последующего выполнения, изменения и обработка. В рамках данного проекта метод ОКМ усовершенствован для моделирования ОС и Веб-систем, артефактами которых являются элементы – функциональные элементы в коде ядра ОС Linux, системные сервисы и сервисы Веб-систем, которые композируются в новое образование (хореографию прикладных систем.

1) Исследованы методы извлечения артефактов и готовых ГОР из оперативных Legacy Systems (OS Linux, Intel, WebSphere IBM, Product Line, Semantic Web и др.), из которых конфигурировались новые заказываемые варианты систем под определенные функции предметной области. К методам извлечения относятся Variability Minning, Data Mining, и др. Основу метода Variability Minning составляет нахождение и извлечение со старого Legacy-кода функциональных артефактов для их повторного использования и формирования из них модели MF, по которой будет производиться ее верификация и сборка элементов в некоторый вариант новой системы с обязательным тестированием функциональности созданного варианта системы.

2) Проведен анализ в классе Web-system объектной платформы: Onion, Spring (Java), Semantic Web для выявления Object-Component Stile анализируемых систем и построения модели ОМ и MF с целью проведения сборки средствами ОКМ и Inversion of Control [3,4].

Приводится магистерские работы на тему я Variability Minning, Data Mining (Р. Газазулина и Ильяс Гарипов).

3) проведен эксперимент по верификации моделей вариабельности ПС, ОС и артефактов архитектуры системы с вариантными точками с помощью инструментов статической верифкации - software model checking. Метод верификации после ОКМ будет модернизироваться под функции OS Linux.

4) исследован метод моделирования в международном проекте E-Science и приведены результатов исследования проектов Коллайдера, систем Grid с реализацией метода моделирования технических и программных систем представления знаний.

5) проведен анализ вариантов тестирования ПТС с помощью дерева ошибок (*FCTA - Fault Contribution Tree Analys) и* булевой логики элементов, расположенных в вершинах графа ОМ. С учетом специфики объектов модели ОМ наиболее приемлемыми подходами к тестированию являются *подходы,* основанные на разработке тестов, учитывающих разные комбинации характеристик элементов модели MF и позволяющие автоматизировать генерацию тестов. Разработана оригинальная концептуальна модель процесса тестирования ПС по онтологии в Семантик Веб при моделировании варианта ЖЦ ISO/IEC 12207 (2015), которая позволяет тестировать модель и ее артефакты. Модель ядра Мос будет создаваться, уточняться и проверяться в экспериментальном варианте в 2017, 2018гг. проекта РФФИ.

Данные задачи для проекта МФТИ 352-2016 описаны в подразделах I, II и III.

Подраздел I проекта РФФИ 352-2016. Теория разработки вариабельных программных и операционных систем

1. Общие положения теории и методов моделирования прикладных систем

Основу моделирования систем и их семейств составляют модели архитектуры и базовые характеристики. Идея метода сформировалась под влиянием ООП и UML (Unified Modelling Language, 1994). Моделирование в этом языке состоит в построении диаграммной модели системы, в вершинах которой находятся отдельные объекты системы, а на дугах отношения между ними. Элементы модели трансформировались к программному коду, который тестировался и документировался. Начиная с 1996 г., появились новые модели: MDA (Model Driving Architecture) OMG, MDD (Model Driving Development), IBM Rational, MDE (Model Driving Engineering), SOA (Service Oriented Architecture) и сервисно-компонентнаz модель SCM IBM и др.

Сформировался общий «взгляд» на моделирование систем по моделям или «каркасам» и метод их трансформации к коду для задаваемой платформы (PSM, PIM). Модельный подход стал доминирующим для разработчиков сложных систем. Он постоянно обсуждается на конференциях MODELS (1999-2016) и получил новое развитие в направлении изменяемости (вариабельности) систем с помощью формальной модели FM и модели конфигурирования вариантов программных и ОС в проекте SEI USA (2004) ProductLine/Productfamily и в ряде других зарубежных проектов. Появились новые языки и подходы к моделированию вариабельных систем для ПС и ОС после проведения Международных конференций SPLE,

VAMOS (2004 – 2015); ICTERI (2006 –2016); Reusebility (1994 –2016) и др. К языкам моделирования относятся:

– сценарии использования (use cases) UML, диаграммные характеристики – FODA, ConIPF, модель архитектуры Koala, xADL и др.;

– OVM, VSL, ConIPF для описания артефактов и наборов готовых ресурсов для приложений с вариантными точками;

– Corba, Prolog и объектно-компонентный метод – ОКМ [2] определения моделей взаимодействия элементов прикладных систем;

– средства описания конфигурационной модели - KConfig (http://www.Sap.org) и Web–инструментов сборки ГОР - Kbuild и др.

Далее приводится краткая характеристика основных подходов и методов моделирования систем ПС и их семейств СПС.

1.1. Современные подходы к моделированию сложных систем

Рассматриваются модели моделирования разного рода прикладных систем.

Язык моделирования UML

UML был первоначально (1998) реализован в Rational Rose, а затем оформлен как стандарт IBM Rational и консорциума OMG (Object Management Group) в виде объектной модели. UML. Этот язык начали использовать многие фирмы-производители ПО (Microsoft, IBM, Hewlett-Packard, Oracle, Sybase и др.) в продуктах (Paradigm Plus 3.6, System Architec, Microsoft Visual Modeler for Visual Basic, Delphi, PowerBuilder и др.). Это диаграммный язык моделирования и документирования систем. В нем модель системы задается набором Use Case диаграмм вида: вариант использования (use case diagrams); классы (class diagrams); поведение (behavior diagrams); взаимодействие (interaction diagrams); состояние (statechart diagrams); деятельности (activity diagrams), реализации (implementation diagrams) и др. Сформированная модель системы задает архитектуру системы, которая трансформируется к программным текстам отдельных элементов в языках C++, Pascal, Basic и др. Однако запись структуры в этом языке отличалась сложностью созданной структуры и требовались дополнительные механизмы проверки правильности. Однако промышленного использования UML не получил, но экспериментальных вариантов систем было много. Проведен анализ подходов к разработке программных систем и описывается ниже.

Модели (MDD, MDA, PSM, CIM, SOA, SCA и др.) для разработки систем

Модель MDD базируется на UML. Элементы модели трансформируются к коду (подобно байт кода JAVA) и задают модель приложения (Application Model), члены которой зависят от платформы реализации элементов программ и данных. В нем управление проводится по точкам вариантности и преобразования данных с помощью платформенных моделей PIM↔PSM. Изготовленные члены семейства различаются не только на уровне платформы, но и на уровне функций ПС, а также заданных требований и полученной оценки качества.

Модель MDA возникла в рамках стандартов ООП, OMA, ORB и языков UML, XML и др. Архитектура привязана к конкретной платформе. Сначала создается общая и независимая от платформы модель приложения, а затем осуществляется ее реализация к требуемой платформе путем трансформации моделей PIM↔PSM, функций ПС, требований к качеству и решениям, активам (assets), которые реализуют альтернативные возможности в системе. Строиться модель домена или прикладной системы, которые трансформируются к модели MDD.

Модель PSM задает состав, структуру, функциональность системы для конкретной платформы. Модель платформы включает технические характеристики, интерфейсы и функции платформы. Она используется при преобразовании модели PIM в модель PSM. В зависимости от уровня детализации платформы, модель может содержать сведения о

различных функциональных элементах системы. Логика системы задает описание основных функций, обеспечивающих их исполнение по назначению. При задании данных используются разные их форматы и механизмы доступа. Трансформация модели PIM и PSM проводится следующими действиями:

– разработка схемы отображения элементов модели (mapping),

– маркирование (marking),

– трансформация (transformation).

Для каждой платформы создается схема преобразования, зависящая от возможностей платформы. (формата, языков, инструментов и др.). Схема включает совокупность элементов и их характеристик, а также механизмы их представления (метамодель, форматы данных и др.). Свойствам метамодели и элементам PIM ставится в соответствие характеристики модели PSM.

Модель CIM (Computation Independent Model**)** основывается на модели PIM и методе ее трансформации к PSM в языке UML. Эта модель включает в себя требования к системе, словарь понятий и формат данных среды, в которой система будет функционировать. Модель ориентирована на описание бизнес-логики, взаимодействие подсистем с помощью интерфейсов. Активно используются модель PIM и MDA, как средства преобразования среды разработки.

Модель SOA – это архитектура из сервисных и системных элементов, которые группируются для серверной стороны с помощью готовых сервисов, служб, интерфейсов, входных/выходных данных и портов обмена метаданными в языке WSDL. Объекты модели SOA задают описание некоторой функции (Function) с заданным качеством сервиса (Quality service) в IT-стандартах комитета W3C.

Модель SCA содержит сервисы и компоненты, которые разработаны различными компаниями: EJB сервер приложений J2EE, сетевые сервисы, объекты планирования, доступа к БД и к системе предприятия. Эта модель обобщает компонентную модель семейства ПП, элементы которой содержит удаленные reuses для обмена гетерогенными данными и сервисами из множества общих сетевых сервисов. SCA сервисы могут иметь различные образования (хореографии), которые используются для интеграции и конфигурационной сборки вариантов систем.

Приведенный набор моделей архитектур ПС является основой моделирования современных программных и операционных систем в среде Интернет

1.2. Подходы к моделированию изменяемых систем (SPLE, OKM, Grid, Etics)

В SEI USA (2005) предложен метод **SPLE** (Software Product Line Engineering) для изготовления программных продуктов – ProductLine/ProductFamily, основанных на модели вариабельности **MF (Model Feature)** и модели конфигурационной сборки готовых ГОР (artifacts, components, reuses, assets, services и др.) по заказу потребителей. ***Клаус Похл*** определил понятие вариабельности в SPLE, как моделm FM для элементов системы, помечаемых вариантными точками. *Вариабельность* – это свойство продукта (системы) к расширению, изменению, приспособлению или конфигурированию с целью использования в определенном контексте и обеспечения последующей его эволюции. Модель FM формируется в процессе разработки ПП аналитиками и разработчиками и включает общие функциональные и нефункциональные характеристики элементов системы, которые являются готовыми reuses (product configuration). К главным аспектам обеспечения вариабельности разных продуктов системы отнесены:

– моделирование вариабельности на уровне артефактов с вариантными характеристиками;

– управление (планирование, контроль и регуляция) конфигурацией системы по модели системы и модели вариабельности FM.

Множество видов одной характеристики, каждая из которых относится к *вариантной характеристике*, образует коллекцию, члены которой присоединяются к ядру системы через точки вариантности.

Точка вариантности – это место в созданной системе, по которой осуществляется выбор элементов вариантов системы. Вариантная характеристика определяет количество точек вариантности системы, присоединяемых к ядру системы.

Основу концепции изменяемости/вариабельности ПП составляют внешние характеристики и свойства функций системы, которые могут изменяться. По модели системы и MF создается конфигурационный файл системы для запуска и выполнения системы. Совокупность систем и подсистем с общим множеством характеристик и требований к ним образуют семейство СПС. Каждый члены семейства проверяются и верифицируются и при обнаружении ошибок в него вносятся измененяемые элементы или удаляются.

ОКМ моделирование систем из объектов, модулей и компонентов*

Объектно-компонентный метод (ОКМ)* обеспечивает логико-математическое моделирование системы на четырех уровнях (обобщенном, структурном, характеристическом и поведенческом). Каждый уровень детализирует и уточняет выделенные в предметной области (ПрО) объекты и задает модель системы в виде графа объектов и их отношения на множестве объектов предметной области с постепенным уточнением их денотатов и концептов по теории Фреге и определением характеристик объектов в модели MF.

* Лаврищева Е.М. Научные основы программ, технологий и систем. - Доклад на Российской Академии наук. Центральный дом учёных. 15.03.2017;

Лаврищева Е.М. Теория графового моделирования сложных систем из модульных элементов для прикладных областей. - Austria-science»1 часть №28/2019. - p.12-30. http://austria-science.info.

Задание структуры системы и действий над объектами (объединение, удаление, замена и др.) осуществляется математическими и логическими операциями ($\cup$, $\cap$, /, $\Diamond$, $\oplus$, – , =, &, и др.).

На первом уровне проводится декомпозиция предметной области на объекты. На следующих определяются внешние характеристики MF и создается графовая ОМ, в вершинах которой находятся объекты, а дуги задают направление связи (интерфейсы) объектов, передаваемых между ними. Для объектов определяется их поведение в заданной среде. Объекты графа переводятся к программным компонентам, которые погружаются в компонентную среду. Объекты и компоненты этих моделей собираются в репозитории компонентной среды и могут конфигурироваться в разные варианты продуктов и ПС.

Теория ОКМ построена с использованием денотационной теории Фреге, теории множеств Геделя-Бернайса и базовых объектных понятий Г. Буча (классы, полиморфизм, наследование и др.). Объект выделяется на уровне анализа прикладной области с привлечением логико-математических формализмов для задания и уточнения функций объектов, их характеристик в объектной модели (ОМ), MF и отображения взаимоотношений и поведения объектов.

Согласно концепции треугольника Фреге, объект (рис.1) — это именуемая часть реального домена с определенным уровнем абстракции, заданная в виде денотат, знака и концепта. Каждый объект ОМ как сущность принадлежит некоторому множеству объектов $\boldsymbol{O} = (O_0, O_1, ..., O_n)$, ***где*** $O_i = O_t$ ($Name_b$ Den_b Con_j***)*** и параметры означают — знак (имя-***Name***), денотат- ***Den*** и концепт-***Con*** объекта**.** Поведение концепта определяется на множестве предикатов $\boldsymbol{P} = (P_1, \boldsymbol{P_2}, ..., \boldsymbol{Pr})$ с помощью функции задания $\boldsymbol{Con_t} = (P_{;1}, P_{i2}, ..., Pjj)$.

Объект треугольника Фреге – это ***денотат,*** отражающий смысл (сущность) объекта. При идентификации объекта используются математические символы (знак, слово, имя), денотат отражает смысл объекта, а ***концепт*** - уровень абстракции. Объект состоит из имени (идентификатора) в одной вершине, денотата – образа предмета в другой вершине, на который указывает этот идентификатор, и концепта в третьей вершине, указывающей смысл денотата.

При моделировании объекта ПрО задается хотя бы одно свойство или характеристика и уникальная идентификация во множестве объектов и предикатов отношений между ними.

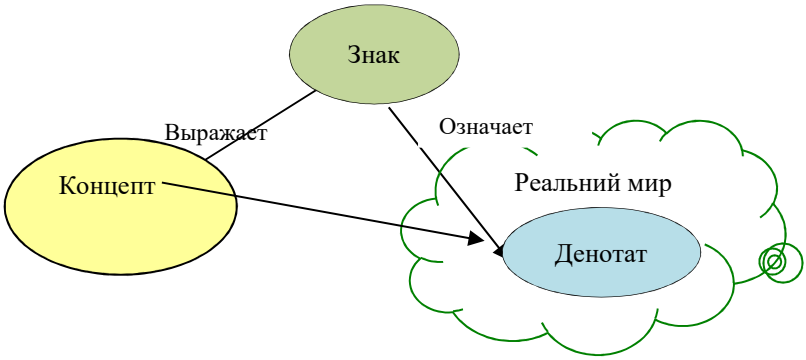


Рис. 1. Треугольник Фреге

Свойство объекта определяется на множестве объектов ПрО унарным предикатом, который получает значение истины при наличии внешних и внутренних характеристик объекта.

Характеристика – это свойство, определенное на множестве внешних и внутренних характеристик с условием получения истины из области значений свойств.

Отношение определяется бинарным предикатом на множестве объектов ПрО, принимающих значение истины на заданной паре отношений объектов типа *IS –A / Part – Of*.

Свойства, соответствующие одной и той же характеристике, могут использоваться в концепции структурной упорядоченности при конкретизации объектов и определении их принадлежности к классу. Определение принадлежности объектов достигается путем реализации отношение принадлежности типа "Part-of", операции детализации и агрегации.

Изменение денотатов и концептов объектов

Все элементы треугольника Фреге – имя, денотат и концепт могут изменяться. Денотат, как некоторая сущность некоторой реальности, соответствует определенному объекту и может представляться в виде совокупности однородных или неоднородных элементов (предметов).

Эти совокупности могут изменяться *декомпозиционным* или *композиционным методом*.

Декомпозиционные изменения концептов включают концепты новых детализированных объектов с учетом концепта объекта или без него.

При изменении уровня детализации (абстракции) концептов объекта исключается одно или несколько свойств или характеристик формируемого концепта с одинаковым денотатом. Каждая из операций изменения объекта имеет определенный приоритет и арность, а также связана с соответствующими допустимыми изменениями денотатов и концептов.

Каждый объект как сущность принадлежит некоторому множеству объектов $O= (O_0, O_1 ,..., O_n)$, где $O_i = O_i (Name_i, Den_i, Con_i)$, а $Name_i, Den_i, Con_i$ соответственно означают – знак

(имя), денотат и концепт объекта. Поведение концепта $Con_i = (P_{i1}, P_{i2}, ..., P_{is})$ определяется на множестве предикатов $P=(P_1, P_2, ..., Pr.)$.

Между определенными элементами множества *О* существуют отношения принадлежности. На этом уровне выделяют сущности ПрО путем собственного субъективного восприятия и описывают их объектами в виде базовых понятий ОМ. Выделение сущностей осуществляют с учетом отличий, определяющих соответствующие им понятийные структуры, исходя из треугольника Фреге, и фиксируют объекты идентификаторами и концептами. Объект имеет хотя бы одно свойство или характеристику и уникальную идентификацию во множестве объектов, предикатов свойств и отношений между ними. Отношение определяется бинарным предикатом на множестве *О* ПрО, принимающих значение истины на паре *IS –A/Part – Of*.

Уровни логико-математического моделирования ПрО в ОКМ

Четырехуровневое объектное моделирование ПрО является развитием модульной парадигмы декомпозиции ПрО из функциональных и интерфейсных объектов и их определения и уточнения на четырех уровнях логико-математического аппарата:

1). **Обобщающий уровень** определяет базовые понятия ПрО – объекты без учета их сущности и свойств. Объект задается в виде денотата в соответствии с теорией Фреге в виде <имя объе> <концепт>.

2). **Структурный уровень** определяет распоожение объектов в структуре модели ПрО, устанавливает упорядочение объектов и представление их структуры в виде графа с операциями над объектами (объединения, пересечения, разности, приложения, симметричной разницы и др.

3). **Характеристический уровень** задает общие, специфические свойства и характеристики концептов объектов в логико-алгебраическом представлении объектов в виде графа с характеристиками и свойствами в модели МХ;.

4). **Поведенческий уровень** определяет поведение и изменение объектов в зависимости от событий, которые они создают при их выполнении.

Рассмотрим более подробно уровни проектирования модели ПрО.

1). Обобщающий уровень проектирования ОМ

Этот уровень задает наивысшую меру абстракции отображения ПрО в ОМ с использованием концепции обобщения с целью определения сущностей ПрО и представление их в виде объектов ОМ как базовых понятий модели. Результатом является спецификация объектов вида:

<имя объекта> <концепт>,

где *имя объекта* – идентификатор из символьной строки литер и десятичных цифр;

концепт – текст, который определяет семантику <денотат объекта>.

Выделение сущностей осуществляется с учетом изменений, которые определяются понятийными структурами – треугольниками Фреге и фиксируются идентификаторами и концептами. Результат этого проектирования – ОМ, в которой размещены именованные объекты ПрО, образующие множество $O=(O_0, O_1, ..., O_n)$, где O_0 – объект ПрО и множество $O'=(O_1, ..., O_n)$ денотаты и концепты объектов после декомпозиции и композиции.

Исходя из треугольника Фреге, объекты получают идентификаторы и концепты. Результат этого уровня проектирования – ОМ, в которой размещены объекты и измененные денотаты.

2). Структурный уровень моделирования ОМ

Логико-алгебраическая концепция структурного уровня рассматривает множество объектов ПрО как алгебраическую систему из совокупности объектов и предикатов определенной сигнатуры, которые получаются с помощью операций:

– 0-арной для задания констант в заданных характеристиках ПрО;

– унарной для отображения свойств отдельных объектов;

– бинарной для задания взаимосвязей между отдельными парами объектов.

Объект имеет внешние и внутренние свойства и характеристики.

Характеристика объекта могут быть внешние и внутренние.

Внешняя характеристика предназначена для описания проблемной ориентации совокупности объектов и их статуса как элементов множеств. Каждое внешнее свойство входит в состав объектной характеристики одного проблемного объекта ПрО.

Внутренняя характеристика предназначена для определения статуса объекта, как одного из критериев формирования множества из объектов, эквивалентного внешним свойствам, выделенных в системе предикатов, в которой каждый объект принимает значение истины.

Объекты, определенные на обобщающем уровне абстракции задаются как множество или элемент множества $O=(O_0, O_1, O_2 . .., O_n)$. Если из него исключить элемент O_0, получаем множество

$O'=(O_1, O_2, , ..., O_n)$, которое трансформируется в такой вид:

$\forall i \; \exists j \; [(i>0) \; \& \; (j>0) \; \& \; (i \neq j) \; \& \; (O_i \in O_j)]$. (1)

В выражении (1) каждый из объектов является множеством и к ним применяются операции теоретико–множественной алгебры $\Omega = (\cup, \cap, /, \Diamond, \oplus, -)$. Выражение (1) определяет отношение часть–целое, экземпляризации и агрегации. Тогда $\Sigma = (S, \Omega)$ определяет алгебраическую систему для структурно-упорядоченного уровня. Оно представляются как множества или элементы определенного множества алгебраической системы $\Sigma = (O', \Omega)$ структурного уровня.

Таким образом, на данном уровне осуществляет решение следующих задач:

– определение и фиксация структурной упорядоченности объектов;

– расширение описания объекта или конкретизацию объекта в структуре ПрО;

– определение новых объектов, связанных с заданными объектами;

– формирование объектного графа ПрО.

Принцип вычитания обеспечивает структурную упорядоченность объектов или системы путем применения отношения принадлежности. Выделение объекта и фиксацию его внешних различий осуществляется на структурном уровне в объектном графе ОМ (рис.2).

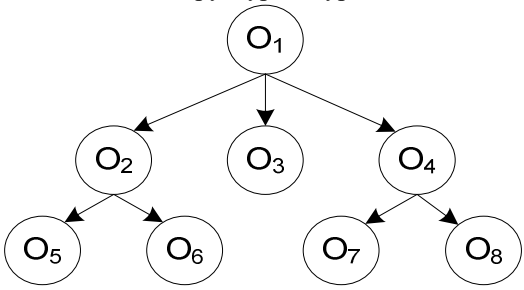


Рис. 2. Структура графа G ={O}

Для графа G проводится корректировка я и реструктуризация с помощью операций объединения, пересечения, дополнения и др. Затем проводится контроль этого графа на структурную упорядоченность новых объектов и определяются операции конкретизации и принадлежности объектов. Контроль полноты и не избыточности графа ОМ позволяет устранять сдублированные объекты в графе. ОМ.

3). Характеристический уровень проектирования ОМ

В соответствии с характеристическим уровнем для каждого из объектов формируется его концепт. Если $O'=(O_1, O_2, \ldots O_n)$ – совокупность объектов ПрО, а $P'=(P_1, P_2, \ldots P_r)$ – множество унарных предикатов, которые связаны со свойствами объектов ПрО, то концепт O' объекта O_i является множеством утверждений, которые построены на основе предикатов с P', принимающих значение истины для соответствующего объекта.

Аксиома 1. Каждый объект ПрО имеет хотя бы одну характеристику, которая задает семантику и уникальную идентификацию во множестве объектов ПрО.

Характеристики определяются типом объекта, принадлежат только одному объекту модели ПрО и задаются в виде списка или множества свойств с несовпадающими типами. Любой объект множества или элемент его может соответствовать только одной внешней и внутренней характеристики. Основное условие определения принадлежности объекта к множеству эквивалентно внутреннему свойству объекта-множества. Для функциональных объектов определяются интерфейсы, которые передают внешние характеристики другим элементам, связанным на графе отношением принадлежности. В ранее построенном графе уточняются характеристики модели MF и интерфейсы: $G=\{O, I, R\}$, в котором O – множество объектов (функций), I – множество интерфейсов и отношений R (relations) между объектами.

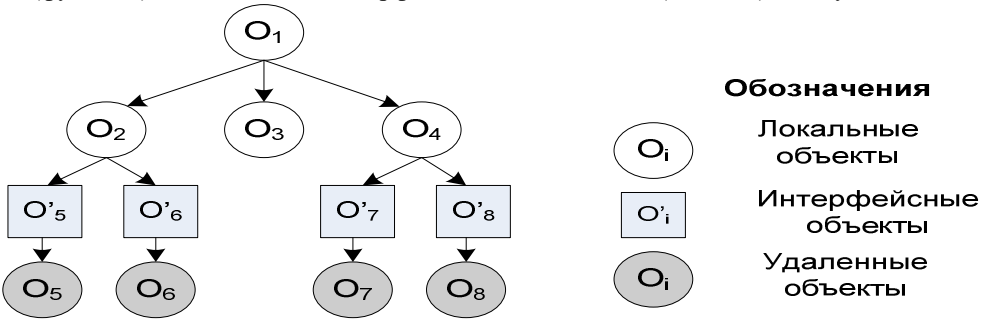


Рис. 3. Граф G на множестве объектов и интерфейсов

В граф G добавлены интерфейсные объекты I (рис.3), O'_5, O'_6, O'_7, O'_8, выполняющие вызов объектов и передачу им данных в требуемом формате. Вершины данного графа G залают функциональные объекты O_1, O_2, O_3, O_4, O_5, O_6, O_7, O_8 и интерфейсные объекты – O'_5, O'_6, O'_7, O'_8, которые размещаются в репозитории, а дуги соответствуют отношениям между всеми видами объектов графа.

Элементы графа $O_1 – O_8$ описываются в ЯП, а интерфейсные объекты $O'_5 – O'_8$ в специальном языке интерфейса IDL (Interface Definition Language), разработанного в рамках проекта CORBA. Параметры внешних характеристик интерфейсных объектов передаются между объектами через интерфейсы и помечаются как *in* (входной), *out* (выходной), *inout* (входной и выходной) в IDL.

Интерфейсные объекты графа содержат описание передаваемых данных, операторы вызовов RPC или RMI и передачи данных. При необходимости передаваемые данные преобразуются брокером ORB к соответствующим форматам среды выполнения метода объекта.

По графу G (рис.3) можно построить программы $P_1 – P_5$ с использованием операторов объединения (сборки) *link*:

1) $P_1 = O_2 \cup O_5$, *link* $P_1 = In\ O'_5\ (O_2 \cup O_5)$;
2) $P_2 = O_2 \cup O_6$, *link* $P_2 = In\ O'_6\ (O_2 \cup O_8)$;
3) P_3;
4) $P_4 = O_4 \cup O_7$, *link* $P_4 = In\ O'_7\ (O_4 \cup O_7)$;
5) $P_5 = O_4 \cup O_8$, *link* $P_4 = In\ O'_8\ (O_4 \cup O_8)$;
6) $P_0 = (P_1 \cup P_2 \cup P_3 \cup P_4 \cup P_5)$.

Сборка двух объектов графа (например, *link* $P_2(O'_2)$ проводится через интерфейс $In\ O'_2$.

Аксиома 2. Расширенный граф G с интерфейсными объектами, структурно упорядочен (наверх), проконтролирован на полноту, избыточность и дублирующие элементы.

Объекты могут иметь несколько интерфейсов, которые могут наследовать интерфейсы других объектов в том случае, когда они предоставляют данные всему множеству входных интерфейсов.

Множество объектов и интерфейсов считаются достоверными, если выполняется условие: каждая внутренняя характеристика эквивалентна внешней характеристике объекта. Если это условие не выполняется, то такой элемент удаляется из множества *O* и из графа соответственно. Проводится контроль графа ОМ с помощью логико-алгебраических операций и выявляются ситуации нарушения структурной упорядоченности объектов, а затем проводится группирование с помощью операций бинарных отношений типа *'is-a'* и для пары "объект-множество – объект-элемент" выполняются операции э*кземпляризации, классификации* и *обобщения.* Для реализации бинарного отношения *'Part–of'* пары "объект-элемент – объект-элемент" используются операции: *агрегации* и *детализации.*

4). Поведенческий уровень проектирования ОМ

Входными данными для этого уровня являются ОМ на предыдущих уровнях. На данном уровне определяется последовательность состояний объектов и процессы перехода состояний. Взаимосвязи между объектами формируются с помощью бинарных предикатов, которые связаны со свойствами объектов ПрО, и детализируют взаимосвязи между состояниями объектов.

Данный уровень базируется на системе предикатов, определяющих подмножество *SA* и множество объектов *S*, которые отображают сущность объектов ПрО и определяют их поведение, без учета внутренних свойств. Понятие времени – это абстрактное понятие, оно соответствует конкретному параметру системы, значения которого упорядочены, и каждому из них соответствует состояния объектов, к которым относятся: атрибут состояния, статический и динамический атрибут состояния. ОМ задает модель состояний для каждого объекта, переход объекта из одного состояния в другое посредством некоторого сообщения или события, а также изменение состояния объекта посредством набора методов, ассоциируемых с этим объектом. Кроме того, объекту соответствуют не только виды сообщений (событий), порождающие изменение состояний и список сообщений, на которые объект реагируют, как на сообщения времени. С каждым состоянием объекта связано некоторая деятельность (*Activity*), которая дальше будет иметь название действия (*Action*), и происходить в тот момент, когда объект достигает определенного состояния. Операции действия имеют доступ к модели состояний и к данным других объектов.

В процессе проектирования ОМ объекты могут объединяться в классы с учетом общих характеристик. *Класс* обозначает множество объектов, имеющих общие переменные, структуру и поведение. Класс конкретизируется в виде:

– экземпляра класса, как объекта, который является элементом определенного множества и сам является классом;

– объединенного класса, как множества, которое является прямой суммой нескольких других множеств;

– класса-пересечения, как множество, которое является общей частью других множеств;

– агрегированного класса, как множества, которое является подмножеством декартова произведения нескольких других множеств.

1.3. Вариабельность систем и управление

Вариабельность – это свойство системы (продукта) к расширению, изменению, приспособлению или конфигурированию с целью использования в определенном контексте и обеспечения ее эволюции. Вариабельность зависит от требований к системе, модели MF_{var}, графа *G* ОМ а также описания тестов, документации на их элементы и др. В целом, модель

может быть реализована как для системы, так и для семейства систем. Объектами управления вариабельностью в ПС и СПС являются:

точка вариантности или вариантный артефакт, т.е. его формальное представление и возможность его реализации для разных ПС;

вариант – элементарный артефакт ПС одного типа с соответствующим ему вариантным артефактом, который представляет собой описание в заданной ПС;

ограничение – предикат, определенный на декартовом произведении декартовых квадратов множества точек вариантности и вариантов;

зависимость – предикат, который определяет допустимые взаимосвязи между точкой вариантности и множеством вариантов для ПС.

Модель вариабельности: $MF_{var} = (SV; AV)$,

где SV – подмодель вариабельности артефактов системы, AV – подмодель архитектуры продукта системы.

Подмодель $SV = ((G_t, TR_t), Con, Dep)$,

где $G_t = (F_t, LF_t)$ – граф артефактов типа t (требования, компоненты, тесты и др.); TR_t – двусторонние связи артефактов типа t; *Con* і *Dep* – предикаты на декартовом произведении множеств артефактов, которые определяют ограничения и зависимости между функциями и показателями качества ПС.

Подмодель $AV = (G, TR, G_t)$,

где G - граф архитектуры системы ПС из артефактов, готовых Reuses (artifacts, assets, services, components) и вариантных характеристик; TR – связь элементов архитектуры и артефактов G_t.

Модель SV конкретизируется в специальную линию разработки ПП и конфигурируется из артефактов и элементов архитектуры в код системы. Точки вариантности позволяют управлять трансформацией элементов графа и заменой одних артефактов другими, новыми функциональными или более корректными элементами архитектуры.

Управление вариабельностью

Е. Дьоминг предложил метод управления эффективной организацией ПС и семейств СПС, выделив четыре функции для управления его вариабельностью:

1) *F1* - функция планирования вариабельности в артефактах ПС (на уровнях инженерии домена и приложений);

2) *F2*- функция реализации вариабельности в архитектуре ПС;

3) *F3* – функция системного мониторинга вариабельности ПС;

4) *F4* – функция актуализации вариабельной ПС.

В основе управления вариабельностью ПС и СПС лежат требования R (Requirements) к системе, которые используются для:

1) обоснования - наличие объективных оснований принятия решений (assets) для *F1* – *F4* (требование *R1*);

2) согласованности – одинаковость способа выработки и реализации решений на всех уровнях абстракции и на всех этапах процесса разработки ПС (требование *R2*);

3) масштабности – независимости способа выработки и реализации этих решений от объема функциональных возможностей ПС (требование *R3*);

4) трассирования – возможности отслеживания связей между характеристиками вариабельности на всех процессах разработки СПС (требование *R4*);

5) визуализация свойств вариабельности и связей между характеристиками (требование *R5*).

Приведенные требования *R1 – R5* для функций *F1 – F4* обеспечивают модельную среду их реализации. Необходимым элементом этой среды может быть интегрированная (*VM*) модель вариабельности семейства ПС, которая объединяет модели подсистем семейства и позволяет дать оценку уровню вариабельности системы и степени ее соответствия требованиям.

1.4. Стандарт конфигурационной сборки прикладных систем

К **основным** задачам конфигурационной сборки ГОР стандарта IEEE 824-1987 в ПС относятся следующие:

– поддержка операций добавления требований, их трассировка в вариантах, используемых ГОР на всех уровнях;

– порождение новых вариантов архитектуры и компонентов в точках вариантности для реализации ранее установленных требований к ПС с учетом заданных ограничений;

– принятие эффективных решений по разработке новых вариантов ГОР через предоставление информации для проведения анализа «затрат/преимуществ» с учетом опыта, накопленного в базе знаний проектов ПС;

– управление конфигурированием ПС путем выбора вариантов ГОР для ПС;

– управление вариабельностью ПС при планировании и расширении готовых компонентов новыми вариантами и контроля полноты и непротиворечивости вариантов ресурсов для требований ПС.

Управление конфигурацией КПИ, ГОР состоит в:

– сборке ресурсов по моделям ПС и *Mvar* (контроль, верификация ресурсов, тестов);

– систематическом отслеживании внесенных изменений путем аудита изменений и автоматизированного контроля изменений;

– поддержке целостности конфигурации и ее аудит;

– ревизии конфигурации путем проверки программных или аппаратных элементов в соответствии с версиями конфигурации с учетом требований к ПС;

– трассировки изменений конфигурации на этапах сопровождения и эксплуатации;

– согласования между собой объектно-компонентных и вариабельных моделей характеристик ресурсов и ПС;

– доказательстве изоморфного отображения объектных и компонентных моделей ПС в программные структуры путем последовательной трансформации методов объектов, их данных, задаваемых в структуре ПС.

При сборке ГОР в ПС используется модель вариабельности MF, содержащая все характеристики функций системы. Вариабельные элементы используются на всех уровнях представления и взаимосвязи ресурсов в ПС.

Готовые ресурсы в стандартном виде накапливаются в репозиториях разрабатываемой ПС. Они отбираются, адаптируются и интегрируются в единую систему.

Основную роль в этих процессах выполняет конфигуратор ИТК. Он обеспечивает связь ГОР и их интерфейсов с вариантами отдельных рабочих продуктов ПС, которые находятся в репозитории, и устанавливает логику вариантности. На некотором уровне система является вариантом.

Управление конфигурацией СПС начинается с выбора КПИ в репозитории, проведения сборки и создания конфигурационного файла, необходимого для выполнения ПС или СПС в инструментальной среде ИТК. К операциям управления конфигурацией вариантов относятся функции *F*1—*F*4 и такие:

– идентификация элементов конфигурации и данных;

– определение функций управления процессом изменений;

– модели и методы вариабельности;

– метрики оценки вариабельности и др.

Это достигается в *модельной среде* конфигуратора для управления артефактами и вариабельностью ПС и СПС. В процессе управления конфигурацией собираются данные для проведения ряда процедур, включенных в план конфигурации:

– управление сборкой ГОР и инструментов построения ПС;

– управление процессами как гарантирование соблюдения плана развития ПС;

– управление средой — управление программным и аппаратным обеспечением, на которых размещена СПС;

– облегчение взаимодействия членов команды, которые работают над проектом СПС;

– отслеживание дефектов и обработка этих дефектов.

Среди операций выполняются стандартные операции - *отчетность и аудит конфигурации.*

На конфигураторе реализуется сборка ГОР и преобразование исходного кода в код для выполнения на компьютере. Одним из шагов ***сборки*** является процесс компиляции исходного кода, где файлы превращаются в промежуточный код или в код выполнения в среде VS.Net. Для сложных программ после компиляции устанавливается связь (нахождение реального положения всех внешних функций), которая выполняется специальной программой — связник. Процесс связи представляет собой замену относительных адресов функций внешних библиотек в реальные адреса, используемые в программе при выполнении. В этом случае конфигуратор является системой автоматизации и предоставления интерфейса пользователя для сборки ПС из набора ГОР в среде СПС.

Используются диаграммы характеристик для реализации конфигуратора ИТК. Диаграммы характеристик представлены терминами Work Flow MS Visual Studio и задают модель характеристик.

Модель характеристик — это дерево свойств системы, представленных диаграммой характеристик, которое используется во время всего процесса разработки ПС в СПС.

Диаграмма характеристик — это визуальное представление свойств модели характеристик с точами вариантности в виде дерева. Словарь терминов языка DSL включает термины, инструкции, технологии WF (Work Flow) и артефакты.

Таким образом, метод ОКМ обобщает понятие объектов, как элементов реального мира с их свойствами и характеристиками, которые определяются и уточняются с помощью математических формализмов отображения объектов ОМ в компоненты и интерфейсы. В нем конфигурационная сборка программных элементов основывается на формальных моделях компонента, среды и системы.

1.5. Основные положения компонентного программирования

Компонентное программирование разработано в в докторской диссертации Грищенко В.В. «Теоретические и прискладные основы компонентного программирования.-ИК НАНУ.-2007.-37с. и отражено в книге Лаврищева Е.М., Грищенко В.Н. Сборочное программирование.- Основы индустрии прораммных продуктов.- Киев.-Наук.Думка., 2009.-371с.

Разработка ПС и ПТС основывается на множестве компонентов $C = (c_1, c_2, \ldots c_r)$, где $c_i \in C$ – произвольный компонент множества С.

Каждый компонент характеризуется некоторой совокупностью свойств (например, расширяемость, взаимозаменяемость, заменяемость и др.).

Пусть $P_1, P_2, \ldots, P_k$ – предикаты свойств компонентов С на множестве компонентов - $P = < P_1(C) , P_2(C),... , P_k(C) >$.

Над компонентами $c_i \in C$ *могут применяться* операции, результаты выполнения которых также являются элементами множества С. Тривиальными примерами таких операций могут быть объединение нескольких объектов в один или другой компонент.

Обозначим через Σ С, R – алгебру, содержащую множество компонентов С и операций R над компонентами С. Каждая операция имеет собственную арность в зависимости от ее семантики. В целом существуют операции, которые задаются на подмножестве целых, а также могут в качестве результатов использоваться подмножества (например, операции объединения, сборки, разделения).

Среди множества операций рассматриваются те, которые для своих результатов сохраняют свойства компонентов. Обозначим эти операции через $R = (R_1, R_2, \ldots R_m)$. Тогда для любых $c_1, c_2, \ldots c_r$, принадлежащих C и любой R_i , $i=1,2, \ldots m$ с учетом арности операции R_i и ее результата справедливо одно из двух выражений: $R_i (c_1, c_2, \ldots c_r) \in C$ или $R_i (c_1, c_2, \ldots, c_r) \notin$ С.

Множество C и операции $R_1, R_2, \ldots R_m$ определяют алгебру компонентов. Примерами реальных операций над компонентами могут быть операции расширения интерфейсов, рефакторинг и др.

Множество компонентов С, рассмотренное выше, является математической абстракцией. В реальном случае, имеется некоторое множество компонентов $c_i \in$ С.

Применив к этому множеству операции $R_1, R_2, \ldots R_m$, получим множество С`, которое будет называть замыканием множества С. Этим термином показано, что С` имеет максимально множество компонентов С и каждый $c_i \in C$.

Компонентная модель системы

Под *компонентной моделью* системы понимается такое абстрактное представление конструируемой системы, в которой элементами являются реальные компоненты, обеспечивающие реализацию отдельных функций системы и выполнение нефункциональных требований к системе. Для одной и той же системы существует множество компонентных моделей в зависимости от концепций моделирования с использованием компонентного подхода. Рассмотрим обобщенный уровень представления модели, используя функции компонентов и совокупность контрактов (интерфейсов) между ними.

Пусть $\prod A_i$ – множество интерфейсов, относящихся к определению функций компонентов. Каждому A_i можно сопоставить интерфейс I_i , который описывает интерфейс как клиент-серверное взаимодействие с соответствующими методами и структурами данных. В соответствии с этим каждому интерфейсу можно сопоставить пару In_i и Out_i, которые будем называть соответственно определяющим представлением I_i и реализующим представлением Out_i. Параметр In_i определяет условие и цель интерфейса со стороны клиента, а Out_i задает аспект реализации интерфейса со стороны сервера. После того, как все In_i и Out_i определены, их можно группировать в различных сочетаниях для элементов С.

Произвольная совокупность входных и выходных данных - In_i, Out_j, где $i \neq j$, *которые* определяют один и тот интерфейс. Каждая совокупность $C_i \cup Iv = \{C_i, I_v, Out_J, Inout_{ij}\}$ входит в модель компонента или является шаблоном компонента, содержащим некоторое множество определяющих и реализующих описаний интерфейсов. Модель компонентной системы имеет вид**:**

$M_{кс} = \{CLm\{Lm_1, \ldots, Lm_n\}, P\ \{P_i, \ldots, P_m\}, CLn\ \{In_1, \ldots, In_k\}, D_i)$,

где CL_m – компоненты из множества реализаций в языках L,

$P\{P_i, \ldots, P_m\}$ – множество предикатов, соответствующих процессам конфигурационной сборки КПИ в ПС с помощью компонентов СLm и интерфейсов In;

CLn – множество интерфейсов компонентов;

D_i – множество данных.

Модель $M_{кс}$ состоит из множества функций реализаций (КПИ, reuses), предикатов, интерфейсов и данных. Условие целостности ПС заключается в существовании для каждого компонента C_1 из C, имеющего исходный интерфейс CIn_1^u, компонента C_2 с соответствующим входным интерфейсом CIn_2^m, и контракт $Cont_{12}^{im} = (CIn_1^u, CIn_2^m, IMap_{12}^{im})$, входящих в C.

Процесс построения $M_{кс}$ включает в себя создание компонентной среды, определение начальных компонентов и определенное множество интерфейсов в соответствии с функциональными требованиями к ПС. Суть моделирования системы состоит в том, чтобы представить модель $M_{кс}$ такую, чтобы для любого элемента системы существовал компонент из С или он мог быть получен из С` посредством конечного числа допустимых операций компонентной алгебры.

Операции над компонентами в среде Интернет

При описании операций над компонентами используются математические операциями $\{\cup, \cap, \oplus, \lozenge, -$ и др.$\}$ для задания и обработки компонентов в компонентной среде. Операции, их свойства ассоциативности, коммутативности, аксиомы и теоремы приводятся в литературе. Тут только перечисляются основные операции:

– *добавления компонента* к компонентной среде;

– *удаления компонента* из компонентной среды;

– *замены компонента*;

– *сборка* компонентных элементов в среде Интернет.

Операция реализации ПС, СПС через интерфейсный посредник

Операция добавления *компонента* заключается в том, что множество интерфейсов компонента расширяется за счет новых входных интерфейсов, связанных с реализацией нового компонента. Эта операция является ассоциационной и коммутативной над множествами компонентов. Доказательство этих фактов вытекает из анализа множеств интерфейсов и множества реализаций, которые входят в состав соответствующих множеств компонентов. Эта операция обеспечивает целостность компонента. Реализация компонента проводится вместе с интерфейсом.

Операция *расширения* существующей реализации семантики эквивалентных операций замещения, где исходная реализация замещается, а расширенная – добавляется. Потому нет необходимости вводить отдельную операцию.

Операция добавления интерфейса выполняется путем замены существующей реализации новой реализацией с сохранением целостности компонента, т.е. для каждого из входных интерфейсов обеспечивается реализация нового интерфейса.

Операция сборки компонентов проводится по стандарту конфигурационной сборки IEEE 828 (Cjnfiguration) операциями config, make, assembly в среде Grid, Etics, Java, MS.Net для получения конфигурационной структуры ПС, ПТС.

1.6. Веб-сервисы для создания прикладных систем и сайтов в Интернет

Для построения сетевых приложений (Web Application) разработаны сетевые службы (Web Services), доступные глобальным пользователям. За последние 10 лет комитет W3C разработал набор стандартов (протоколов) и языков описания программ, процессов и технологий для решения разного рода задач и бизнес-приложений в среде Интернет 0-12]. Далее рассматриваются *системные и прикладные сервисы*, которые представляются в среде систем CORBA, W3C, Grid, Etics, Java и др.

Системные сервисы и ресурсы

Системные сервисы обеспечивают решение разного рода прикладных, деловых и бизнес и экономических задач. К ним относятся:

– общие сервисы системных сред для поддержки процессов и функций обработки программ и данных (например, службы именования, каталогизации и др.);

– объектные сервисы, которые управляют объектами, классами и услугами по формированию и обработке объектно-ориентированных систем (например, службы диспетчеризации объектными запросами, управления интерфейсом и др.);

– сетевые сервисы стандартной модели OSI, моделей SOA (Service-oriented Architecture), SCA (Service-Component Architecture), как инструменты представления и обработки ресурсов в сети Интернет, которые реализуют деловые, финансовые, экономические и другие услуги при решении соответствующих задач;

– готовые программные и информационные ресурсы (services, artifacts, reuses, assets и др.), используемые как многоразовые услуги при решении разных задач в E-science и в других прикладных областях знаний.

Некоторые из сервисов стали обязательной частью общесистемных средств (VS.Net, IBM, Intel, Linux и др.), другие используются в специальных областях (например, медицина, биология) в плане предоставления услуг при работе с современными данными (FDT, GDT, Big Data).

Каждая служба определяется именем, по которому осуществляется поиск в распределенной среде пространства имен через транзакции, устанавливающие соответствие “имя-объект” для организации и управления отдельными сервисными ресурсами глобальной сети с помощью сообщений по визуализации общения с требуемыми отдельными сервисными ресурсами, которые используются при моделировании ПС и ПТС из готовых ресурсов сети Интернет.

В сети Интернет и E-science позволяют создавать прикладные системы ПС и веб-системы с помощью разных видов сервисов под упавлением Web сервера. В E-science эти функции выполняются с средствами **Grid, Eticx** (с 2000г.). Цель Grid - создание компьютерной инфраструктуры нового типа, обеспечивающей глобальную интеграцию информационных и вычислительных ресурсов, специального ПО и набора стандартизованных служб для обеспечения совместного доступа к географически распределенным ресурсам. Разработан принципиально новый подход к обработке огромных объемов экспериментальных данных и моделированию сложнейших физических процессов и созданию бизнес-приложений с большими объемами вычислений. Grid включает набор системных средств по реализации разных научных задач. Систем **Eticx** используется для построения сетевых приложений (Web Application) с использованием сетевых Web Services, доступных глобальным пользователям. За последние 10 лет комитет W3C разработал набор стандартов и языков описания программ, процессов, систем и технологий для реализации разного рода задач и бизнес-приложений в среде Интернет.

Глобальная информационная сеть (WEB-среда) объединяет десятки миллионов документов по всему миру и развивается благодаря стандартам консорциум W3C. Консорциум W3C объединяет наиболее крупных производителей программного обеспечения для WEB технологий (Web серверов и WEB браузеров). W3C обеспечивает компьютерным программам взаимодействие в сети, как сетевая интероперабельность.

Стандарты глобальной информационной сети подразделены на 4 группы.

1). Представления форматных данных (HTML, XHTML, CSS, MathML, SVG).

2). Представление структурированных данных (XML, XLink, XInclude, XSL, DOM и др.).

3). Протоколы удаленного выполнения программ и сервисов (SOAP, WSDL).

4). Представление семантических данных (RDF, OWL средствами онтологии Семантик -Веб).

Для задания представлений новых ПС и ПТС используются языки WWW3C:

WSCI (Web Services Choreography Interface),

WSCL (Web Services Conversation Language),

BPMN (Business process, model and notation),

BPEL (Business Process Execution Language) for Web Services и др.

Системные сервисы CORBA

Полный набор сервисов для обработки объектов содержится в системе CORBA (см. 1-6). В ней впервые реализована объектная модель и системные сервисы для работы с объектами:

– брокер объектных запросов (***Object Request Broker* — ORB**), обеспечивающий взаимодействие объектов в разных языках программирования;

– общие объектные сервисы (***Common Object Services* — COS**), обеспечивающие сервис управления изменениями, реализациями и контролем объектов,

– транзакциями, подпроцессами и т.п.;

– общие средства обслуживания (***Common Facilities* — CF**) услуги, предоставляющие прикладные функции для объединения в различные конфигурационные структуры с учетом заданных требований (например, средства печати, работы с БД, электронная почта и др.);

– объектные приложения (***Application Objects* — AO**), к которым относятся приложения и их компоненты, реализующие задачи и объекты пользователя, функционирующие в объектной среде, и над которыми могут производиться операции типа - открыть, инсталлировать, переместить и поместить.

Система CORBA обеспечивает взаимодействие удаленных объектов в распределенной среде типа OpenStep. Для задания взаимодействия объектов используется язык интерфейсов IDL (Interface Definition Languages). Интерфейсы в IDL запоминаются в репозитории интерфейсов (*Interface Repository*), а реализации объектов — в Репозитории реализаций (*Implementation Repository*). Независимость интерфейсов от реализаций объектов позволяет использовать их разными приложениями статически и динамически.

Объект-клиент и объект-сервер обмениваются между собой с помощью запросов, каждый из которых исполняется брокером ORB с помощью компонентов, создаваемых на основе описания интерфейсов клиента, сервера и ядра ORB.

Интерфейс клиента (*Client Interface*) обеспечивает взаимодействие с объектом-сервером и состоит из трех базовых интерфейсов:

– stub-интерфейса, содержащего описание внешне входных параметров и операций объекта в IDL;

– интерфейс динамического вызова (Dynamic Invocation Interface — DII) объекта, определяемого во время выполнения программы клиента путем поиска описания интерфейса в Репозитории интерфейсов или в Репозитории реализаций объектов;

– интерфейс сервисов ORB (ORB Services Interface), содержащего набор сервисных функций, которые клиент запрашивает у сервера через брокер ORB.

Система заняла видное место среди других систем, как инструмент взаимодействия разных систем между собой через брокер ORB брокера. Эти средства используются при создании конфигурационной структуры ПС и ПТС из готовых ресурсов.

Семантические ресурсы Интернет для моделирования ПС и ПТС

Семантик-Веб в (World Wide Web) содержит семантические ресурсы, языки, средства и инструменты разработки онтологий систем и бизнес-процессов с использованием накопленных знаний в виде ресурсов Reuses, КПИ и др. Семантические ресурсы Веба используют исследовательские программы институтов США и Евросоюза, а также фабрик больших и малых систем, проектов с открытым кодом (http://semwebprogramming.org).

Семантик Веб предоставляет средства для автоматизированной обработки научных задач, больших данных в разных форматах, интеграцию данных из коллажей (Mash-Ups),

поиск и композирование веб-сервисов, управление интеллектуальными агентами в мобильных приложениях и др. Обеспечивается решение задач с использованием многочисленных сервисов и научных достижений в области предоставления бизнес-услуг. Для использования больших объемов информации используются новые словари и концептуальные модели онтологизации научных задач и приложений с помощью предметно-ориентированных языков (OWL, DSL и др.) и инструментов (FODA, Protégé, DSLTool и др.).

К основным направлениям создания концептуальных моделей различных доменов относятся:

– онтологические модели, методы и средства доступа к мировым информационным ресурсам;

– модели описания готовых ресурсов как повторно используемых знаний;

– библиотек онтологии областей знаний;

– методы и средства создания комплексов программ, инструментов и формирования различных вариантов онтологии;

– использование методических и учебных материалов по применению Семантик Веб для описания профессиональных моделей и систем.

К средствам описания онтологий доменов, систем относятся: метапонятия - классы, факты, аксиомы, фасеты, слоты и др. Между ними могут быть следующие виды отношений:

– *обобщение* как сужение существенных признаков понятия с расширением круга понятий объектов и их объема;

– *конкретизация*, как добавление существенных признаков, благодаря чему содержание понятия расширяется, а объем сужается;

– *агрегация*, как объединение ряда понятий в новое понятие, существенные признаки нового понятия могут быть или сумме признаков компонентов или существенно новыми;

– *ассоциация*, как наиболее общее отношение, что утверждает наличие связи между понятиями, не уточняя зависимости их содержания и объемов.

Анализ новый инструментов и средств Семантик Веб дает разработчикам проекта возможность использовать отдельные языковые и инструментальные средства для моделирования доменов разного назначения в современной среде Интернет.

Заключение к заявке Проекта РФФИ 352-2016

Разработан объектно-компонентный метод (ОКМ) для моделирования систем с обеспечением вариабельности ПС и ПТС. Этот метод использовался при создании инфраструктуры информатизации НАНУ (2010-2013). Сформировался вариант метода ОКМ, который отработан на компонентах и защищен в докторской диссертации Грищенко В.Н.* Затем была доработана объектная часть метода моделирования с участием аспирантов, отдельные фрагменты которой защищены в диссертациях (Колесник А.Л.- 2013 и Стеняшин А.Ю. -2016 и др.) и опубликованы статьи:

– Грищенко В.Н.Компонентный метод моделирования технических и программных систем. Автореф. док.дисертации физ.-мат.наук.-2007.-К.:ИК АН НАНУ.- 41с. диссертация 320 с.

– Лаврищева Е.М., Колесник А.Л., Стеняшин А.Ю. Объектно-компонентное проектирование программных систем. Теоретические и прикладные вопросы – Весник КГУ, серия физ.-мат. наук, 2013. – №4. – С. 103 – 117.

– Lavrischeva E.M. Object-component Development of Application and systems. Theory and Practice» в журнале USA “Software Engineering and Application” (2014). Данная статья получила признание не только в Украине, в США и в Лондонском журнале “London press” 4 .11.2016. Лаврищева Е.М. стала членом «Quarterly Franklin Membership» (ID #5134421UK) с

провом бесплатно публиковать по этой тематике статьи в журнале «London Journal of Research in Computer Science and Technology» (LJCST);

– в 2015-2016 проведено усовершенствование метода ОКМ под задачи ОС, получен конечный вариант теории ОКМ, который опубликован в препринте «Теория объектно-компонентного метода моделирования систем» в ИСП РАН, 2016.- 48с. Препринт переведен на английский язык и опубликован в LJCST UK в 2017 году.

Публикации по I подразделу проекта РФФИ 352-2016

А. Лаврищева Е.М. Программная инженерия. Парадигмы, технологии и CASE-средства.-Учебник, 2 издание.-Москва, Юрайт, 2016 (https://biblio-online.ru/book/DCE62C40-BE54-4478-9BA5-7BE6200A8967).

Аннотация. В монографии рассмотрены парадигмы, технологии и Case-средства для разработки сложных компьютерных систем из программных ресурсов разных парадигм программирования. В *первом разделе* даны базовые понятия программной инженерии и метода сборки разноязычных модулей в сложные системы, а также средства автоматизации и реинженерии ресурсов и систем. Во *втором разделе* приведены новые формальные механизмы парадигм программирования (модульной, объектной, компонентной, аспектной и сервисной). Дано определение метода сборки ресурсов названных парадигм в сложные системы. *В третьем разделе* описаны технологии, линии изготовления элементов парадигм, их конфигурационной сборки, инженерии качества и CASE-средства поддержки парадигм и обучения языкам C#, JAVA, VBasic в среде веб-сайтов ИТК и фабрики программ КНУ. Для разработчиков и специалистов, занимающихся теоретическими и прикладными вопросами проектирования и реализации сложных компьютерных систем, а также для студентов высших учебных заведений по специальности программная инженерия, компьютерные науки и информатика.

Ключевые слова: компонент, объект, метод сборки, операции, качество, веб-сайт, Java.

В. Е.М. Лаврищева. Теория объектно-компонентного моделирования программных систем. Препринт ИСП РАН 29, 2016 г.- **с**. 1-48. (www.ispras.ru/preprintts/docs/prep_29_2016.pdf).

Аннотация. Представлен формальный аппарат объектно-компонентного моделирования предметной области, включающий логико-математическое описание на четырех уровнях проектирования объектной модели. На каждом уровне моделируются и уточняются объекты, устанавливаются их связи (интерфейсы) и отношения, базовые характеристики функций и логика поведения объектов. Объекты и их интерфейсы группируются в классы и отображаются в графе объектной модели. Объекты переводятся к виду компонентов с сохранением связей. Предложены формальные модели (компонента, интерфейса, среды и системы) компонентной среды, обеспечивается технический перенос объектов в компонентную среду и сборка объектов и компонентов стандарта конфигурацииэлементов в систему.

Ключевые слова: моделирование, логико-математический аппарат, предметная область, уровни моделирования, модели, объекты, граф, компоненты, изменяемость, многоразовые компоненты, сборка, конфигурация.

С. Лаврищева Е.М. Петренко А.К. Моделирование семейств программных систем. *Труды ИСП РАН*, 2016, том 28, вып. 5-6, 2016. ISSN 2220-6426 (Online), ISSN 2079-8156 (Print). DOI: 10.15514/ISPRAS-2016-28(3)-12

Аннотация. Анализируются подходы к моделированию систем, начиная с момента появления языка UML и ряда моделей MDA, MDD, MDE, PIM, PSM, SOA и др. Борьба со сложностью созданных больших прикладных систем привела к определению моделей характеристик MF (Feature Model), отображающих функциональные свойства элементов

системы и возможность их изменять, удалять и заменять новыми. Дается конкретизация современных подходов к моделированию сложных систем с помощью приведенных моделей. Определена семантическая сущность объектной теории моделирования изменяемых программных, операционных систем, их семейств, а также конфигурационной сборки программных продуктов (ПП) в (Product Lines/Product Families). Рассматриваются задачи управления вариабельностью и вариантами систем и их семейств с учетом требований заказчика. Проводится анализ моделей операционных (на примере Linux) и моделей программных систем (на примере генерационной GDM модели). Для них определены задачи верификации, тестирования, трансформации и извлечения (Minning) элементов из Legasy Systems для изменения и замены. Сформулированы цели проекта РФФИ 16-01-00352 по моделированию операционных, прикладных и Веб-систем в среде Интернет.

Ключевые слова: моделирование; проверка моделей; уточнение моделей; вариабельность; верификация; тестирование; семейство систем; управление моделями; управление вариабельностью, конфигурационная сборка,

D. Лаврищева Е.М., Карпов Л.Е., Томилин А.Н. Семантические ресурсы для разработки онтологии научной и инженерной предметных областей // Научный сервис в сети Интернет: труды XVIII Всероссийской научной конференции (19-24 сентября 2016 г., г. Новороссийск). — М.:ИПМ им. М.В.Келдыша, 2016. — С. 223-239. — URL: http://keldysh.ru/abrau/2016/16.pdf

Аннотация. Представлены основные положения и ресурсы онтологии семантической сети для определения концептуальных моделей сложных предметных областей и сетевых приложений. Ресурсами являются предметно-ориентированные языки описания онтологий (OWL, DSL, CPL и др.) и bycnhevtyns их поддержки (FODA, Protégé, DSL Tool VS.Net, JavaEE и др.). Представлена онтология предметных областей – вычислительная геометрия и домена программной инженерии – жизненный цикл стандарта ISO/IEC 12207. Представлен общий набор семантических ресурсов онтологии, включая средства предметно-ориентированного языка DSL, систем Protégé и DSL Tool VS.Net для представления этих предметных областей. Приведены граф-схемы этих областей.

Ключевые слова: семантическая сеть; онтология, модель онтологии, ресурсы онтологии, предметный язык, концептуальная модель, OWL, Protégé; бизнес приложения, задачи.

E. Ekaterina M. Lavrischeva. Assemblling Paradigms of Programming in Software Engineering.- 2016, 9, 2016.- p.296-317, http://www.scrip.org/journal/jsea, http://dx.doi.org/10.4236/jsea.2016.96021 Received 20 April 2016; accepted 24 June 2016; published 27 June 2016. Journal of Software Engineering and Applications, 2016, 9, 296-317.

Abstract Assembling paradigms programming are based on the reuses in any programming language (PL) with the passport data of their settings in WSDL. The method of assembling is formal and secures co-operation of the different reuses (module, object, component, service and so on) being developed. A formal means of these paradigms creation with help of interfaces is presented. Interface IDL (Stub, Skeleton) is containing data and operations for transmission data to other standard elements linked and describes in the standard language IDL. Assembling will be realized by integration of reuses elements in these paradigms on the instrumental-technological complex (ITC).

Keywords: Paradigm, Assembling, Theory, Interface, Reuses, Object, Component, Service, Program Systems, Interoperability, Multilanguages, Reengineering.

F. Лаврищева Е.М. Парадигмы программирования сборочного типа.- Труды Конференции УКРПрог-2014, № 2-3. – с. 121-134.

Аннотация. Представлен формальный аппарат парадигм программирования сборочного типа – модульного, объектного, компонентного и сервисного программирования. Теоретический аппарат каждой парадигмы обеспечивает формальную разработку отдельных программных элементов этих парадигм, определение интерфейсных элементов в

стандартном виде и их сборку на единой технологической основе в среде комплекса ИТК сайта http://sestudy.edu-ua.net.

Ключевые слова: парадигма, сборка, теория, интерфейс, повторное использование, объект, компонент, сервис, программа, система, взаимодействие, мультиязыки, реинжиниринг.

G. В.В. Кулямин, Е.М. Лаврищева, В.С. Мутилин, А.К. Петренко. Верификация и анализ вариабельных операционных систем. Труды Института системного программирования РАН, том 28, вып. 3, 2016, стр. 189-208. ISSN 2220-6426 (Online), ISSN 2079-8156 (Print). DOI: 10.15514/ISPRAS-2016-28(3)-12.

Аннотация. В данной работе рассматриваются проблемы верификации и анализа сложных операционных систем с учетом их вариабельности, наличия большого количества разнообразных конфигураций. Исследуются методы, позволяющие реализовать эти проблемы, провести их обзор и классификацию. Выделены классы методов, использующих для анализа инструменты, не учитывающие вариабельность, и выборки вариантов системы и методов, использующих специализированные инструменты, учитывающие вариабельность. Как наиболее перспективные с точки зрения масштабируемости, выделены техники анализа, использующие выборки вариантов системы, обеспечивающие покрытие кода и комбинации значений конфигурационных параметров, а также специализированных, учитывающие вариабельность кода техники анализа с итеративным уточнением модели поведения системы на основе примеров. Созданный вариант системы конфигурируется, верифицируется и тестируется соценкой качества.

Ключевые слова: операционная система, семейство систем, модель вариабельности, верификация, статический анализ, проверка моделей, проверка типов, покрытие кода, покрывающий набор, интеративное уточнение модели, верификация, тестирование.

Подраздел II проекта РФФИ 352-2017. Общие подходы к моделированию вариабельных ОС

Анализ и сопровождение ПО современных операционных и Legacy Systems показывает, что ПО систем сложное и требует специальных мер для снижения трудозатрат. Одним из методов снижения затрат на создание и сопровождение сложных систем, решающих большое количество разнообразных задач, является создание вариабельных систем (или семейств систем, software product families, software product lines). Экономия здесь достигается за счет создания многократно используемых элементов нескольких систем с близким набором функций, предназначенных для различных групп пользователей, сокращая таким образом затраты на создание таких систем.

На сегодняшний день существуют многочисленные техники разработки вариабельных систем различных типов, включая и операционные системы (ОС). К ним относятся генерационная, конвейерная техника Чернетски, техника построения (building) отдельных готовых элементов (ГОР) с удаленным доступом и их конфигурационной сборки в рамках Европейского проекта GRID и отечественная техника построения модели характеристик MF на уровнях проектирования ОМ и метода управления по моделям MF и ОМ новыми вариантами систем из семейства по требования заказчика.

Основу разработки вариабельных систем составляет модель вариабельности (variability model) и механизмы обеспечения вариабельности (variation mechanism) для новых

систем, в том числе с извлекаемыми элементами ОС. Модель вариабельности MF задает пространство возможных вариантов систем заданного семейства систем. Обычно MF определяется набором характеристик (features) или конфигурационных параметров, множествами их возможных значений и некоторыми ограничениями на возможные их комбинации. Каждому вариант у системы соответствует некоторый набор значений всех характеристик. Механизмы управления вариабельностью обеспечивает возможность построения необходимых вариантов системы из набора создаваемых и сопровождаемых артефактов.

Механизмы обеспечения вариабельности разделим на следующие группы:

– интеграционные, основанные на разработке отдельных элементов для различных вариантов и дальнейшей их интеграции в разные комбинации элементов систем;

– генеративные, использующие готовые артефакты систем для генерации из них параметризованных элементов из общей основы, отвечающих требованиям задаваемого варианта системы;

– извлекаемые (variability Minning) и уточняющие семантику функциональных элементов, широко используемых Legacy Systems и OS (Intel, Linux, IBM, MS и др.) с целью их конфигурационной сборки в вариантную структуру для новых видов применения;

– условно компилируемые с помощью макросов #ifdef, #elif, #else в языках C/C++, позволяющие собирать код, объединяющий различные элементы из набора значений характеристик, относящихся к набору параметров условной компиляции (макросов #define и #undef и параметров запуска препроцессора);

– смешанные, объединяющие названные техники обеспечения вариабельности создаваемых новых вариантов систем на современных архитектурах компьютеров.

В области операционных систем - ОС (под операционной системой понимается ядро и базовые библиотеки ОС, предоставляющие приложениям интерфейсы для работы с различными вычислительными ресурсами и аппаратным обеспечением) и выступающие в качестве механизма вариабельности смешанного типа с использованием условной компиляции указанных языков C/C++. Одним из методов проверки правильности сложных систем является верификация моделей и систем, учитывающая вариабельность и наличие большого количества разнообразных конфигураций ОС, а также тестирование созданного программного продукта из готовых ГОР и отдельных подсистем семейства.

Операционные системы и подход к моделированию вариантов ядра OS Linux

Современные операционные системы ОС имеют большую сложность. Согласно проведенных исследований, ядро Linux версии 2.6.32 имеет 6319 характеристик, более 10000 ограничений, которые могут задействовать до 22-х отдельных характеристик. При этом большинство характеристик ОС зависит как минимум от 4-х других, а максимальная глубина дерева зависимостей равна 8. Такая сложность приводит к большому количеству ошибок, связанных, прежде всего, с трудностями учета всех факторов, которые должен принимать во внимание разработчик отдельного элемента кода в рамках этой ОС. Соответственно, для выявления и преодоления разного рода ошибок необходимо использовать специализированные техники анализа и верификации. Анализ, артефактов заключается в рассмотрении особенностей ОС выбора допустимых элементов методом Data Mining и и одновременную проверку отдельных фрагментов и артефактов, из которых собирается выходной код варианта ОС или ПС. Накладываются особые требования на анализ сложных операционных систем и их ПО Эти требования специфичны для каждой ОС, которые описываются ниже.

2.1. Сущность подхода Data Mining, Variability Mining при извлечении артефактов из ОС

Процесс извлечения знаний The variability-mining process состоит из 4-шагов:

1. Определение экспертных моделей ОС, доменов и описание соответствующих характеристик и их отношений (функция locking).

2. Разработка начального смысла каждой функции в унаследованном кодовом ядре (т. е. во фрагменте кода, соответствующем выбранному объекту), используются методы операции Lock и Unlock.

3. Для каждой функции выявляются коды в виде последовательности функций, начиная с кода, относящегося к этой функции.

4. Перепись фрагментов кода таким образом, чтобы вариант системы был сгенерирован из этих фрагментов кода в виде системы.

Процесс mining может выполняться последовательно и с чередованием. Для того, чтобы выявить объекты из готовой системы и определить их отношения, необходимо начать извлечение отдельных объектов постепенно, а затем сформулировать их дополнительные функции. В рамках этого процесса выполняется третий этап: поиск всего кода функции. Остальные шаги являются далеко не тривиальными, но поддерживаются существующей концепцией и инструментами майнинга. На самом деле, необходимо найти всю реализацию объектов в Legacy коде с помощью инструментальной поддержки лействий разработчиков. Процесс **variability-mining** определяет фрагменты кода, при котором разработчики должны идти дальше и обновлять их всякий раз, пока дополнительная информация доступна. Например, изменение функций и их взаимосвязей, seeds, или когда разработчики аннотируют дополнительные фрагменты кода. В качестве известного программного инструмента используется система LEADT.

В этой системе можно выполнить ряд действий по выполнению Variability Mining:

– на 1-м шаге ставится задача общения разработчика с большим числом экспертов и зависит от специфики рассматриваемой системы;

– на 2-м шаге: для нахождения сборки элементов могут воспользоваться операциями Ctr-F. Могут применяться простые инструменты такие, как GREP и более сложные LSI и FLAT;

– на 3-м шаге никаких особых инструментов на данный момент пока не существует, поэтому надо разработать новые механизмы исследования функций системы и определения набора функций через проведения испытаний системы и для выполнения этих функций.

– на 4-го шаге необходимо использовать предпроцессоры С, такие как #ifdef и #endif.

При реализации 3 шага оказывает влияние следующие положения:

– согласовать код с извлекаемыми feature так, чтобы код работал как с ней, так и без нее;

– необходимо точно соотностись эти features участкам кода с высокой степенью детализации.

Знания о прикладной области проводятся экспертами с помощью традиционной concern-location техники и привлечения аппарата variability mining.

Для поддержки исследований на 3-м шаге используется инструмент для согласованного нахождения фрагментов кода данной feature. Предложенные шаги действий позволяют описывать элементы кода, features и отношения между ними в виде статического графа зависимостей, который отличается от традиционно принятой concern-location техники, используемой раннее для определения отдельных feature. В качестве механизмов извлечения знаний о свойствах и характеристиках могут использоваться такие методы, как байесовские модели, продукционные системы, семантические модели, методы дерева решений, кластеризация и др. Некоторые методы будут использоавать с участием студентов МФТИ и рекомендаций монографии «Технология разработки и моделирования изменяемых систем, Юрайт, 2016.

Анализ языков описания моделей и верификации ОС

В области операционных систем (и системного ПО) при описании моделей чаще всего используется язык Kconfig конфигурации ядра Linux с 2002 г. Язык CDL (Component

Definition Language) и ОС реального времени eCos используются для встроенных систем. Оба языка поддерживают все основные элементы метода FODA (Feature-Oriented Domain Analysis). Однако поддержка сложных ограничений, связывающих характеристики, не являющиеся элементами одного метода в дереве зависимостей, в инструментах, работающих с Kconfig. Исследования этих языков проведен В.Мутилиным в рамках проекта ИСП РАН (см. статью ниже)

Эти две модели характеристик Kconfig рассмотрены на [cite R. Zippel and contributors, kconfig- language.txt," available in the kernel tree at www.kemel.org, seen 2012-04/10 и Component. Definition Language (CDL) [cite B. Veer and J. Dallaway, "The eCos component writer's guide," seen Mar. 2010 at ecos.sourceware.org/ecos/docs-latest/ cdl-guide/cdl-guide.html.]. Обе модели используются в открытых операционных системах (IBM, VS.MS, Intel, Linux и др.). Они используются для описания вариабельности ОС Linux, и в десяти других проектах с открытым кодом. Язык CDL появился как часть eCos — операционной системы реального времени для встроенных устройств. ОС Linux и eCos имеют большой набор опций конфигурации с тысячами характеристик, что объясняет проблемы в процессе управлении вариабельностью.

Язык Kconfig был разработан для ядра Linux и применяется с 2002 года. Пользователю предоставляется конфигуратор xconfig, позволяющий выбирать характеристики в графическом интерфейсе. В настоящее время поддерживается и развивается в рамках ядра ОС язык CDL.

В языке Kconfig довольно сложно правильно описывается зависимость между характеристиками, так как инструменты, использующие Kconfig, не проверяют полностью их непротиворечивость. В выражениях зависимости между характеристиками можно включать любые другие характеристики с помощью конструкции SELECT. В этом случае разработчик должен убедиться, что конфигурация остается правильной. На практике, это зачастую не так и приводит к возможности выбрать конфигурацию, которая формально содержит противоречивые характеристики, и при этом «работает» у пользователя.

Как инструкции Kconfig, так и в обсуждениях зарубежных статей отмечается, что конструкция SELECT является причиной значительной части ошибок в конфигурации [cite R. Lotufo, S. She, T. Berger, K. Czamecki, and A. Wasowski, "Evolution of the Linux kernel variability model," in SPLC, 2010.]

Ошибки в описаниях Kconfig могут приводить к невозможности выбора некоторых опций, а значит невозможности покрыть некоторые участки кода. Y. Xiong, A. Hubaux, S. She, and K. Chamecki, "Generating range fixes for software configuration," in ICSE, 2012.

Оба языка конфигурации Kconfig и CDL поддерживают сравнительно небольшой набор операций анализа по сравнению с тем, что имеется в академических языках [см. D. Benavides, S. Segura, and A. Ruiz-Cortes, "Automated analysis of feature models 20 years later: A literature review," Information Systems, vol. 35, no. 6, pp. 615-636, 2010.]. Это показывает, что имеются существенные возможности для развития данных языков).

Дополнительные исследования этой проблемы провел Мутилин В.А в статье Статический анализ верификации для задач ОС. – ИСП РАН 2017.07.17. –с. 21-30. В этой статье приводятся ряд замечаний к статическому методу анализа вариабельности.

2.2. Анализ генерации вариантов конфигурации ОС, ПС

Генерация конфигураций ОС и ПС основывается на тестировании программ [cite C. Nie and H. Leung. A Survey of Combinatorial Testing. ACM Comput. Surv., 43(2): 1-29, 2011]. Из-за большого числа характеристик и сложности реальных систем (количество вариантов растет экспоненциально от количества конфигурационных опций) перебор всех вариантов оказывается невозможно на практике. Разработчики, как правило, анализируют некоторое подмножество вариантов, называемых типовым множеством (sample set), используя подручные инструменты. Вопросы анализа вариантов, выборки вариантов, их формального

описания и варификации проводились Мутилиным В.И. и Кулямином В.В. Выборка производится на основе эвристики выбора, либо вручную экспертом, либо с помощью некоторого алгоритма. Исследователи и практики предлагают разные варианты эвристик выбора. Рассмотрим четыре эвристики, используемые на практике:

– единственная конфигурация (Single-conf);

– случайная (Random);

– покрытие кода (Code-coverage);

– попарное покрытие (Pair-wise).

Другие стратегии выборок можно найти в [cite A Survey of Combinatorial Testing],

Единственная конфигурация

Наиболее простой эвристикой выбора является подбор представительного варианта конфигурации, который включает большинство или все характеристики вариабельной системы. Обычно этот вариант подбирается экспертом вручную. Преимущество такой эвристики в том, что требуется анализировать лишь один вариант, а, следовательно, это быстро.

С помощью выбора большего набора характеристик, можно покрыть как можно больше кода системы, но принципиально не можем покрыть взаимоисключающие части или нетривиальные взаимосвязи, специфичные для некоторых комбинаций характеристик.

В соответствии с работой Dietrich [cite C. Dietrich, R. Tartler, W. Schroder-Preikshat, and D. Lohmann. Understanding Linux Feature Distribution. In Proc. AOSD Workshop Modularity in Systems Software (MISS), pages 15-20. ACM, 2012.] в сообществе разработчиков ядра Linux распространенной конфигурацией для анализа является allyesconfig, в которой включено большинство характеристик. Аналогично многие программные системы поставляются некоторыми конфигурациями по умолчанию, удовлетворяющими большинство пользователей, и часто включающими довольно много характеристик.

Случайная эвристика

Наиболее простым подходом к перебору вариантов является их случайный выбор. Например, если в проекте имеется n характеристик, мы можем выбирать n случайных независимых значений по включению соответствующих характеристик. В проектах, где имеются ограничения на взаимосвязи характеристик, нужно исключать неправильные конфигурации, а оставшиеся верифировать. Верификация может продолжаться до исчерпания времени или ресурсов.

Эвристика по покрытию кода

Эвристика по покрытию кода основывается на критериях покрытия кода из области тестирования. В отличие от тестирования, эвристика покрытия кода нацелена на выбор конфигураций, а не на выполнение кода [cite R. Tartler, D. Lohmann, C. Dietrich, C. Egger, and J. Sincere. Configuration Coverage in the Analysis of Large-scale System Software. SIGOPS Oper. Syst. Rev., 45(3):10-14, 2012]. Цель эвристики — выбрать минимальный набор вариантов конфигураций так, чтобы каждый фрагмент кода системы включался по крайней мере в один из вариантов. В отличие от эвристики с единственной конфигурацией, эвристика по покрытию позволяет покрывать взаимоисключающие фрагменты кода. Однако включение каждого фрагмента кода хотя бы один раз не гарантирует того, что будут покрыты все комбинации фрагментов.

Дать обзор (анализ) структуры выходного кода с MV (конфигурационный файл до и после MV). Тестирование - этот второй этап проверки правильности кода.

Рассмотренные ниже методы не связаны или нет с моделью вариабельности, вернее точками вариантности. Проведя эти виды тестирования (один из), надо набрать информацию по обнаруженным ошибкам (по приведенным подходам) в коде и надо уточнить MV т.е. характеристики функций с ошибками и отметить в МК для повторной сборки.

Одним из инструментов, в которых реализован подход подбора конфигурации по покрытию кода является VAMPYR (Вампир). Имеются результаты его применения к ядру ОС Linux версии 3.2, набору программ Busybox и ядру L4/FIASCO [cite VAMPYR: Static Analysis of Variability in System Software: The 90,000 #ifdefs Issue].

На ядре ОС Linux лучший результат был получен для архитектуры mips с процентом покрытия 91, тогда как в конфигурации allyesconfig покрывается всего 55%. Для архитектур arm и x86 достигнутое покрытие гораздо меньше 84% и 88%, тогда как allyesconfig дает 60% и 79% соответственно.

Причины того, что не достигается полное покрытие лежат, во-первых, в недостатках реализации инструмента VAMPYR. Во-вторых, в несогласованностях Kconfig описаний.

Эвристика попарного покрытия

Эвриктика попарного покрытия основывается на наблюдении, что большинство ошибок в программных системах вызваны взаимодействием не более двух характеристик [cite M. Calder and A. Miller. Feature Interaction Detection by Pairwise Analysis of LTL Properties: A Case Study. Formal Methods in System Design, 28(3):213—261, 2006.

D. Kuhn, D. Wallace, and A. Gallo. Software Fault Interactions and Implications for Software Testing. IEEE Trans. Software Engineering, 30(6):418-421, 2004.

G. Perrouin, S. Oster, S. Sen, J. Klein, B. Baudry, and Y Traon. Pairwise Testing for Software Product Lines: Comparison of Two Approaches. Software Quality Journal, 20(3-4):605-643, 2012 N. Siegmund, S. Kolesnikov, C. K"astner, S. Apel, D. Batory, M. RoseimT'uller, and G. Saake. Predicting Performance via Automated Feature-Interaction Detection. In Proc. Int. Conf. Software Engineering (ICSE), pages 167-177. IEEE, 2012.] При использовании этой эвристики требуется найти наименьший набор конфигураций, в которых содержатся все пары характеристик. Вычисление попарных наборов становится нетривиальным, при наличии зависимостей между характеристиками. Задача является NP-полной (также, как и вычисление минимального покрывающего набора) [cite M. Johansen, 0. Haugen, and F. Fleurey. Properties of Realistic Feature Models Make Combinatorial Testing of Product Lines Feasible. In Proc. Int. Conf. Model Driven Engineering Languages and Systems (MODELS), pages 638-652. Springer, 2011].

Формальное описание задач ПС, СПС средствами Monadic Second-OrderLogic*

Monadic Second-OrderLogic содержит язык описания задач для последовательного и параллельного программирования, основанного на графовых схемах (traces) задач в биологии и в других приложениях. Задачи описываются с помощью математических операций объединения, замыкания и сравнения. Графовая структура задачи — это дерево, по которому проводилось формальное описание задачи. Язык формального описания объектов по дереву называться *древовидным языком с* трансдукционными (transductions) терминами (https://hal.archives-ouvertes.fr/hal-00646514): Bruno Courcelle, Joost Engelfriet. Graph structure and monadic second-order logic. A language-theoretic approach. — Cambridge University Press, 2011. — 742 с.

Контекстно-свободная грамматика языка НА использовалась для описания конечных автоматов, а метод лексического анализа сопоставлялся с каркасом описания. Алгоритмы описания задач задается монадической логикой второго порядка, включая реляционные структуры данных. Формальное описание задачи подвергалось валидации и верификации. Монадическая логика второго порядка используется для определения и классификации наборов слов в терминах этого языка и последующего распознавания слов конечным автоматом.

Каждый язык, который определяется средствами монадической логики второго порядка (включая свойство слов) может быть распознан конечным автоматом, и наоборот. Это означает, что монадическую логику второго порядка можно рассматривать как язык

спецификаций высокого уровня для компилирования в «machine код» конечным автоматом с распознанием слов, удовлетворяющих спецификации в монадической логике. Тот же результат справедлив и для членов конечных автоматов на деревьях. В результате монадическая логика второго порядка в настоящее время стала одним из основных инструментов, используемых в теории формализации языков описания приложений, контекстно-свободных грамматик, a finite автоматов и конечных преобразователей a finite автомата с выходом.

При компиляции этот язык использует атрибутивные грамматики, не содержащие контекста, снабженные семантическими правилами. Их правила связывают описания графа (зависимого графа) с деревьями вывода. Атрибутная грамматика на самом деле является примером контекстно-свободной грамматики графов (на основе замены hyperedge правилами переписывания. Семантика параллелизма представляется графами с применением конечных автоматов.

Одной из важных позиций теории этой логики является робастная теория контекстно-свободной грамматики графов с алгебраической формулировкой конечных автоматов и трансдукции графа. Теория контекстно-свободных грамматик используется в произвольных алгебрах Мезея и Райта и операциях на графах. Таким образом, графовое описание алгоритмов приводит к описанию элементов, которые строятся с помощью этих конечных операций. Контекстно-свободная грамматика графов — это конечный набор правил вида A0 ! f (A1; : : : ; An), где каждый Ai является нетерминальным символом грамматики и f — одна из выбранных графовых операций. Это правило означает, что, если графы G1, …, Gn генерируются соответственно к A1, ... An, где A0 может генерировать граф (G1, …, Gn).

Главная цель языка спецификации графов с помощью распознавания графов без контекста графов и графовых преобразований. Это дает новый фундаментальный результат для слов, терминов и свойств графов, которые могут быть указаны в языке Логики монадики второго порядка. С использованием этих спецификаций можно построить конечный автомат в терминах и преобразованиях спецификации дерева. Логика описания графов используется для представления автоматов и датчиков в терминах графа. Таким образом, монадическая логика второго порядка рассматривается как спецификация конечных автоматов на графах. Данный язык начал использоваться в проекте РФФИ при моделировании изменяемых вариабельных систем:

Создание виртуальных кластеров, виртуальных кластеров APACHeSpark в облачных средах с использованием средств оркестрации. - О.Д.Бондаренко, Пастухов Р.К., Кузнецов С.Д. Труды ИСП РАН том 28., выпуск 6.- с 111-120.-Москва,

Моделирование семейств программных систем. - Е.М.Лаврищева, Петренко А.К. Там же. С.49-53.

2.3. Верификация моделей вариабельности ПС и ОС

Модель верификации (статика) вариабельной MV с точками вариантности в модели конфигурации (МК), а потом анализ варификации. Анализ вариабельности использует фактор одинаковости вариантов систем между собой и ускорения верификации исходя из приведенных работ:

- T. Th"um, S. Apel, C. Kfastner, M. Kuhlemann, I. Schaefer, and G. Saake. Analysis Strategies for Software Product Lines. Technical Report FIN-004-2012, University of Magdeburg, 2012. Variable abstract syntax trees, Variability-aware type checking, Variable control-flow graphs, Variability-aware liveness analysis Principle: Keeping variability local. Предлагается вариант языка Java Featherweight Java.
- S. Apel, C. Kastner, A. Gr"oBlinger, and C. Lengauer. Type Safety for Feature-Oriented Product Lines. Automated Software Engineering, 17(3):251-300, 2010.
- Belaware, W. Cook, and D. Batory. Fitting the Pieces Together: A Machine-Checked Model

of Safe Composition. In Proc. Int. Symp. Foundations of Software Engineering (FSE), pages 243-252. ACM, 2009.
- S.Chen, M. Erwig, and E. Walkingshaw. An Error-Tolerant Type System for Variational Lambda Calculus. In Proc. Int. Conf. Functional Programming (ICFP), pages 29—40. ACM, 2012.
- B. K"astner, S. Apel, T. Th”um, and G. Sake. Type Checking Annotation-Based Product Lines. ACM Trans. Software Engineering and Methodology, 21(3):1—39, 2012.
- S. Thaker, D. Batory, D. Kitchin, and W. Cook. Safe Composition of Product Lines. In Proc. Int. Conf. Generative Programming and Component Engineering (GPCE), pages 95-104. ACM, 2007.

Анализ моделей вариабельности и верификации ОС

В рамках проекта РФФИ 16-01-00352 был проведен глубокий и многосторонний анализ моделей вариабельности, верификации ОС и представлен в статье В.В. Кулямин, Е.М. Лаврищева, В.С. Мутилин, А.К. Петренко. Верификация и анализ вариабельных операционных систем. Труды Института системного программирования РАН, том 28, вып. 3, 2016, стр. 189-208. ISSN 2220-6426 (Online), ISSN 2079-8156 (Print). DOI: 10.15514/ISPRAS-2016-28(3)-12.М.

Были проведены исследования зарубежных работ (см Berger T., She S., Lotufo R., Wąsowski A., Czarnecki K. A study of variability models and languages in the systems software domain. IEEE Transactions on Software Engineering, 39(12):1611-1640, 2013. DOI: 10.1109/TSE.2013.34) и др. предложены методы анализа языков моделирования вариабельных систем из четырех групп формального анализа:

– пропозициональная логика;

– онтология предметных областей знаний в Семантик Веб;

– программирование в ограничениях;

– верификации моделей, компонентов и систем.

Далее дается их описание.

Анализ на основе пропозициональной логики

В рамках этого подхода ограничения модели характеристик транслируются в представляющие их логические формулы, которые затем анализируются с помощью решателей, инструментов дляавтоматического доказательства различных видов (на основе SAT, BDD и пр.) и инструментов для работы с формальными языками типа Alloy, B, Z.

Анализ на основе онтологий

Этот подход основан на трансляции модели вариабельности в модель онтологии. Например, трансляция в OWL DL (Ontology Web Language Description Logic), разрешимое подмножество языка OWL, обладающее достаточной выразительной мощностью. После трансляции становится возможным использовать автоматизированные инструменты анализа онтологий RACER.

Анализ на основе программирования в ограничениях

В этом подходе модель вариабельности транслируется в описание CSP (Constraint Satisfaction Problem), которая затем анализируется с помощью существующих инструментов программирования с ограничениями (constraint programming.

Анализ на основе проверки моделей

Преобразование ограничений модели характеристик и проверки моделей (model checking), которые затем решаются соответствующими инструментами.

Специализированные алгоритмы

Предлагаются специализированные алгоритмы для решения конкретных задач анализа моделей характеристик, например, с оценкой числа допустимых вариантов.

Использование инструментов и методов анализа

Анализу подвергаются отдельные варианты систем. Эта задача аналогична задаче выбора небольшого, но представительного множества ситуаций, которые будут

использоваться в тестах, возможных при работе тестируемой системы. Эта задача обычно решается при помощи выбора так называемого критерия полноты тестирования, или критерия покрытия. Так же и при решении первой логично сформулировать некоторый критерий покрытия пространства вариантов, достижение которого (т.е. проведение анализа для набора вариантов, удовлетворяющего этому критерию) по некоторым причинам можно считать достаточным для выявления всех существенных свойств и ошибок, характерных для всего набора возможных вариантов.

Такие методы можно назвать анализом выборки вариантов (sample-based analysis).

Модификация методов и инструментов анализа

При работе с несколькими вариантами методов и внесения в их работу необходимых изменений по анализу свойств сразу в несколько вариантов проверяемой системы, используется случай близости вариантов друг к другу. Для каждого выполнения анализа более крупные куски кода, которые уже могут быть компонентами (или группами компонентов) с четко выделенным интерфейсом и определенным поведением, анализ свойств которой можно адекватно выявить у системы. При этом возникает потребность критерия покрытия допустимых вариантов и для множества компонентов с точками вариации - критерий покрытия вариаций. Этот критерий также должен определенным образом гарантировать выявление всех существенных свойств и ошибок вариантов системы и обеспечить возможность выбора лишь небольшого количества групп, каждая из которых в итоге будет подвергнута анализу. Подобные методы иногда называют методами, учитывающими вариабельность анализом (variability aware analysis) и нацелены на семейство (family-based).

Обзор описанных в литературе методов анализа вариабельных ОС показывает, что оба указанных подхода используются в исследованиях и на практике, однако более активно развиваются методы первого типа, использующие критерии покрытия пространства вариантов. Данная техники пока не достигла масштабируемости, необходимой для поддержки анализа промышленной ОС, такой как Linux.

Существует еще один метод *анализа семейств систем,* нацеленный на характеристики (feature-based), проводимые таким образом, чтобы выявить свойства всех вариантов, обладающих заданным значением выделенной характеристики. Однако используемый для системного ПО механизм вариабельности (условная компиляция) крайне усложняет проведение подобного анализа при наличии многих характеристик, поскольку специфичные для заданной характеристики свойства трудно выделять на фоне сложного поведения с большим числом характеристик и их взаимосвязей.

Для такого анализа в Linux имеется инструмент их построения, *randconfig*, часто используемый при необходимости получить случайные конфигурации. Случайный выбор набора вариантов не гарантирует для всего анализируемого семейства или достижения определенного критерия покрытия. Критерии покрытия вариантов учитывают, насколько в выбранном наборе вариантов представлены различные фрагменты кода, которые могут быть включены или исключены при построении допустимого варианта системы Эти критерии используются при тестировании, — первые нацелены на выбор некоторого набора вариантов, содержащих какие-то комбинации фрагментов кода, а вторые на выполнение определенных частей кода.

Наиболее широко используемым критерием такого рода является требование того, чтобы каждый (актуальный) фрагмент кода из Репозитория программного семейства входил хотя бы в один из вариантов, отобранных для анализа. Несмотря на то, что это не самый сильный критерий — при его использовании могут быть не выявлены ошибки, проявляющиеся только при определенных комбинациях варьируемых фрагментов, — даже его достижение в сложных случаях сопряжено со значительными затратами.

Для выбора **набора конфигурации ядра Linux**, покрывающих как можно больше кода, используется специализированный инструмент VAMPYR, который дает получить максимальное покрытие кода 91% для архитектуры mips (для архитектур x86 и arm удалось

получить всего 88% и 84%). Использование одной конфигурации allyesconfig дало для этой архитектуры покрытие 55% кода (соответственно, 79% и 60%), что показывает, также, насколько на практике одна, даже «самая представительная» конфигурация может мало затрагивать возможные варианты поведения систем семейства. Для повышения этих значений могут понадобиться гораздо более сложные подходы, способные, например, обеспечить включение части кода, выделенного несколькими директивами #if/#ifdef.

Математической основой таких методов служит алгоритм построения покрывающих наборов (covering arrays), которые часто используются при конфигурационном тестировании. Покрывающий набор глубины t определяет матрицу значений, в которой столбцы соответствуют характеристикам (значениями в столбце могут быть только допустимые значения соответствующей характеристики), строки – выбираемым вариантам, и каждая комбинация t значений в любых разных t столбцах обязательно встречается в одной из строк.

Простейшим случаем является покрывающий набор глубины 2 (или попарный, pair wise), позволяющий покрыть в рамках набора вариантов, задаваемого его строками, все сочетания пар значений характеристик. Задача построения минимального покрывающего набора NP-полна, но существуют эффективные алгоритмы построения наборов, лишь немного больших, чем минимальные. Для сложных моделей вариабельности большей проблемой является удовлетворение всех налагаемых моделью ограничений, поэтому техники создания покрывающих наборов должны дополняться достаточно эффективными методами построения или выбора удовлетворяющих ограничениям конфигураций.

Методы анализа и верификации вариабельных систем, обеспечивают проверку сразу многих возможных вариантов одновременно, затрачивая большие затраты на нее за счет большого объема общего кода во всех вариантах.

Учет вариабельности требует при реализации инструментов анализа с помощью модифицированных деревьев абстрактного синтаксиса, размеченных условиями использования тех или иных узлов, представляющими собой пропозициональные формулы над равенствами характеристик и их возможных значений.

Опубликованные методы такого рода обычно относятся к методам проверки типов (type safety checking, well-formedness checking) или к методам проверки моделей (model checking), использующих дедуктивную верификацию (theorem proving) и символический мониторинг.

Наиболее перспективно с точки зрения масштабируемости из этих методов выглядят техники проверки типов, дающие возможность эффективно проверять простейшие свойства корректности, и техники проверки моделей с их итеративным уточнением, требующие минимального вмешательства человека при анализе большого по объему кода со сложным поведением. Остальные методы становятся чрезмерно сложными при учете реалистичных моделей вариабельности.

Таким образом, задачи верификации и анализа современных промышленных ОС проводятся с учетом их вариабельности или наличия большого числа возможных конфигураций. Основные проблемы – это трудности выявить значимые свойства при анализе отдельных фрагментов кода, из которых собираются эти варианты. На основе проведенного обзора таких методов, учитывающих большой объем кода и сложность современного системного ПО, а также нацеленность на проведение как можно более полного анализа поведения всех конфигураций системы, выбираются наиболее перспективных для дальнейшего развития – это хорошо масштабируемые техники, использующие выборку вариантов на базе покрытия кода и/или на базе покрытия различных комбинаций значений конфигурационных параметров, а также техники анализа кода, использующие итеративное уточнение моделей на основе контр примеров.

В инструментах, работающих с практически важными языками Kconfig и CDL, возможности анализа моделей весьма ограничены. Для Kconfig поискн согласованностей и недопустимости конфигураций выполняются автоматически только для моделей с ограничениями, не включающими характеристики, не имеющие общего родителя. В CDL

имеется средства анализа непротиворечивости конфигурации, основанная на пропозициональной логике. При модификации конфигурации инструмент анализа определяет противоречия и подсказывает пользователю способы их разрешения. В языках Kconfig и CDL имеется возможность для существенного развития описания конфигураций системного ПО.

2.4. Средства статической верификации ядра системы Linux

В рамках магистерской работы студентом МФТИ Гариповым И. проведено исследование ядра ОС Linux версии 3.2. Установлено, что оно содержит более 12 000 настраиваемых опций, которые контролируют включение 31 000 конфигурационно-зависимых файлов исходного кода, которые в свою очередь содержат 89 000 блоков #ifdef. Инструменты статического анализа могут существенно облегчить процесс поддержки качества баз кода таких объемов.

Однако конфигурирование значительно усложняет автоматизированное тестирования ПО и процесс поиска ошибок. Для корректной проверки, инструменты должны производить обработку одной конкретной конфигурации, поэтому программисты должны вручную извлечь множество конфигураций, чтобы гарантировать, что все конфигурационно-условные части кода проверяются.

Основные действия для выполнения статического анализа следующие:

1) Процесс сборки в Linux связан с созданием правил для последовательного их выполнения в директориях функций Makefile, способствующих компиляции соответствующих С-файлов и обработки параметров конфигурирования файлы могут быть компилированы в образ ядра, либо в загружаемые модули ядра (LKM).

2) Генерация варианта продукта проводится путем выбора конфигурационных опций из модели вариабельности, описывается в языке Kconfig. Каждая опция Kconfig имеет тип (bool, tristate, string, int), который определяет выбор значений для входа в Kconfig.

3) Kbuild скрипты в Makefile или Kbuild файлы запускают процесс сборки. Данные скрипты написаны на языке MAKE и распределены по коду ядра. Большая часть Makefile файлов в Linux находятся в подкаталогах опций, и просто отвечают за установку MAKE переменных obj-y, obj-m, и obj-n, тем самым сообщая top-Makefile файлу список файлов, которые должны быть построены либо статически, либо как загружают модуль ядра, или же не должны быть построены вообще.

4) Перевод Kconfig в пропозиционную логику двух типов конфигурационных переменных, один из которых заканчивается суффиксом _MODULE и передается для С-препроцессора и Makefile-файлов. Этот процесс вводит логические ограничения, которые задают неточности и осуществляется конфигурационно-зависимый дефект с помощью специальной опции.

Инструменты верификации моделей и диаграмм характеристик ПС

В этом пункте представлен перечень доступных инструментов моделирования характеристик ПрО с помощью разных инструментов верификации этих моделей.

Feature Model Plugin или FeaturePlugin **(FMP).** Инструмент реализован как плагин для Eclipse. Он поддерживает моделирование, построение модели характеристик МХ и конфигурирования СПС на ее основе. Использует BDD-решатель для определения количества возможных комбинаций характеристик в модели характеристик, но не имеет средств другого анализа характеристических диаграмм. Не поддерживает атрибутные модели характеристик (attributed feature models) и не может включать более чем один решатель для проведения анализа. Инструмент доступный по адресу – http://gsd.uwaterloo.ca/projects/fmp-plugin.

xFeature основан на XML инструменте моделирования характеристик, также реализованный как плагин для Eclipse. Поддерживает моделирование СПС и порождаемых ПС. Не поддерживает автоматического анализа модели характеристик.

Доступный по адресу – http://www.pnp-software.com/XFeature/ Download.html.

CaptainFeature. Это инструмент моделирования характеристик, который использует нотацию FODA для построения модели MX. Включает интегрированный конфигуратор для специализации созданных моделей. Не поддерживает автоматический анализ моделей. Доступный по адресу – https://ngfe1049srsid5978550sourceforge.ngfe1049srsid5978550 net/ngfe1049srsid5978550 projects/ngfe1049srsid5978550 captainfeature.

Requiline. Это инструмент инженерии требований для эффективного управления линиями программных продуктов. Из описания группы характеристик и требований «выводится» конфигурации продукта. Включает функцию контроля (checker) согласованности и не выполняет других операций анализа. Проводится в таких средах функционирования – Microsoft.NET, Oracle DBMS. Доступен по адресу http://www-lufgi3.informatik.rwth-aachen.de/TOOLS/requiline/ Product/download.php при наличии заявки по e-mail.

Pure::Variants. Инструмент поддерживает моделирование характеристик, конфигурирование, базовые операции анализа с помощью средств Prolog решателя с ограничениями (constraint solver). Инструмент доступен по адресу – http://www.software-acumen.com/downloads/.

AHEAD Tool Suite (ATS). Это набор инструментов для разработки линий продуктов поддерживает модуляризацию характеристик и их компоновку. Может выполнять определенные операции анализа модели характеристик посредством SAT-решателей и сохранять ее в виде грамматики. Доступен по адресу –

http://www.cs.utexas.edu/~schwartz/ATS.html.

Из представленных инструментов фактически только FeaturePlugin поддерживает практически анализ модели характеристик по адресу – http://gp.uwaterloo.ca/fmp2rsmVerifier/demo.htm – на примере (demo) использования верификатора FeaturePlugin.

Анализ этих инструментов в работе участника проекта РФФИ Мутилина И.С. Статический анализ верификации» проведен средствами config. показывает, что наиболее подходящий инструмент для работы с ОС – это **CaptainFeature**.

Заключение к подразделу II проекта РФФИ 352 -2017 и литература

Исследованы подходы к обеспечению вариантов операционных систем (ОС). Дана характеристика сформировавшихся подходов к организации процесса генерации новых вариантов ОС; определены методы проведения анализа систем с целью извлечения артефактов из действующих Legacy Systems (OS Linux, Intel Web-systems и др.); сформулирован подход к верификации объектов и вариантов конфигураций моделей, а также определен метод конфигурационной сборки функциональных артефактов ядра ОС Linux. Проведены эксперименты по обеспечению статического анализа и верификации отдельных артефактов, фрагментов ядра этой ОС, подготовлены бакалаврская и магистерская работы студентами по данной тематике. Определены новые положения, продиктованные спецификой ОС Linux, которые будут уточняться и развиваться в рамках усовершенствованного метода ОКМ, после проведения экспериментов. Опубликована статья «Верификация и анализ вариабельных операционных систем» в трудах ИСП РАН с участием членов группы проекта (Кулямина В., Мутилина В., Петренко А. и др.).

А. Лаврищева Е.М. Онтология по моделированию домена ЖЦ стандарта ISO/IEC 12207.

Аннотация. Рассмотрен онтологический подход к представлению модели жизненного цикла стандарта ISO/IEC 12207-2007, включая описание основных, организационных и процессов поддержки. Для процесса тестирования приведено онтологическое описание понятий, концептов и задач в терминах системы Protégé, результатом которой является общепринятый язык XML, по которому можно получить машинную программу. Эти онтологические описания процессов жизненного цикла являются базисом их автоматизации на любой платформе компьютера, особенно это касается процесса тестирования по программе, которой проведено проверка программ в языке Ruby.Описание процессов и реализации представлены в инструментально-технологическом комплексе на веб-сайте http://7dragons.ru/ru сайта ИСП РАН.

Ключевые слова: онтология, модель, понятия предметной области, процессы, основные

B. Lavrischeva Ekaterina. Ontological approach to the formal Spesification of the Standart Life Cycle. “Science and Information Conference -2015”, Jily 28-30, London, UK. www.conference-thesal.org.-p.964-972.

Abstract. Approach is offered to the formal specification of Standard Life Cycle (LC) of the program systems (PS) by the ontology facilities with purpose automation and generation of prgram systems (PS) by the ontology facilities with purpose automation and generation of the varants LC for making the appropriate kinds process for development different PS. Ontological aproach to presentation LC model of the standard ISO/IEC 12207–2007 is included the specification of geeral, organizational and support processes. These processes are presented in the subject-oriented DSL, which than transformed to XML for realization. One of the processes, the testing process is given in terms of Protégé systems. An eventual result of this system Protégé got generally at accepted to the XML, suitable for implementation tasks testing PS on computer.

Keywords: ontology, the life cycle standard, model of life cycle, processes, actions, task, testing, DSL, description, Protégé, XML, WSDL.

C. E. M. Lavrischeva. The theory graph modeling systems from quality modulesof the application areas.

Abstract: The graph modeling of applied systems (AS) from ready resources (modules) are presented. The graph isrepresented by an adjacency and reach ability matrix. A new program structures are modeling by mathematical operations (unions, connections, etc.). The assemble of programs structures (complex, system, packets, AS, OS, IS, etc.) from modules in LP,b as resources are integrating by such operations (link, make, weaver, config, etc.) and controlled on the quality every recourses and the making systems from them.

Keyword: graph, operation assembly, link, making, make, waver.

D. В.В. Кулямин, Е.М. Лаврищева, В.С. Мутилин, А.К. Петренко. Верификация и анализ вариабельных операционных систем. Труды Института системного программирования РАН, том 28, вып. 3, 2016, стр. 189-208. ISSN 2220-6426 (Online), ISSN 2079-8156 (Print). DOI: 10.15514/ISPRAS-2016-28(3)-12.

Аннотация. В данной работе рассматриваются проблемы верификации и анализа сложных операционных систем с учетом их вариабельности, или наличия большого количества разнообразных конфигураций. Исследуются методы, позволяющие преодолеть эти проблемы, проводится их обзор и классификация. Выделены классы методов, использующих для анализа инструменты, не учитывающие вариабельность, и выборки вариантов системы и методов, использующих специализированные инструменты, учитывающие вариабельность. Как наиболее перспективные с точки зрения

масштабируемости, выделены техники анализа, использующие выборки вариантов системы, обеспечивающие покрытие ее кода и комбинаций значений конфигурационных параметров, а также специализированные, учитывающие вариабельность кода техники анализа с итеративным уточнением модели поведения системы на основе контрпримеров.

Ключевые слова: операционная система, семейство систем, модель вариабельности, верификация, статический анализ, проверка моделей, проверка типов, покрытие кода, покрывающий набор, итеративное уточнение модели.

E. S.V.Kozin, V.S.Mutilin. Static Verification of Linux Kernel Configuration.

Abstract. The Linux kernel is often used as a real world case study to demonstrate novel software product line engineering research methods. It is one of the most sophisticated programs nowadays. To provide the most safe experience of building of Linux product line variants it is necessary to analise Kconfig file as well as source code. Ten of thousands of variable statements and options even by the standards of modern software development. Verification researchers offered lots of solutions for this problem. Standard procedures of code verification are not acceptable here due to time of execution and coverage of all configurations. We offer to check the operating system with special wrapper for tools analyzing built code and configuration file connected with coverage metric. Such a bundle is able to provide efficient tool for calculating all valid configurations for predetermined set of code and Kconfig. Metric can be used for improving existing analysis tools as well as decision of choice the right configuration. Our main goal is to contribute to a better understanding of possible defects and offer fast and safe solution to improve the validity of evaluations based on Linux. This solution will be described as a program with instruction for inner architecture implementation.

Keywords: Software Product Lines, Linux, Kconfig, Preprocessor. DOI: 10.15514/ISPRAS-2017-29(4)-14, 15.09.17.-с.21-30.

F. К Лаврищева Е.М., Петренко А.К. Подход к сборочному созданию варианта ядра OS Linux .-ИСП РАН 2017.- Том.28, выпуск 6.-р.49-65.

Аннотация. Ядро операционной системы Linux - это пример современных инженерных решений в области создания продуктовых линеек программного обеспечения. Сегодня это одна из наиболее сложных программных систем. Для того, чтобы обеспечить наиболее безопасное построение вариантов продуктовой линейки, необходимо анализировать конфигурационная операция Kconfig исходного кода. Ядро содержит десять тысяч вариабельных переменных несмотря на современную инженерию. Исследователи в области верификации предлагают большое количество решения проблемы анализа. Стандартные процедуры верификации здесь не могут быть применены из-за времени проверки покрытия всех конфигураций. Нами предлагается инструмент, который базируется на связи уже существующих программах для проверки кода и конфигурационного файла с метрикой покрытия. Такой пакет – это эффективный инструмент для расчета всех допустимых конфигураций для предопределенного набора кода и Kconfig. Предложенные методы могут быть использованы для улучшения существующих инструментов анализа, а также для выбора правильной конфигурации. Основная цель - лучше разобраться в возможных дефектах и предложить быстрое и безопасное решение для проверки ядра Linux. Это решение будет описано как программа с инструкцией по реализации внутренней архитектуры.

Ключевые слова: операции Kconfig, ядро ОС Linux, конфигурация, анализ, дефекты, код.

G. Lavrischeva E.M. Ontology of Domains. Ontological Description Sofware Engineering Domain—The Standard Life Cycle, Journal of Software Engineering and Applications, July 24, 2015.

H. Ekaterina Lavrischeva. Ontological Approach to the Formal Specification of the Standard Life Cycle, "Science and Information Conference-2015", Jule 28-30, London, UK, www.conference.thesai.org.- p. 965-972.

I. Ekaterina M.Lavrischeva. Ontology of Domains. Ontological Description Software Engineering Domain—The Standard Life Cycle, Journal of Software Engineering and Applictions, 2015, 8, **-**, p.1-15. Published Online July 2015 in SciRes. http://wwscirp.org/journal/jseai

К. Е.М. Лаврищева. Компонентная террия и коллекция технологий для разработки индустриальных приложений из готовых ресурсов. Труды Четвертой научно-практической конференции «Актуальные проблемы системной и программной инженерии», АПСПИ-2015, 20-21мая 2015, с 101-119.

L. Е.М.Лаврищева. Object modelling of subject domains, Заочный.журнал «Объектные системы», 2015, 6с.

М. Лаврищева Е.М., Карпов Л.В., Томилин А.Н. Системная поддержка бизнес задач в глобальной информационной сети.-Труды конференции «Научный сервис в сети Интернет -2015», 21-26 сентября 2015, Абрау-Дерсу.

N. Е. М. Лаврищева Л. Е. Карпов. Н. Томилин. Сервисные средства интернет для решения бизнес-задач, Труды ИМИ им. ИМ Келддыша.-Научный сервис.- том 5, 2015..

Подраздел III проекта РФФИ 352-2018. Метод анализа технологии программирования и ОС

Введение

Программирование ПО ОС начало развиваться одновременно с появлением ЭВМ путем ручного программирование общесистемного обеспечения для функционирования первых ЭВМ. Потом появился термин “Software Engineering - SE”, который прозвучал на конференции НАТО (1968) и обозначал инженерные аспекты изготовления вычислительной продукции (ЭВМ и программного обеспечения – ПО) высокого качества (www.sei.com). Постепенно на практике формировались процессы разработки ПО и систем, которые упорядочивали процессы программирования для определения отдельных элементов систем и математического обеспечения ОС решения научно-технических задач на ЭВМ.

В СССР создавались ЭВМ и ПО для них. Сформировалась технология программирования (ТП) прежде всего трансляторов с простых языков (типа адресного), языков программирования Алгол-60, Фортран, Кобол и др. для описания математических и технических задач, а также ОС для управления процессом вычислений этих задач. Кабинет Министров СССР принял постановление (1976) о фондах программ со статусом программной продукции производственно-технического назначения. ГКНТ СССР финансировал разработку средств автоматизации технологии создания сложных программ (ПРИЗ, АПРОП, ПРОЕКТ, ПРОТВА, ПРОМЕТЕЙ, АЛЬФА, и др.). В СССР были сделаны в ОС ЕС ЭВМ под руководством академика В.М. Глушкова систему автоматизации производства программ – АПРОП- 1976. Финансы и статистика.-137с. Система АПРОП была изготовлена и сдана ГосКомиссии в 1976г. и передана в ГосФАП в 1977г. В 1975г. В.М. Глушков был на конференции IFIP и после приезда высказал идею о конвейерной сборке программ, подобно конвейерной сборки автомобилей Форда. Эта концепция началась разрабатываться на АСУ ТП в ГДР (1975) для медицинских приборов. По договору с ВПК Липаев В.В. (1978-1985) руководил созданием программных комплексов Прометей, Яуза, Руза, Протва и более 100 приборов для авиации, морфлота и космоса с помощью системы АПРОП и книги Лаврищевой Е.М., Грищенко В.Н. Связь разноязыковых модулей в ОС ЕС.- 1982.-Финансы и статистика.-137с.

За рубежом в 80-х прошлого столетия созданы CASE-средства – SADT (Structured Analysis and Design Technique), SSADM (Structured Systems Analysis and Design Method), IDEF0, IDEF1, IDEF2 (Integrated Definition Functions). Позже сформированы , продуктовые линии (Software Product Line Engineering) коммерческого изготовления ПП, экспериментальные фабрики программ (Дж.Гринфильда, Г.Ленца, М.Фаулера и др.) и системные AppFab на платформах (IBM, VS.Net, Intel, CORBA, Java, Intel и др.) для сборки готовых элементов, шаблонов, кодов в системе автоматизации предприятий, бизнес приложений и др.

В рамках международного комитета ISO/IEC созданы стандарты, регламентирующие процессы разработки - жизненный цикл (ЖЦ) стандарта ISO/IEC 12207 (1996, 2002) и гарантии качества. Процессы стандарта соответствовали областям знаний SWEBOK. Технология использования процессов ЖЦ стандарта ISO/IEC 12207 - 2007 для проектирования и разработки программ, комплексов программ и систем традиционными методами представлена в ряде зарубежных монографий и учебных пособий В.В.Липаева, Лаврищевой Е.М. в МФТИ-2017, объектно-ориентированными методами – UML, Rational Rose и др.

В связи с накоплением в мировом информационном пространстве огромного количества готовых программ и сервисов (более 100мил.) для многоразового использования готовых ресурсов при разработки больших и сложных программных (ПС) и информационных систем путем сборки (IBM, Oberon, Unix) отечественная АПРОП и/или интеграция (Assembling, Build, Continious Integration и др.). Для сборки модулей была разработана и реализована книга «Связь разноязыковых модулей в ОС ЕС.-Лаврищева Е.М., Грищенко В.Н..- 1982., обеспечивающая сборку многоразовых ресурсов (модулей, компонентов, объектов, сервисов, reuses, assets и др.) через модуль посредник (stub, skeleton), описываемые в языке WSDL, а их интерфейсы в языках (IDL, API, SIDL и др.).

Проектирование систем традиционными методами SADT, SSADM и др. приводило к увеличению сложности систем и трудностям внесения в них изменений, что обозначено было кризисом сложности. Технология сборки сложных систем из готовых КПИ и их интерфейсов проводилась модульным принципом программирование приложений и систем. Использование многоразовых reuses упрощает процесс поиска и внесения изменений в сложные системы.

Сборочный принцип разработки систем и их семейств, присущий всем современным технологиям, обеспечивает комбинирование КПИ и их конфигурацию в новые варианты приложений и систем, способных выполняться в современных гетерогенных средах. Опубликован препринт по технологии сборки в МФТИ:

Е.М.Лаврищева. Технология сборки модулей, объектов, компонентов, интеллектуальных, , информационных и сервисных ресурсов Интернет.- 2018-2023. – препринт.-49с. Долгопрудный, Институтский пер.7 .rio@mipt.ru.

Начиная с 1990 годов, активно развивался объектно-ориентированный подход (ООП). Появились новые объектно-ориентированные ЯП (C++, VBasic, Java и др.) и CASE-средства (Rational Rose, UML, CORBA, COM и др.), системы поддержки процессов разработки объектных программ и оценки их качества. Они стали альтернативой традиционного программирования монолитных систем и способствовали представлению программных объектов в виде самостоятельных многоразовых продуктов со свойством изменяемости. Консорциум CORBA реализовал для ООП объектную модель, брокер объектных запросов через интерфейс к собранным объектам на сервере. Объекты описываются в ЯП этой системы. Rational Rose и UML предоставили пользователю богатый набор диаграмм (use case) проектирования систем из объектов и компонентов и инструменты их реализации, тестирования и управления качеством продуктов. Эти системы можно считать системными фабриками программ, основанными на использовании готовых КПИ.

С 2000-х в мировой индустрии сформировалась технология Software Product Line Engineering. Ее основу составляет стандартная модель архитектуры системы и модель характеристик (Feature Model), включающая базовые функциональные характеристики артефактов архитектуры домена и точки изменяемости артефактов приложений и семейства. Эта модель реализует свойство вариабельности готового программного продукта (ПП), обеспечивает автоматизированную обработку функций системы и возникающих нерегулярных ситуаций, выдавая сведения для внесения изменений в терминах значений характеристик и вариантных точек.

В 2001г. международный комитет IEEE и ACM определил ядро знаний SWEBOK (Software Engineering body of Knowledge, www.swebok.com, 2001г.) программной инженерии. В нем дано такое определение – *это система методов, средств и дисциплин планирования, разработки, эксплуатации и сопровождения ПО, готового к внедрению*. Ядро включало в себя 10 разделов знаний (area knowledge). Первые пять разделов – это инженерия требований, проектирование, конструирование, тестирование и сопровождение ПО. Следующие пять разделов – это организационные процессы (управление проектом, конфигурацией, качеством, методами и средствами инженерии ПО). Был предложен теоретический аспект SE, состоящий в классификации ее дисциплин, которые регламентируют инженерную, экономическую, управленческую и производственную деятельность процесса разработки программного продукта (ПП). Далее рассматриваются технологии программирования в нашей стране.

3.1. Подходы к созданию технологий программирования производственных продуктов

Основу технологии производства программных продуктов (ПП), в том числе Product Line USA SEI составляют – линии продуктов. Подход к разработке технологических линий (ТЛ) предложен Е.М. Лаврищевой (1987) и апробирован в проекте Института Кибернетики АИС «Юпитер–470» для автоматизации военно-морского флота СССР (1982–1991). В рамках этого проекта создано шесть ТЛ. Каждая из них конкретизирована в виде визуальных структур моделей конкретных форм, документов, таблиц и схем понятий, их процессов и маршрутных схем создания ПП. По ним было реализовано около 500 программ обработки данных для разных объектов АИС. ТЛ разрабатывается по типу нано технологии с технологическим маршрутом на этапе технологической подготовки разработки (ТПР).

Маршрут задает процессы и операции, которые соответствуют выполнению задач автоматизируемой предметной области с помощью технологических модулей и комплекса информационного, методического, математического обеспечения операций процесса (рис.4).

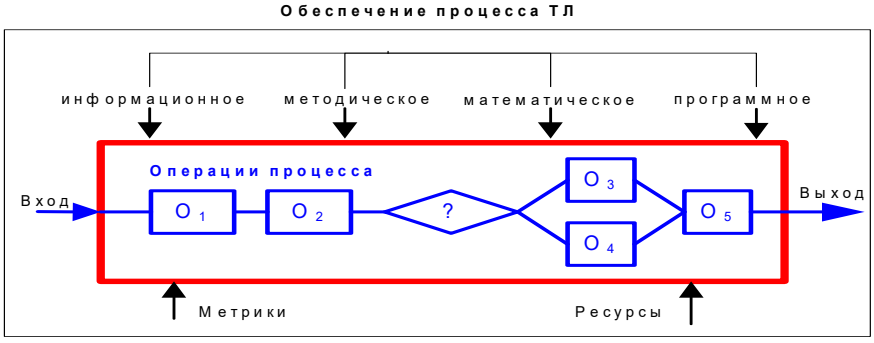


Рис.4. Маршрут технологической линии

При выполнении операций подбираются необходимые ресурсы (модули, модели качества) и средства оценки программ (метрики, показатели, характеристики) для реализации

конкретных задач и функций ПС. На вход маршрута подается состояние элемента процесса, а на выходе – результат, т.е. готовый продукт.

Все ресурсы упорядочены маршрутом ТЛ в структуру проектных решений по реализации и внесению изменений в отдельные элементы. продукта. Маршрут ТЛ описывается специальным языком с использованием инструментов, технологических модулей (ТМ) типа CASE и методиками управления последовательностью действий для выполнения процессов построения ПП, проверки эффективности работы отдельных элементов технологии и конечного продукта.

Работы в области метатехнологии велись за рубежом. К ним можно отнести языки UML, DSL, WorkFlow, новый стандарт языка нотации процессов – BPMN (Basic Process Modelling Notation) и др. Метод сборки программ по ТЛ (1987) послужил основой формирования стандарта ЖЦ ПС (ISO/IEC 12207– 1996, 2007 и ISO 9126–99) и специальных процессов для удовлетворения потребностей разных доменов. В 2000 годах появились продуктовые линии Product line и Product family (семейство СПП). Они определены в словаре ISO/IEC FDIS 24765:2009(E), Systems and Software Engineering Vocabulary как «группа продуктов или услуг, которые имеют общее управляемое множество свойств, которые удовлетворяют потребностям заданного сегмента рынка продуктов». Технология ПП и СПП определяет инженерию доменов и процессную модель продукта.

Модель инженерии – это модель разработки отдельных КПИ типа reuses, т.е. модель процесса создания готовых продуктов, управления разработкой КПИ и их сборкой в СПП. Процессная модель – это модель разработки «для обеспечения повторного использования» (for reuse) и разработки «с использованием готовых КПИ» (with reuse).

ТЛ и продуктовые линии семейств СПП и СПС используют готовые ресурсы, которые сокращают время и повышают уровень качества отдельных членов ПС и их семейств. Набор линий и отмеченных элементов (рис.5) образуют основу фабрик программ.

На данный час для подготовки разработки линий используется готовый инструмент TFLEX CAI, а также язык BPMN W3C и метод сборки семейств СПП из готовых КПИ и артефактов. Данный подход отражен на сайте ИСП РАН http://www.7dragons.ru/ru и в дальнейшем станет основой создания новых видов продуктов, веществ и приборов из микро элементов.

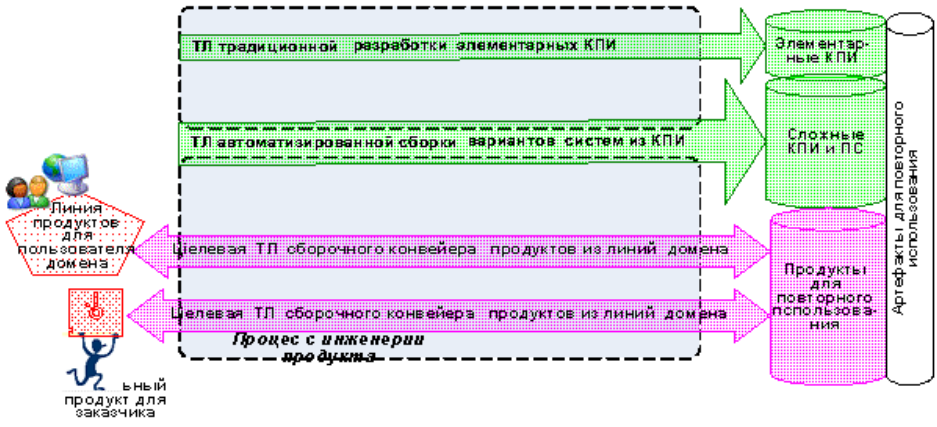


Рис.5. Набор линий на фабрике программ

Технологическая подготовка производства разных 3D изделий и приборов реализована в системе CAE (Computer-Aided Engineering) и используется в машиностроении и авиоционной промышленности.

Перспектива создание линий программ для вычислительных задач МФТИ

На кафедре «Информатики и вычислительной математики» ведется разработка портала для поддержки междисциплинарных наук факультета «Прикладная математика». В портале представлена линия биотехнологии для применения математических методов в медицине, биологии, физике и др., а также создается линия «вычислительная геометрия».

Вычислительная геометрия (Computational Geometry) - это часть компьютерной графики и алгебры. Используется в практике вычислений, управления станками с числовым программным управлением и др. Применяется также в робототехнике (планирование движения и задачи распознавания образов), в геоинформационных системах (геометрический поиск, планирование маршрута), дизайн микросхем и др. В портале будет реализовываться линия производства задач вычислительной геометрии. Кроме того, в МФТИ разрабатываются новые методы анализа состояния ледовитого океана и исследования недр земли и космоса. Магистрант МФТИ А. Островский в 2013г. разработан метод распознавания ДНК, который представлен на сайте http://7dragons.ru/bio.seg и задает:

– фрагменты генов – экзоны и интроны: с заданной последовательностью нуклеотидов гена;

– по заданной последовательности аминокислот белка выявляется их принадлежности к пространственным структурам – спиралям, листам и нерегулярным образованиям и др.

Данный метод найдет применение в генетике и медицине.

Ученые лаборатории квантовых систем МФТИ, МИСиС и МФТТ РАН создали первый в России сверхпроводящий кубит – элемент для квантовых компьютеров, который создает принципиально новые возможности для обработки информации. Кубит состоит из нескольких джозефсоновских контактов – двух сверхпроводников, разделенных тонким слоем диэлектрика. Электроны могут просачиваться через диэлектрик. Квантовые биты смогут выполнять вычисления, которые недоступны современным Персональным компьютерам.

Развитие индустрии ПП в мире характеризуется усовершенствованием КПИ и линий сборочного типа на фабриках программ. Каждая линия способствует повышению производительности и снижает себестоимость выпуска ПП. Каждый из названных способов идет по пути построения гибких и высоких технологий, приближающих к принципам нано технологий, которые автоматически синтезируют микроэлементные ресурсы в специальные ПП. В ряде научных публикаций отражены отечественные технологии производства ПС методом сборки объектов и компонентов в систему или семейство в Интернет-среде.

Основу подготовки студентов кафедры информатики и вычислительной математики МФТИ составляет курс “Программная инженерия”. Они обучаются теории и методам разработки программных и информационных систем, а также процессам стандарта ISO/IEC SWEBOK, ISO/IEC 12207 Life Cycle, ISO/IEC Quality 9126 и технологии для e-science (физика, математика, биология и др.).

3.2. Зарубежные ТЛ SPLE (Software Product Line Engineering)

Согласно стандарта ISO/IEC FDIS 24765:2009(E) «Systems and Software Engineering Vocabulary» была разработана **Product Line** – это "группа продуктов или услуг, которые имеют общее управляемое множество свойств, удовлетворяющие потребностям определенного сегменту рынка или вида деятельности". В технологии SPLE главную роль выполняет процесс управления вариабельностью отдельных артефактов (компонентов, reuses, assets и др.) систем и их СПП с учетом требований. Основные процессы создания программного продукта (ПП) SPLE – **domain engineering** and **application engineering**, которые отработаны практически при производстве коммерческих продуктов.

Процесс инженерии домена состоит в определении и реализации общих свойств (the commonality) обеспечения вариабельности и изготовления новых *вариантов* ПП. *Процесс*

инженерии приложений состоит в определении специфики отдельных приложений домена и изменяемости артефактов.

На Product Line выполняется процесс инженерии домена, который называется разработкой «для обеспечения повторного использования» (for reuse), и процесс инженерии приложений, как разработку ПС «с использованием КПИ» (with reuse). Модель Product Line отражает оба процесса разработки и сборки КПИ в СПП. Эти процессы повторяются при разработки новых КПИ и ПС с учетом требований.

В инженерию *доменов* и приложений для обеспечения изменяемости артефактов (requirements, architecture, components, test cases и др.) и архитектуры ПП добавляется модель вариабельности домена (Domain Variability Model) – набор функций, которые отмечаются вариантными точками и констрейнами для внесения по ним изменений. Модель вариабельности приложения (Application Variability Model) используется для получения разных вариантов членов семейства СПП. *Точка вариантности* – это место в артефакте ПП, определяющее выбор варианта артефакта и ПП.

Вариабельность – это способность семейства ПС или отдельного приложения и их артефактов к расширению, изменению, приспособлению или конфигурированию с целью использования в конкретном домене.

Процесс инженерии ПО домена в SPLE

Технология разработки продукта для домена включает процессы обработки требований, проектирования архитектуры, реализации домена и тестирования домена.

Инженерия требований домена определяет общие свойства функциональности, качества ПП и документ на требования и техническое задание на разработку домена.

Проектирование домена заключается в создании стандартной архитектуры по модели требований домена. Процесс ориентирован на механизмы конфигурации архитектуры для включения в нее спецификаций приложения и характеристик изменчивости. Вариабельность архитектуры определяется областью реализации и точками вариантов артефактов.

Реализация домена – это детальное проектирование проектных решений (assets) и artifacts, которые разрабатываются для этого домена. Обеспечивается configurable артефактов приложения путем задания вариантных контекстов для конфигурации.

Тестирование домена – это процесс проверки артефактов и КПИ по заданным требованиям с помощью тестов испытаний в вариантных точках артефактов продукта. Составляется стратегический план испытаний artifacts и домена в целом. Основу плана составляют наборы тестов, тестовых случаев и сценариев их выполнения.

Процесс изготовления приложений в SPLE

Технология разработки приложения включает аналогичные процессы: обработка требований приложения, проектирование приложения, реализации приложения, тестирование приложения домена. Эти процессы идентичны процессам домена. Они базируются на характеристиках приложений и домена. Для обеспечения **адаптивности и изменчивости** отдельных артефактов домена используется FM (Model Feature) характеристик артефактов, вариантных точек приложений и домена. Модель вариабельности приложения - основа процесса управления конфигурацией (configuration management) и создания продукта из готовых артефактов и reuses на процессе конфигурации (product configuration).

Для поддержки доменной инженерии разработана библиотека **DEMRAL.** Ее характеристики: основные понятия АТД и алгоритмы, которые оперируют в АТД; использование свойства контейнера, как в АТД; основная математическая теория реализации задач; разновидности алгоритмов берутся в АТД. Моделирование домена – это выявление

сущностей, их идентификация, определение диаграмм характеристик и алгоритмов задач в среде OS Linux.

3.3. Типовые AppFab (фабрики программ) на современных ОС платформах Интернет

В каждой операционной системе массового применения созданы фабрики программ.

В библиотеках системы содержатся наборы документов, заготовок, КПИ, наборы данных и др. Они используются для сборки и интеграции из них приложений и систем для разных целей, включая управление предприятиями, большими данными и др.

К фабрикам системного типа относятся:

1) Фабрика ПО (Fabrics Software) для систем и сервисов;

2) AppFab в системе коллективной разработки VS.Net;

3) AppFab в системе Grid Европейского проекта;

4) AppFab IBM для создания доменов предприятий и бизнес-приложений;

5) AppFab в системе CORBA для сборки разнородных программных ресурсов;

6) AppFab в системе INTEL и др.

Фабрики Программного обеспечения (ПО)

Эта фабрика ПО разработана группой **p&p.** Она включает целую коллекцию разного рода ресурсов, блоков кода, документации, образцов приложений и т.п. В состав фабрики вошли принятые схемы и стандарты, рекомендации, инструменты, шаблоны решений, упрощающие запуск новых приложений. Архитекторы модифицируют требуемые коды сервисных ресурсов модели SOA Visual Studio Industry Partners (VSIP), а затем повторно устанавливает их на фабрике ПО. Используется модели Guidance Automation eXtensions (GAX) и Guidance Automation Toolkit (GAT) в Visual Studio. GAX – это среда исполнения в VSIP с использованием пакетов рекомендаций. GAT – это набор инструментов реализации пакетов рекомендаций в GAX Visual Studio, которые имеются на любой другой фабрике, в том числе в IBM. Основу фабрики составляет **фабрики служб.** Существует два варианта: веб-служба ASP.NET (ASMX) для Windows Communication Foundation (WCF) в среде .NET Framework 3.0. Версии для веб-службы WCF находится в сообществе разработчиков фабрики ПО и GotDotN. В ее состав входят процессы предприятия, договора и пакет документаций и рекомендаций для приложения-образца Global Bank агентами.

В версию ASMX входят два пакета рекомендаций – один для решения задач ASMX и другой для задач доступа к данным. В версию WCF также входит два пакета рекомендаций – один для задач WCF и один для задач, связанных с безопасностью. Диспетчер пакетов рекомендаций обеспечивает запуск из меню инструментов Visual Studio 2007, а также включение и отключение пакетов рекомендаций. Это аналогично модели SCA в IBM WebSphere. Взаимодействие со службами выполняется в соответствии с договором на обслуживание, в котором определены операции сервиса и служб. Для каждой операции определяется тип сообщения и тип данных. Адаптер реализует договор обслуживания и его адаптацию к уровню предприятия, отражающего его объекты, логику деятельности объектов предприятия и его рабочие процессы.

Системные фабрики интеграции разнородных компонентов и данных

К ним относятся: IBM WebSphere, Microsoft Biz Talk 2004, BEA WebLogic Oracle 10g, SAP NetWeaver, ИВК "Юпитер» (см. табл. 1.).

Таблица 1. CASE–системы интеграции программных ресурсов

Платформа	Разработчик	Содержание платформы
IBM WebSphere	Корпорация IBM	Сервер приложений J2EE, брокеры обмена данными, КПИ, портал, workflow/BPM, средства EII, SCA
Microsoft Biz Talk 2004 и компоненты .Net	Корпорация "Майкрософт"	Сервер приложений COM, брокеры обмена данными, RGB доставки, портал, workflow/BPMN
BEA WebLogic	Корпорация "BEA Systems" (в 2008 г. присоединенная к корпорации "Oracle")	Сервер приложений J2EE, брокеры обмена данными, ГОР, сервер прикладных приложений, портал, workflow/BPMN
Oracle 10g	Корпорация "Oracle" http://www.oracle.com	Сервер приложений J2EE, брокеры обмена данными, ГОР, портал, workflow/BPM, средства EII
SAP NetWeaver	Корпорация SAP http://www1.sap.com/ www.sap.ru	Сервер приложений J2EE/ABAP, брокеры обмена данными, портал, инструменты BPMN
ИВК "Юпитер	Компания ИВК (Россия) http://www.ivk.ru/	Брокеры обмена данными, КПИ, среда выполнения, сертификация, защита данных

CASE корпорации IBM WebSphere. Эта платформа позволяет работать на основе веб-технологий. Ядро WebSphere включает в себя WebSphere Application Server (WAS) и открытые стандарты J2EE, XML и веб-сервисы. Это многофункциональный набор (коллекцию) инструментов *интеграции приложений* в рамках предприятия (EAI) на уровнях: данных, обмена сообщениями, бизнес-процессов, B2B business to business; сервисных компонентов; бизнес-логики программ на JAVA. В него входят:

1) сервер обработки очередей сообщений MOM (Message Oriented Middleware), операций интеграции Business Integration Interchange Server (ICS) и MQ Business Integration Message Broker (WSMB);

2) сервер приложений WAS;

3) Portal Server для поддержки функционирования в WAS;

4) система проектирования Workflow, совместимая с WSMB;

5) Сервисно-компонентная архитектура (SCA) для создания бизнес приложений.

В состав WebSphere входит Business Integration Workbench для проектирования бизнес-процессов и управления ими. Продуктами платформы являются образцы в классах, брокер сообщений WSMB, WAS и встроенные возможности для задания бизнес-правил и сценариев workflow, а также сборки компонентов Enterprise JAVA beans (EJB). Сервер WAS связывает ресурсы веб-сервисов в единый процесс и включает следующие средства:

– WebSphere Business Integration *for Automotive* для поддержки автоматизированного создания сервисно-ориентированных приложений деловых процессов;

– WebSphere Business Integration *for Financial Networks* обеспечивает консолидацию разнородных сетей обмена сообщениями в финансовой деятельности. Поддерживает интеграцию и оптимизацию операций проектирования, производства и выпуска новых продуктов и поддержку "унаследованных" систем и разнородных приложений;

– WebSphere Business Integration *for Energy* and Utilities обеспечивает оптимальную интеграцию процессов эксплуатации, управления активами и их обслуживания;

– WebSphere Business Integration Express *for Item Synchronization* обеспечивает связь информации из цепочек услуг для небольших компаний.

Модель SCA – это сервисно-компонентная модель для определения бизнес-сервисов, создания и настройки бизнес-решений через модель интерфейсов в языке WSDL, артефактов процессов в языке BPMN и реализацию продукта в Java ™.

Сервисный модуль WebSphere Process Server (эквивалентен EAR J2EE) объединяет модули SCA и SDO. Последний обеспечивает обмен данными между сервисными компонентами через объект данных. В нем содержатся ссылки к метаданным и информация о передаваемых данных, которые описываются в Java интрефейсе – commonj.sdo.DataObject и обрабатываются инструментами JMS, Enterprise JavaBeans или Web сервисы.

Таким образом, IBM WebSphere предоставляет набор средств по созданию приложений из компонентов, разного рода сервисов и сборки бизнес приложений.

CASE Microsoft.NET Framework. Эта платформа предоставляет разработчику функциональность в J2EE и в среде ОС Windows. Инструменты, необходимые для реализации разных интеграционных подходов, сгруппированы в виде нескольких продуктов, а отдельные функции возложены на ОС (например, управление транзакциями MTS, веб-сервер Internet Information Server, библиотеки и среда "управляемого кода" .Net).

Основная функциональность BizTalk Server 2004 – сервер интеграции на базе XML, как брокер сообщений, осуществляет преобразование данных и коммутацию сообщений. Сервер приложений выполняет бизнес-логику – низкого (компоненты EJB) и высокого (через механизмы workflow) уровней. Этот сервер поддерживает высокоуровневую бизнес-логику и интеграцию систем, выполнение логики низкого уровня реализуется моделью COM+ или .Net. Модель Microsoft позволяет размежевать работу программиста и аналитика бизнес-процессов. Бизнес-аналитик может графически "рисовать" бизнес-процесс, задавая схемы обмена документами и передачи управления через workflow. Для интеграции определяются точки вызова внешней функции через COM-объекты, Веб-сервисы и т. п**.**

CASE WebLogic Integration. Это инструмент создает бизнес-логики на языке JAVA и интегрирует приложения с обеспечением взаимодействия с бизнес-партнерами (B2B). Платформа включает в себя пять основных компонентов: виртуальная машина JAVA, сервер приложений, средство построения порталов, пакет инструментов интеграции, среда разработки. Ключевое преимущество платформы – возможность снижения требований к группе разработки за счет использования трехуровневого подхода к созданию приложений, подобно подходу корпорации Microsoft. Программы создаются на языках JAVA, Visual Basic или COBOL. Платформа BEA основывается на новейших стандартах XML (XSLT, XQuery и т. п.), веб-сервисах и средствах доставки, совместимых со стандартом JMS и брокером сообщений. WebLogic имеет набор интерфейсов для интеграции приложений, файлов и баз данных разной природы. В нее входят набор готовых конвекторов, системы документооборота, анализаторы форматов файлов и средства обращения ко всем модулям программ Windows и JAVA для взаимодействия с другими интеграционными платформами.

CASE Oracle Integration. Данная платформа предоставляет полный набор средств корпорации "Oracle" для интеграции приложений из разнородных приложений. Платформа соединяет технологии нескольких классов со стилями интеграции: данных (технология Transparent Gateways и конвекторы базы данных), интерфейса пользователя, сервера и системы MOM. В Oracle Application реализованы новейшие возможности СОА и веб-сервисов управления их координацией и композицией приложений для любых сред разработки приложений. В ней выполняется конструктивное развитие двух современных парадигм конструирования приложений с корпоративными вычислениями и использованием сервисов (Service-Oriented Computing) и сетевых вычислений (Grid Computing). Предлагается аспектная инфраструктура реализации SOA и объединения (federate) приложений с доступом к ним и к сервисам. Сетевые вычисления предусматривают использование собираемых из модулей серверов и блоков памяти, которые имеют низкую стоимость при эксплуатации приложений поддержки деловых процессов вида;

1) Business Intelligence для анализа бизнес-информации предприятия, построения ПС путем сборки, анализа и распределения разноуровневой информации;

2) Business Integration для интеграции разнородных приложений, включая объединение отдельных подсистем и автоматизацию деловых процессов;

3) Identity Management - управление идентификацией личности, что позволяет администрировать средствами защиты и снижать стоимость владения точками уязвимости.

Для конструирования приложений используется пакет – Oracle Integration Interconnect. В нем имеются средства композиции и координации сервисов (служб) предприятия и ESB-развертывание интеграционных приложений на предприятии. Для интеграции связей используются Repository метаданных SAP и Directory. Особенность платформы SAP – это интеграция не только на уровне конектора к шине обмена данными, но и на высших уровнях, совместимых с системой управления контентом, и порталом среды, включающей Eclipse, IBM WebSphere, SAP Web Application Server через SAP JAVA Connector и SAP .NET Connector.

Платформа ИВК "Юпитер". Этот продукт обеспечивает функции интеграции на уровне данных и обмена сообщениями для потребностей государственных структур, которые выдвигают повышенные требования к защите информации и интеграции унаследованных и устаревших ПС. Эта платформа соединяет характеристики виртуальной машины, транспортной магистрали, отдельные свойства систем документооборота и средства защиты информации. Она состоит из двух высокоуровневых логических блоков:

1) обеспечения стандартной функциональности на отдельном компьютере;

2) определения связи между разными компьютерами.

На каждом компьютере представлена унифицированная модель вычислительного процесса, которая соединяет среду выполнения ИВК "Юпитер" и набор библиотек для создания приложений, которые могут выполняться в средах типа Cloud Computing через модуль API ИВК "Юпитер". Продукт обеспечивает контроль целостности вычислительного процесса в начале загрузки и при выполнении. Имеется встроенная возможность эмуляции IP поверх многих унаследованных транспортных протоколов с гарантированной доставкой сообщений в гетерогенной сети и реализованы возможности документооборота.

Таким образом, из приведенного описания современных платформ индустрии приложений следует, что разработаны типичные фабрики в среде общих систем с базовыми средствами создания, компоновки и выбора специальных КПИ и сервисов, а также сборки КПИ и сервисов в новые приложения для функционировать в любой среде, в том числе в Grid и Cloud Computing.

Современные фабрики программ в СССР

Фабрика программ – это интегрированная инфраструктура сборочного производства из готовых КПИ новых ПП (систем, семейств систем, ИС, АСУ, АСУТП и др.).

Фабрика включает в себя комплекс системных средств, инструментов и сервисов, а также репозитории и библиотеки программ для накопления и выбора готовых КПИ. Ядро фабрики – операционная среда и методы изготовления (UML, компонентный, структурный, модульный, сервисный и др.) отдельных КПИ. В среду фабрики вводится набор веб-сервисов и веб-семантики для управления выбором необходимых сервисов из Интернета и их использования, как КПИ, при сборке и композировании новых ПП.

Коллекция фабрик программ приведена ниже:

1) система АПРОП (ИК);

2) система Sun Microsystems (IBM)*;*

3) OMA–архитектура или система CORBA (OMG)*;*

4) фабрика "ручной" сборки разноязычных программ Инга Бейя;

5) фабрики программ по методу UML Дж. Гринфильда;

6) среда для групповой разработки ПП – MS.VSTS;

7) инфраструктура вычислений Grid;

8) фабрика Г. Ленца на основе Use Case;

9) каркас фабрики программ Авдошина;

10) Фабрика «continiuos Integration» Фаулера;

11) Фабрика программ ЕПАМ и др.

Системы АПРОП, IBM, CORBA проложили первый путь к созданию современных фабрик программ. Рассмотрим их базовые положения.

АПРОП – это первая фабрика, которая работала в среде ОС ЕС (1976) и обеспечивала сборку разноязычных модулей в монолитную структуру на ЕС ЭВМ через интерфейсных посредников, сгенерированных с помощью специальной библиотеки функций преобразования нерелевантных FTD типов данных (например, символьного к целому и т. п.), которые передаются операторами CALL в ЯП (4GPL) модулях и реализуют методы численного анализа и статистики, которые расположены в специальном банке модулей и .библиотеки из 64 функций преобразования отличающихся типов данных .

IBM – среда сборки разноязычных программ в 4GPL (1980–е годы) с помощью внешних интерфейсных данных, которые трансформируются функциями XDR-библиотеки, Sun Workshop, Toolbox и т. п. к соответствующей платформе Дальнейшим развитием новых направлений производства ПП является модель архитектуры SOA, веb-сервисы, языки C, C++, JAVA, RUBY, SCRIPT, которые обеспечивают связь программ в ЯП и передачу данных через порт системы AIX. Интеграция разнородных программ выполняется на общей платформе IBM в средах – ONC (Sun Microsystems), MVS, VM, OS/2, AIX, Open source, на сервере (WebSphere Application Server Compunity Edition).

CORBA или OMA-архитектура (Apple, IBM, Win-NT, x-Open, Dec) обеспечивает связи разноязычных объектов в ЯП (C, C++, Smalltalk, JAVA, Cobol, ADA-96 и др.) через интерфейсные посредники (stub, skeleton, dill), которые описываются в языке IDL для клиент-серверной архитектуры (Client-interface, Server-Interface) с использованием сред (COSS, DCE/RPC, PCTE, ToolTalk, JAVA2SDK, NetPilot CCS и т. п.). Данные между клиентом и сервером передаются через модули посредники (stub, skeleton) в языке IDL или протоколами TCP/IP, IIOP через брокер ORB, который обеспечивает их разным сервисом, в том числе по преобразованию несовместимых типов данных разноязычных объектов, устранения неоднородности платформенных данных взаимодействующих объектов клиента и сервера. Эта среда поддерживает связи с другими средами CORBA, OLE/DCOM, SOM/DSOM, IBM OS и т. п.

Фабрики программ на платформе Microsoft. Среда MS.NET содержит большое количество средств и инструментов: готовые ресурсы (компоненты, сервисы) Интернета, языки – JAVA, C++, Basic, JAVA, Pascal, C#, библиотеки CLR и FCL, сборка кодов (exe, dill) в ПП, веб-обслуживание, управление проектом PM–2007 в виртуальной среде VSTS и MSF. Среда ориентирована на коллективную разработку систем, в нее входят: пакет инструментов VSTS–2007 (Visual Studio Teams Systems); MSF (Microsoft Solution Arhitecture) для построения производственной архитектуры предприятия; модели процессов и систем; Professional Studio и Foundation Server для поддержки процессов проектирования, кодирования, тестирования и формирования версий ПП (SDLC, IDE, MS Office, MS SQL server, MS Visual Studio 2007); модель усовершенствования процессов (CMMI Process Impovement), процесс сборки с использованием CLR-библиотеки (Common Language Routine), FCL-типы, трансляторы, General code (exe), Portable Executable code и т. п. В среду обработки находятся средства определения сроков работ, трудозатрат, оценки показателей качества готовых ПП и др.

Visual Studio Teams Systems – это семейство продуктов для коллективной разработки ПС, их взаимодействия, а также для отбора, распределения и выполнения работ в

программном проекте. Команда исполнителей проекта разделены на четыре категории с учетом их уровня знаний и навыков программирования систем (рис.1).

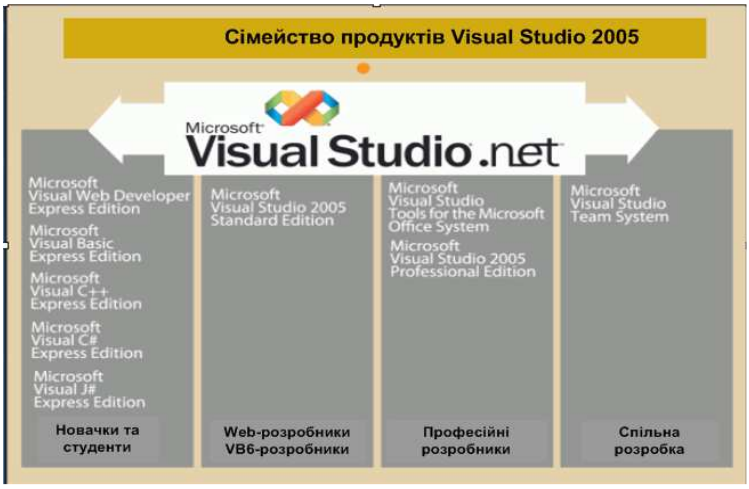


Рис.1. Категории разработчиков проектов ПП

Специалисты четвертой группы - это квалифицированные и ответственные за функциональную правильность разработки ПС. К средствам поддержки процесса разработки элементов ПС относятся IDE (Integrated Development Environment).

В нем проводится проверка требований, отслеживание процессов и результатов разработки с оцениванием степени их готовности и выполнения в Microsoft Office Project и Microsoft Office Excel. Тестирование элементов ПС осуществляется специальными средствами, которые интегрируются с Visual Studio.

В среде VSTS имеются такие средства:

– *Microsoft SQL Server 2007* используется для сохранения всех рабочих результатов, исходного кода и данных интеграции Team команды. Поддержка разных типов отчетов членов команды и работы портала обеспечивается инструментами *Analysis и Reporting Services* и *SharePoint Portal Services*;

– *Microsoft Project 2007* – альтернативное средство для руководителей проекта;

– *Microsoft Excel 2007* – средство для управления проектированием ПС и т.п.

В среде VSTS осуществляется разработки продукта на основе *требований,* заданий *(*task) на выполнение, определения рисков, поиска ошибок в системе и путях их исправления.

Таким образом, среда VSTS предназначена для коллективной разработки систем и разных ПП посредством инструментов управления процессами ЖЦ на проекте и графиком работ, а также для отслеживания и оценивания результатов на качество, стоимость и сроки выполнения проекта.

Средства и инструменты методологии MSF.

Эта методология – система стратегий, принципов и управления проектом построения производственной архитектуры предприятия с учетом: объемов работ в проекте, времени и стоимости, количества персонала, коммуникаций, закупок и контрактов, рисков. В ней разработана модель архитектуры предприятия с такими аспектами: *области приложения*, *бизнес, информация, технология*. Для разработки ПП создается скоординированный технологический план, который отвечает приоритету архитектуры и получению максимального эффекта при минимуме расходов.

Метод MSF базируется на анализе и требованиях к ПС, проектировании проектных решений, которые учитывают базовые концепции предприятия и приоритетность архитектуры. Этот метод содержит в себе набор моделей: производственная архитектура;

проектная группа; процесс разработки ПС; управление рисками; процесс проектирования применения. *Модель производственной архитектуры* – это набор принципов для создания версии производственной архитектуры предприятия согласно четырех перспектив. Основная задача этой модели – приспособление производственной архитектуры к бизнес-целям организации для поэтапного выпуска последовательных версий, отмеченных приоритетами для получения целевой производственной архитектуры. Члены проектной группы на процессе разработки создают: код системы, конфигурацию, функциональную спецификацию и сценарии тестирования. *Модель процесса* проектирования определяет цель и задачи производственной архитектуры с помощью концептуальной, логической и физической фаз, т.е. выполняется систематический переход от абстрактных концепций к конкретным техническим решениям. *Модель применения* – это трехуровневая структура, созданная сценарным методом разработки системы.

Таким образом, методология MSF ориентирована на проектирование программного и информационного обеспечения предприятия с помощью рассмотренных принципов, моделей и методов решения задач предприятия.

Фабрика сборки И. Бея. Базис этой фабрики – разноязычные программы, интерфейсные посредники, конфигурационные файлы. Программы описываются в ЯП (VC++, VBasic, Matlab, JAVA, Visual Works Smalltalk и т. п.) и выполнялись на платформах (MS.Net, HP, Apple, IBM и др.). В интерфейсном посреднике данные передаются данные друг другу. Для некоторых данных, типы которых нерелевантные или и их форматы зависят от другой платформы или от механизмов передачи данных (протоколы, вызовы, интерфейсные карты MIO–16E–2 и др.) проводилось ручное их преобразование. Описание интерфейсных связей проводилось с помощью RMI, RPC, Java Native Interface. Затем они реализовывались в виде exe-файлов. Принцип взаимодействия разноязычных программ апробирован на Domain, Application Models, Model Interconnection, Microsoft Communication Foundation. Для внесения изменений в типы данных использовались современные визуальные средства (панели, сценарии, иконки и т. п.).

Фабрика программ Дж. Гринфильда. Для любой строящейся фабрики сформулированы методологические и технологические аспекты производства ПП методом UML с использованием моделей архитектур систем и компьютеров, механизмов интеграции разнородных компонентов с типами данных FDT и описанием интерфейса в языках (IDL, XML, RDF и т. п.). Главные объекты производства: *reuse*, которые накоплены в общепринятых хранилищах (библиотеках, репозиториях и т. п.) Интернет и имеют сертификаты качества; *активы* (assets), программы, приложения и системы; *модели,* шаблоны и инструменты UML для реализации ПП на линиях производства; *веб-сервисы,* процессы линий; методики измерения и контроля качества ПП и т.п. Анализ системы моделирования UML для применения на фабрике показал, что UML удобен для задания эскиза ПП, в который используются компоненты и интерфейсы. Проблема модификации ПП не решена в визуальном языке UML и др. Фактически автором разработан меморандум современной фабрики ПП, которая базируется на reuses, линиях, моделях систем, каркасах процессов и продуктов.

Фабрики для вычислений в Grid. Проект Grid ориентирован на организацию распределенных вычислений задач в разных научных направлениях (физика, математика, медицина, биология и др.). Он включает ряд подпроектов: Gcube-операционная среда, ETICS – как сборочная среда и т. п. Она предназначена для производства распределенных систем разного назначения методом интеграции (сборки) существующих готовых КПИ и ПП с применением веб-порталов и многоплатформенных ресурсов. Технология создания пакетов из исходных или перекомпилированных программ в двоичном представлении обеспечивается репозиторием при выборе компонентов системы в альтернативной сетевой среде Grid. Паспорта КПИ представляются в стандарте WSDL (таблица. 1) и сохраняются в репозитории системы.

Таблица 1. Формат стандарта описания паспорта КПИ

Дата создания	Дата создания КПИ автором (дата аттестации специфицированной версии продукта)
Дата изменения	Дата внесения изменений в КПИ
Версия	Версия КПИ
Платформа	Платформа, для которой создавался КПИ и на которой проверена его работоспособность
Операционная система	Операционная система, для которой создавался КПИ и на которой проверена его работоспособность
Размер	Общий размер КПИ (продукта, документации, и т.п.)
Описание	Короткое описание КПИ, список внесенных изменений, системные требования, требования к пользователям, список необходимых программ для корректной работы, справка, и т.п.
Правила использования	Описание операций выполнения КПИ, правил эксплуатации и т.п.

Виды средств Grid – проект, подсистема и компонент реализуется с использованием формата CIM для связи между разными компонентами с помощью системы MySQL и паспортных аннотаций КПИ, которые включают характеристики (имя, лицензия, глобальный идентификатор – ID (GUID), URL и т.д.).

Компоненты компилируются и собираются в конфигурационном файле ПП. Сервисы – главные ресурсы инфраструктуры вычислений. Они обеспечивают процесс вычислений научных задач глобального масштаба. Ресурсы задаются в протоколе для передачи данных по распределенной сети. Сервисы предоставляются стандартным протоколом, интерфейсом API и инструментом SDK (Software Development Kits). Сборка программ, компонентов, подсистем и систем научного назначения реализуется протоколом (Global Protocol). Подсистема ETICS содержит средства разработки, тестирования пакетов и услуг, а также сборку и конфигурирование программных элементов на основе механизмов плагинов и открытых интерфейсов для обслуживания потребителей или поставщиков.

Главные особенности рассмотренных фабрик программ – это методы производства инструментальной поддержки среды для автоматизации процессов ЖЦ или ТЛ.

Фабрика ПО Ленца. Ключевые элементы этой Software Factory – схемы и шаблоны. Схема Software Factory – это описание того, как реализовать продукты, которые могут быть произведены на этой фабрики. Технология базируется на MDD и поддерживается архитектурным фреймворком. Типичный пример шаблона – завод ПО, а именно, инструмент Visual Studio, обеспечивающий разработку проектов из многократно используемых компонентов при реализации проектных решений и требований. Эти требования и решения описываются на языке DSL в виде задания архитектуры, блоков и документов, которыми заказчики приложения будут пользоваться при реализации соответствующей линии продукта или ее экземпляра.

Архитектурный каркас фабрики Авдошина. Основной ресурс и активы данной фабрики – архитектурный каркас, являющийся отправной точкой, в разработке любого продукта на линии. В нем аккумулируются активы и ресурсы: классы, компоненты, конфигурации, образцы, проектные решения и т. п. Активы выбираются на стадии разработки линии фабрики. База знаний фабрики, включает в себя разные пособия, справочники, статьи, примеры программ и программы-образцы. Моделирование понятий и основных концепций в языке DSL способствует генерации кода и конфигурации структуры продукта. Для реализации фабрики задается схема фабрики, которая охватывает все активы, точки зрения и

связи между ними. Из схемы берется шаблон фабрики и уточняется набор необходимых ресурсов для автоматизации работы среды разработки. Любая схема фабрики состоит из набора взаимозависимых точек зрения, каждая из которых дает возможность посмотреть на систему с разных сторон. Точки зрения отражают логическую и физическую стороны системы, набор используемых компонентов и документирование архитектуры семейства ПП. Каждая точка зрения имеет имя и описание, она указывает разработчику, что строить, как строить и из чего (моделей, средств, шаблонов и т. п.).

Шаблон фабрики – это пакет с ресурсами фабрики и экземплярами схемы категорий:

1) библиотеки и каркасы с КПИ, разработанными на линии (.NET Enterprise Library);

2) рекомендации, технологии, распоряжения и руководства процесса построения ПП;

3) языки предметной области и дизайнеры, которые задают необходимый уровень абстракции при разработке приложения и генерации кода по модели.

Построенный шаблон фабрики загружается в интегрированную среду разработки для автоматизации сборки ПП и получения продукта.

Фабрика программ "Continiuos integration" (непрерывная интеграция, повторная сборка) М.Фаулера (M.Fowler, K.Beck). Это автоматизированный процесс, обепечивающий разработку, поиск изменений в коде и в системе, сборку, контроль v собранных версий, развертывание и тестирование приложения. На фабрике используется гибкие (Agile) методологии, включая такие практики, как модульное (unit) тестирование, рефакторинг, стандарт кодирования. Каждое изменение в системе контроля версий (например, CVS) должно интегрироваться и проводиться на специальной машине, в среде которой проект будет развернут (production enviroment) для уменьшения риска, связанного с конфигурацией программного и аппаратного обеспечения. Процесс интеграции выполняется на сервере интеграции (continuous integration server) – CruiseControl, который написан на Java и CruiseControl.NET. К основным процессам сервера интеграции относятся: **Build** – это компиляция (трансляция) исходных кодов в исполнимые файлы и их сборка средствами Ant, Maven в Java и NAnt в.NET. **UnitTest** предназначен для тестирования модулей приложения с помощью автоматизированных тестов. Проект **Deploy** развертывается после проведения Build и всех модульных тестов UnitTest. Развернутое приложение тестируется процессом Test с помощью автоматических функциональных тестов. После прохождения регрессионных тестов считается, что интеграция прошла успешно. В противном случае требуется вносить исправления и процесс интеграции повторяется.

Фабрика EPAM предоставляет широкий спектр технологий и решений, процессов, сертифицированных по стандартам качества CMMI Level 4 и ISO 9001, высокоэффективные инструменты управления проектами (http://www.epam-group.ru/# sthash.zPWSiln4.dpuf) аутсорсинга для разработки, внедрения, интеграции, тестирования и сопровождения программного обеспечения. Практически создано более 15 филиалов в СНГ.

3.4. Парадигмы технология сборки систем из готовых ресурсов – ГОР, КПИ, Reuses

Разработка разнородных артефактов, reuses и КПИ осуществляется в рамках парадигм программирования (модульной, объектной, компонентной, сервисной, генерирующей). Цель каждой парадигмы – создать соответствующий базовый элемент и интерфейсный объект для связи разнородных базовых элементов парадигм в структуру ПС. Разработан формальный аппарат каждой парадигмы программирования. Он включает теоретические и прикладные аспекты проектирования соответствующих элементов и операции их интеграции в сложные ПС. Сборка разноязычных элементов основана на теории преобразования фундаментальных типов данных (FDT), возникших еще в 70–х годах прошлого столетия в работах Дейкстра, Хоара, Вирта, Агафонова, Ершова и др. и стандарте общих типов данных ISO/IEC GDT 11404 –2006 (General Data Types) для генерации типов данных GDT ↔FDT.

Данная теория первоначально реализована в системе АПРОП (1980), АИС «Юпитер–470» (1982–1991) и на экспериментальной фабрике программ Киевского национального университета (КНУ) 2011 (http://programsfactory.univ.kiev.ua). В состав фабрики программ включен набор CASE–средств, который поддерживает аппарат формального построения отдельных объектов и компонентов данных парадигм. К ним относятся средства поддержки спецификации элементов в ЯП, описания интерфейсов в языке IDL и сохранения их в библиотеке готовых элементов. Сборка элементов парадигм в новые разные виды ПС осуществляется с помощью специальных операций сайта ИТК (http://sestudy.edu-ua.net). В их разработке принимали участие аспиранты и студенты КНУ и филиала МФТИ при институте Кибернетики имени академика Глушкова.

Стили парадигм сборочного типа. К настоящему времени сформировались:

1) стили программирования по принципу модульности и стандартом модулей, которые собираются в более сложную структуру;

2) метод повышения эффективности межмодульных интерфейсов при передачи данных на компьютеры с разными форматами данных;

3) базы КПИ (Библиотеки, Репозитории), их идентификация, выбор и проверка пригодности применения исходя из стандартизованного описания элементов и их интерфейсов.

Базовые элементы и КПИ являются программными «кирпичиками", которые соединяются интерфейсами, как «болтами и гайками» технические изделия. Программные элементы могут быть в исходном и двоичном видах. Методология сборки позволяет подключать новые элементы, ресурсы, которые определяются в модульном, объектном, компонентном и сервисном программировании. Эти ресурсы должны разрабатываться стандартизировано в WSDL любых операционных средах IBM, MS, Microsystems, CORBA, COM, Oberon и др.

Модульное сборочное программирование. Этот подход был исторически первым и базировался на процедурах и функциях, разноязыковых модулях и методологии императивного программирования в среде ЕС ЭВМ прототипа IBM–360.

Объектное сборочное программирование. Подход базируется на методологии объектно–ориентированного программирования и предполагает использование библиотек методов и классов (исходные коды или упаковки классов) в динамическую библиотеку. Существуют конкретные подходы, поддерживающие это программирование, например, CORBA, Rational Rose, ОКМ и др.

Компонентное сборочное программирование. Основные идеи подхода – распространение классов в двоичном виде и предоставлении доступа к методам класса через строго определенные интерфейсы, которые позволяют снять проблему несовместимости компиляторов и обеспечивать смену версий классов без перекомпиляции. Существуют конкретные технологические подходы, поддерживающие компонентное сборочное программирование: COM (DCOM, COM+, .NET, OBERON и др.).

Аспектное сборочное программирование. Оно дополняет компонентное программирование концепцией аспекта для изменения варианта реализации критичных по эффективности процедур и программ. Это программирование заключается в сборке приложений из аспектных компонентов, инкапсулирующих различные варианты реализации (безопасность, синхронизация, надежность и др.). Существуют конкретные технологические подходы, поддерживающие данное программирование

Сервисное сборочное программирование. Это новая концепция интеграции сервисов и обслуживания ПС. К сервисам относятся: общие системные сервисы для поддержки реализации и выполнения ПС (связь, управление, каталогизация и др.); объектные сервисы, которые поддерживают объекты и классы, операции их выполнения и др.; веб–сервисы Интернет для быстрого решения поставленных задач, а также сервисно-

компонентные службы SCA и SDO для создания приложений предприятий. Существуют конкретные системы, поддерживающие данное программирование.

Сборочный конвейер для поддержки стилей сборки

Сборка первоначально была представлена как способ объединения разноязычных объектов в ЯП и преобразования типов данных (ТД) с помощью теории спецификации и отображения типов и структур данных ЯП средствами алгебраических систем с операциями и функциями релевантного преобразования одних ТД в другие.

Сборка ресурсов включает:

1) метод программирования, который подчиняется общим закономерностям и функциям, используется для поддержки КПИ программных элементов, объектов, КПИ и интерфейсов;

2) механизм оценки стандартизованных КПИ и их интерфейсов и сборки продукта. Этим он отличается от процессов синтеза, композиции и от других методах программирования.

Базовые элементы сборки обладают свойствами (наследования, полиморфизма и инкапсуляции). Они включают описание данных и операций метода выполнения базового элементов для обеспечения связи разноязычных КПИ. Для технологичности сборки все базовые элементы сборки должны иметь паспорта в стандарте WSDL, в которых специфицированы данные, необходимые для информационной связи в более сложные структуры и выполнения в операционной среде.

Важное условие сборки – наличие большого количества разнообразных комплектующих КПИ, которые обеспечивают решение широкого спектра задач из разных предметных областей. Для их сборки задается схема сборки и операторы вызова (CALL, RPC, RMI и т. п.) в модуле, связанным отношением связи с другим модулем. В вызове задается список параметров и значений, которые при их передачи другому модулю проверяются на соответствие ТД исходя из аксиом и утверждений системы преобразования одних ТД к другим в классе ЯП. Результат отображения – сгенерированные интерфейсные модули–посредники для эквивалентных преобразований ТД в процессе выполнения.

Процесс сборки любых КПИ как готовых ПП – это линия сборочного конвейера, в котором роль "деталей" выполняют КПИ разной степени сложности, а роль "стыковщика" – интерфейсы–посредники. Последние присутствуют во многих методах и стилях программирования. На фабрики программ разработчики работают с КПИ как с деталями и выбирают те из них, которые могут быть комплектующими, т. е. повторно используемыми. Интерфейс каждой пара объектов сборки зависит от использования данных, их ТД и передаваемых значений, а также от наличия библиотек классов и функций преобразования ТД. В системе MS.Net имеются специальные библиотеки – CLS (Common Language Specification), стандартных процедур CLR (Common Language Runtime), типов CTS (Common Type System), которые реализуют типы данных в языке спецификации CLS в системе типов .NET.

Интерфейс сборки. Типы интерфейса

Для сборки разноязыковых модулей был использован термин интерфейс (1976), соответствующий стыковочному элементу (болту, гайке) в промышленной сборке стандартных деталей. Интерфейс – это связь двух отдельных сущностей. В компьютерной области интерфейс определяется на разных уровнях: от уровня видимых коммуникаций между людьми до аппаратных, программных, пользовательских, языковых и других интерфейсов. Аппаратный интерфейс – это разъемы, конекторы и другие устройства для объединения компонентов в компьютерную систему и обеспечения перемещения информации с одного компьютера в другой. На *программном уровне* интерфейс между программами и ОС, между ОС и аппаратурой обеспечивает передачу и преобразование входных/выходных данных взаимодействующих программ в ОС или во время объединения

компьютера с периферийным оборудованием. В ЯП интерфейс – типы данных и описание констант, переменных, параметров и сложных структур данных, которые образуют межъязыковой интерфейс ЯП и способ эквивалентных преобразований.

В программировании, интерфейс содержит паспортные данные программного модуля, включающие сведения о передаваемых данных и их типах. Каждый модуль специфицировался в одном из ЯП ОС ЕС. Стыковочный элемент (интерфейс) описывался в специальном языке MIL прототипе IDL (Interface Definition Language). Для каждой пары разноязычных модулей формировался интерфейс в виде модуля связника. По оператору Link A, B, C осуществлялась сборка модулей A, B, C и их интерфейсов в продукт на ЕС ЭВМ в рамках специально разработанной системы АПРОП. Полученный продукт отличался от продукта традиционного программирования наличием самостоятельных отдельно разработанных модулей и их интерфейсов. Тем самым имеется возможность изменять модули и интерфейсы, которые осуществляют обмен данными и преобразование нерелевантных ТД с помощью (64) функций библиотеки интерфейса.

Идея интерфейса получила развитие к 1987–1984 годам, когда были созданы специальные языки описания программных интерфейсов IDL, API, SIDL и др.

Описание интерфейсов в IDL OMG CORBA

Язык описания интерфейсов IDL разработан в OMG CORBA. В этом языке описываются интерфейсы как посредники (stub, skeleton). В них задается описание интерфейса, начиная с ключевого слова **interface**, имени интерфейса, типов параметров и операций вызова объектов. Параметрами обмена могут быть: **in** — входной параметр, **out** — выходной параметр, **inout** — совместный параметр. Пример описания параметров интерфейсов в виде **stub F_1** и **skeleton F_2** приведен ниже.

interface F_1 {
void f (in float s [l]); }
interface F_2 {
const long l=3 }
interface P_3: F_1 ( ) A, F_2( ) B.

Описание параметров интерфейса модуля в IDL CORBA имеет вид:

Request Operations
module CORBA {
interface Request {
Status add-arg (
in Identifier name
in Flags arg flags
);
Status invoke (
in Flags invoke flags // invocation flags
);
Status send();
Status get response (
out Flags response flags // response flags); };

В ТП сформировались следующие **виды интерфейсов**:

– модульный, программный (RPC, RMI, IDL, API, ISO и т.п.);

– межъязыковый интерфейс (Java Native Interface, SIDL– Scientifically IDL, Fundamental Data Types, GDT – General Data Types);

– сервисный интерфейс (IContract, Web, ISO);

– промежуточный интерфейс (Middleware, Virtualware, ISO, межсистемный, и между клиентом и сервером и т.п.);

– технический интерфейс (стандарт интерфейсной карты МПО–16Е–2).

Новый вид интерфейса в WCF *Icontract* содержит описание атрибутов и операций передачи данных от одного сервисного объекта клиента (*Service consumer*) к другому (*Service provider*). Их описание задается в языке XML. Передача интерфейсов между ними выполняет протокол, в котором задаются атрибуты и операции интерфейса.

В системе WCF реализовано четыре вида интерфейсов-контракта:

1) службы операций, вызываемых клиентом;

2) фундаментальне типы данных (int, float, string и др.) для передачи их другим модулям;

3) ошибки, которые могут содержаться при передачи контрактов клиенту;

4) сообщения для взаимодействия объектов между собой.

Каждый интерфейс IContract задает атрибуты и операции обмена данными между клиентом и сервером.

Модели взаимодействия систем и продуктов в современных средах

В связи с постоянным изменением архитектур компьютеров, появлением распределенных, клиент-серверных и гетерогенных сред выявилась неоднородность ЯП, в плане представления типов данных, подходов к их реализации в системах программирования, а также форматов представления и передачи данных. Стандарт ISO/IEC 11404–1996 определил подход к решению вопросов интерфейса ЯП с помощью независимого языка LI (Language Independent) от ЯП. До настоящего времени отсутствует реализация LI и пользователи разных ЯП выбирают реализацию интерфейса в разных средах.

Под *взаимодействием* понимают совместимость двух и больше объектов. Данный термин имеет специальный спектр применения в программистской деятельности (например, взаимодействие программ и сред между собой). Способность к взаимодействию двух и больше программ в разных средах зависит от способа обмена информацией между ними. К способам обмена данными относятся RPC, RMI, ORB (stub, skeleton), IContract и др. Эти операции связывают разнородные программы через интерфейс в языке IDL, APL, SIDL и т. п. Именно интерфейс стал механизмом обеспечения взаимодействия (interconnection) разнородных программ и систем.

Один из способов взаимодействия систем представлен в стандартной модели OSI (Open System Interconnection). Модель определяет виды взаимодействия систем на семи уровнях: прикладном, представления данных, сеансовом, транспортном, сетевом, канальном и физическом. Системные средства взаимодействия реализованы операционными системами в разных средах (DCE, OSF, ONS, OLE/COM, OMG CORBA, WCF, Eclipse и др.). В них связь между компонентами системы происходит через запрос к прикладному уровню модели OSI. Механизмы взаимодействия обеспечивают связь компонентов и систем, которые реализованы в соответствующих операционных средах.

Моделью взаимодействия – это набор параметров для обмена разнородной информацией между разными системами. Модель отображает систему отношений, которая существует между программами и системами в виде интерфейса и сообщений.

Каждая среда базируется на своих интерфейсах взаимодействия и включает в себя общие методы и средства доступа к данным. Программы, изготовленные в одной из сред, могут быть перенесены из одной среды в другую через интеграцию в репозиторий Eclipse. Эти среды обеспечивают реализацию процессов ЖЦ и объединение результатов ЖЦ в разные структуры ПП с помощью рассмотренных способов взаимодействия программ.

В рамках курса по ТП студентами проведена экспериментальная реализация принципов взаимодействия программ для следующих пар системных сред:

1) Visual Studio.Net↔Eclipse – это виртуальная среда для технологии разработки отдельных программ в языке C# и спецификации интерфейса для переноса готового продукта в Репозиторий системы Eclipse. Эта система отображает связь с исходной средой разработки программ с помощью плагинов и конфигурационного файла с параметрами и операциями обработки данных в среде и среде выполнения Eclipse.;

2) CORBA↔JAVA↔MS.Net обеспечивают разработку программ на ЯП этой среды и устанавливают связи между этими средами с целью размещения разработанных программ в Репозитории Eclipse для предоставления доступа другим разработчикам к этим программам;

3) IBM VSphere↔ Eclipse предоставляет средства для разработки новых программ с использованием ЯП, которые допустимы в среде или в VSphere виртуального варианта системы.

Кроме этих моделей взаимодействия программ и сред исследованы системные средства взаимодействия в WCF (Windows Communication Foundation) с применением протокола IContract.

3.5. Конфигурационная сборка программных артефактов, ПС и СПС

Конфигурационная сборка проводится по стандарту IEEE 828 -1996 после реализации метода сборки Липаева в 1985. Основу стандарта составляет:

модель инженерии домена для разработки отдельных КПИ, артефактов и их сборки;

процессная модель для обеспечения повторного использования» (for reuse) артефактов и разработки ПС и СПП «с использованием КПИ» (with reuse);

модель управления вариабельностью для реализации процесса конфигурации продукта из КПИ с учетом точек в моделях вариабельности.

Модель вариабельности семейства систем СПС представляет собой пару взаимно согласованных моделей: *VM* = {SV, AV*},* где

SV – модель структуры СПС;

AV – модель активов СПС процесса разработки.

Модель *VM* используется для:

– последовательного управления изменчивостью интеграционной структуры СПС;

– представления уровня изменчивости продукции СПС;

– учета затрат и времени для изготовленных систем семейства.

Процесс управления конфигурации включает следующий набор действий:

– систематическое отслеживание внесенных изменений в отдельные КПИ конфигурации, проведения аудита изменений и контроля изменений в каждый отдельный компонент;

– поддержки целостности конфигурации, ее аудит и обеспечение внесения изменений в один из объектов конфигурации, а также в связанный с ним другие объекты;

– контроль конфигурации путем проверки программных или аппаратных элементов в соответствии с вариантом конфигурации и требованиями;

– трассировка изменений конфигурации на этапах сопровождения и эксплуатации ПО.

Компонентная конфигурация **-** это архитетура системы из совокупности компонентов и КПИ, взаимодействующих согласно требований и правил этой модели через интерфейс для каждой пары взаимодействующих между собой компонентов; количество компонентов у компонентной конфигурации может быть произвольным.

Конфигурация СПС с использованием КПИ сокращает время и повышает уровень качества ПС. Ее экспериментальный вариант реализован на сайте ИТК (http://sestudy.edu-ua.net), а также в системных фабриках AppFab IBM, Net, Intel и др. Управление

конфигурацией ПС в СПС из элементов разных парадигм программирования приведен на рис. 2.

К операциям *управления конфигурацией* ПС в СПП относится:

– идентификация конфигурации, элементов конфигурации и данных;

– функции управления процессом изменений;

– модели, методы и метрики оценки вариабельности;

В модельную среду конфигуратора КПИ и артефактов входят:

– схемы формального описания артефактов:

– решения квадратного уравнения с визуальным представлением модели характеристик с отмеченными точками вариантов (рис. 1);

– база данных, включающая требования, архитектуру, набор базовых КПИ и ПС;

– конфигуратор, объединяющий артефакты в ПС и СПП.

В процессе сборки происходит преобразование исходного кода в код или КПИ в СПС, которые могут быть запущены на компьютере или преобразованы в код выполнения. Одним из шагов создания сборщика является процесс компиляции исходного кода, где файлы превращаются в промежуточный код или в код выполнения простых программ.

Для СПС после компиляции ПС происходит процесс связи специальной программой — связником. Процесс связи представляет собой замену относительных адресов функций внешних библиотек в реальные адреса, используемые в программе при выполнении.

Реализация *модели вариабельности* – компактное представление всех продуктов СПП с точки зрения характеристик (функциональности). Эта модель визуально представляется диаграммой, которая содержит характеристики ПС их зависимости, ограничения в виде (деревьев) характеристик или возможных комбинаций (рис.1).

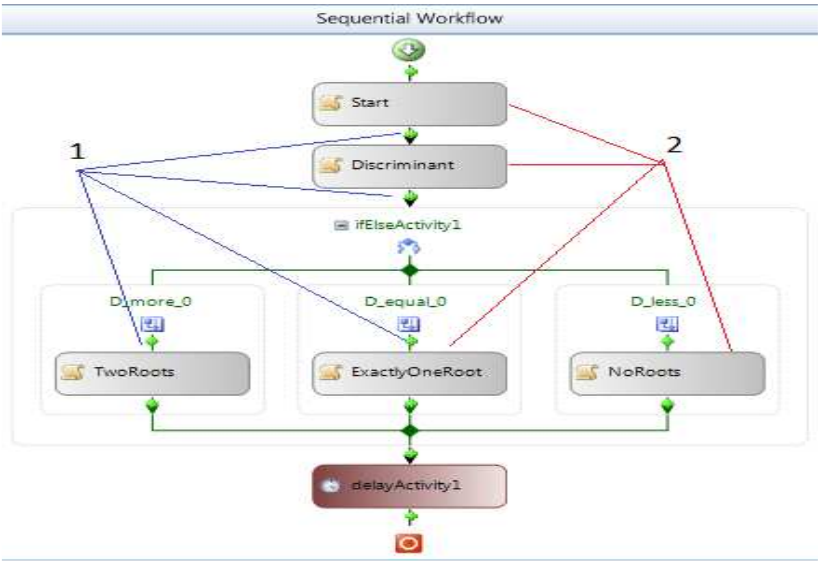


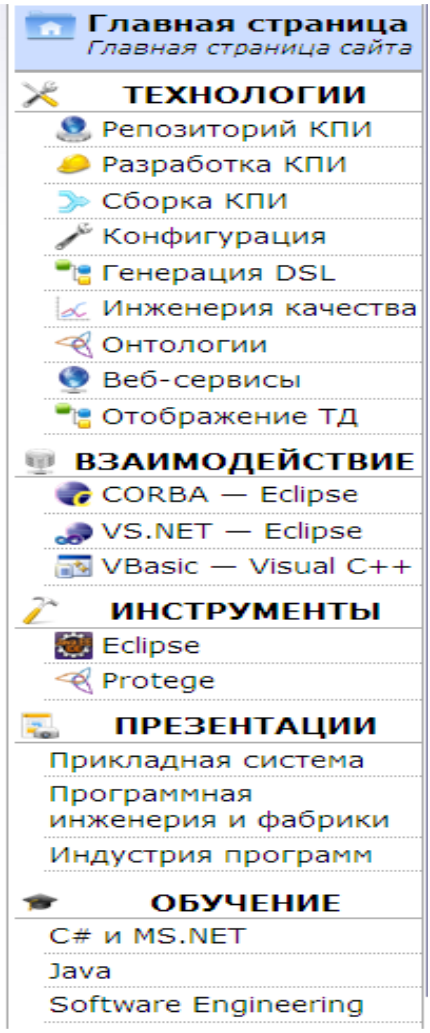
Рис. 1. Дерево решения квадратного уравнения с точками вариантов

Для примера взят алгоритм решения квадратного уравнения в наглядном представлении набора возможных точек вариантности, вариантов и ограничений на их использование. На данном рисунке представлена схема работы алгоритма задачи в виде КПИ, который описан средствами Windows Workflow Foundation (WWF). По точкам вариантов в схеме WWF выполняется управление вариабельностью ПС. WWF среды Visual Studio

разрешает разработчику вставить любой КПИ в заданную точку вариантности и получить продукт из заданной коллекции компонент в Библиотеках.

Конфигуратор реализован в среде ИТК. Он обеспечивает сборку КПИ и компонент, которые могут изменяться. Обеспечивает добавление и поддержку модификации структуры системы по точкам вариабельности. Приведенная схема алгоритма обработана на инструменте WWF в среде Visual Studio. Все компоненты сохраняются в репозитории объектов данной программы. Описанный пример решения задачи нахождения корней квадратного равнения в xoml код есть не что иное, как обычный расширенный xml, который описывает последовательность вызовов команд бизнес логики.

ИТК включает технологические операции для проведения разработки ресурсов, и проведения операций сборки ресурсов в сложную программу обработку данныхти (рис.2):



а

Сайт ИТК содержит следующие технологические разделы.

ТЕХНОЛОГИИ ПОСТРОЕНИЯ ПС:
Технология обслуживания репозитория КПИ,
Технология разработки КПИ,
Технология сборки КПИ,
Технология конфигурирования КПИ,
Технология генерации описания КПИ в языке DSL,
Технология оценка затрат и качества,
Технология онтологии вычислит. геометрии, ЖЦ ISO/IEC12207,
Технология веб-сервисов,
Технология отображения типов данных ISO/IEC 11404.

ВЗАИМОДЕЙСТВИЕ программ, систем и сред:
Модель CORBA – Eclipse–JAVA,
Модель VS.Net C# – Eclipse,
Модель Basic – C++.

ИНСТРУМЕНТЫ ИТК:
Система Eclipse,
Система Protégé.
Система оценки стоимости и качества.
ПРЕЗЕНТАЦИИ ПС В ИТК:
Система ведения зарубежных командировок НАНУ,
Слайды про ИТК, фабрики программ \
Методологии построения ТЛ
Фабрика программ КНУ
ОБУЧЕНИЕ:
Технологии программ в C# VS.Net и JAVA,
МетодикаЯП Java
Электронный учебник "Программная инженерия" на сайте КНУ http://programsfactory.univ.kiev.ua.

б

Рис. 2. Главная страница веб-сайта ИТК ИСП
а – название разделов веб-сайта, б – перечень технологий веб-сайта

Реализованные в ИТК средства и технологии ориентированы на производство ПС и СПС из готовых КПИ по простым линиям обработки КПИ в разных ЯП, которые фактически отображают идею сборочного создания современных программ, систем и их семейств. В ИТК

входит фабрика программ КНУ (http://programsfactory.univ.kiev.ua), реализованная студентами под руководством автора курсов "Программная инженерия" и "Технологии программирования ИС". В 2007г. опубликован учебник по методам и средствам программной инженерии, который читался в МФТИ РАН и используется на сайте www.intuit.ru.

В состав сайта ИТК входят линии, которые разработаны студентами при обучении технологии программирования и программной инженерии:

1. Линия проектирования программ в MS.NET.
2. Представление артефактов в языке WSDL.
3. Сертификация КПИ для хранения в репозитории.
4. Конвейерная сборка КПИ в системы.
5. Учебник по «Программная инженерия»

К сайту обратилось более 35000 из разных стран мира. Есть намерения развивать ее в направлении технологий для изготовления новых механизмов и веществ в рамках e-sciences (биология, физика, химия и др.).

Заключение по подразделу III проекта РФФИ 35202018

В подразделе рассмотрены коллекции CASE-инструментов и средств, включающих системные AppFab и прикладные (Application Fabric) фабрики, специальные фабрики программ (Дж.Гринфильда, Г.Ленца, М.Фаулера, С.Авдошина и др.), продуктовые (Product Line /Product Family) и технологические линии. Все они построены из многоразовых компонентов и reuses и предоставляют средства разработки систем и их семейств. Представлено сборочное программирование, в котором описаны метод сборки, модели вариабельности и интероперабельности, обеспечивающие взаимодействие и изменяемость систем и их семейств.

Рассмотрены формальные аспекты создания КПИ и описания их паспортов в языке WSDL и интерфейсов в языке IDL для сборки и конфигурирования из них изменяемых систем, способных выполняться в современных системных платформах. Метод и модели сборочного программирования реализованы в ИТК в виде коллекции технологий поддержки процессов разработки КПИ, систем и их семейств, взаимодействия систем и их изменений через конфигурацию систем из готовых ресурсов.

– модели и методы моделирования физических, математических и прикладных задач в рамках кибернетики и информатики;

– математический системный анализ, декомпозиция областей знаний и интеграции логических алгоритмов описания задач прикладных областей (ПрО) знаний;

– метод сборки программных комплексов и систем (ПС), семейств систем (СПС), технических систем и OS Linux из готовых элементарных объектов модульного типа для реализации задач авиации, морфлота и космоса ВПК (1975-1989, МНИИПА, В.В. Липаев) на отечественных ЭВМ;

– методы верификации и тестирования готовых ресурсов ПС, СПС и обмениваемых данных с проверкой надежности и качества программ и интерфейсов, а также безопасного функционирования готовых ПС и СПС, собранных из ресурсов (модулей, объектов, компонентов и сервисов) Интернет;

– средства общесистемных сред (MS.Net, IBM-360, Corba, Protégé, Eclipse, BSD, Grid, Etics, Intel, Linux и др.) для связывания модулей в ПС и СПС, функционирующих с данными, передаваемыми через интерфейсный посредник, и их эквивалентного преобразованием с учетом стандарта ISO/IEC 11204 GDT: 1997-2001, 2007 к формату сред Интернет;

– сборка экспериментального варианта OS Linux методами Data Mining и Reuses Mining при анализе ядра OS и генерации варианта ядра для применения в прикладных областях знаний (медицине, биологии, химии, физики, математики и др.;

– использование онтологии Семантик Веб (OWL, RDF, BPMN) и ЯП (Basic, Snobol, Python, Ruby, Java, Smalltalk, C++, WSDL и др.) для описания задач ПрО на ЖЦ с помощью сервисных и готовых ресурсов, накапливаемых в современных Репозиториях и Библиотеках Интернет;

– анализ моделирования физических задач Колайдера в Европейском проекте средствами систем GRID, ETICS и Интеллекта с использованием Big Data, накапливаемых в Базах данных Интернет и проведением конфигурационной сборки стандарта IEEE 828 Configuration -1999 -2007, FNSI/IEEE 829? 1991? 1008? ISO/15846-1998 ТО –Процессы ЖЦ ПС. Конфигурационное управление ПС и др.

– поиск и выбор готовых ресурсов в Библиотеках ИС-2, MatLab, Demral и др. для конфигурационной сборки информационных и интеллектуальных ресурсов прикладных областей знаний. Подана заявка на патент «Сборщик информационных, интеллектуальных и прикладных систем» в ФИПС-21.

Литература к п. 3.1 - 3.5

1. Лаврищева Е.М. Развитие отечественной технологии программирования.–Кибернетика и системный анализ, 2014, т50, № 3.– С. 121–143.
2. Лаврищева Е.М. Методы программирования. Теория, инженерия, практика. Наук. Думка, 2006.– 451 С. (www.twirpx.com).
3. Лаврищева Е.М. Software Engineering компьютерных систем. Парадигмы, Технологии, CASE- Средства программирования.–Научная Думка, 2014.–284с.
4. Software engineering as a scientific and engineering discipline E. M. Lavrishcheva, 2008, Volume 44, Number 3, Pages 324–332
5. Classification of software engineering disciplines, E. M. Lavrischeva 2008, Volume 44, Number 6, Pages 791–796, Software–Hardware Systems.
6. Липаев В.В. Программная инженерия сложных заказных программных продуктов. Учебное пособие.–Мак-Пресс, Москва, 2014.–308с.
7. Липаев В.В. Программная инженерия сложных заказных программных продуктов. Учебное пособие.–Мак–Пресс, Москва, 2014.–308с.
8. Лаврищева Е.М., Грищенко В.Н. Связь разноязыковых модулей.–Москва.– Финансы и статистика.–1982.– 136 с.
9. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование. Основы индустрии программных продуктов.– К.: Наук. Думка, 2009.–371 С. (www.twirpx.com).
10. Лаврищева Е.М., Петрухин В.А. Методы и средства инженерии программного обеспечения. – М.: МОН РФ, 2007. – 415 с.– Aviable (www.intuit.ru).
11. Эммерих В. Конструирование распределенных объектов. Методы и средства программирования интероперабельных объектов в архитектурах OMG/CORBA, Microsoft COM и Java RMI. – М.: Мир, 2002. – 510с.
12. Буч Г., Рамбо Д., Джекобсон А. Язык UML. Руководство пользователя: Пер. с англ. – М.: ДМК, 2000. – 432 с.
13. Pohl K., Böckle G., Linden F.*J.* Software Product Line Engineering: Foundations, Principles and Techniques. – New York: Springer–Verlag. – 2005. – 437 p.
14. Computer Journal of IEEE Computer Society, Face Recognition Technology, November 2013.
15. Ведеров А.М. CASE–технология. Современные методы и средства проектирования информационных систем. –М.: Финансы и статистика, 1998.–176с.
16. Соммервилл И. Инженерия программного обеспечения.– Изд. Дом «Вильямс», Москва, Санкт-Петербург, Киев.– 2002.–623с.

17. Чернецки К., Айзенекер У. Порождающее программирование. Методы, инструменты, применение.– Издательский дом Питер. – М. – СПб. – Харьков. – Минск.– 2005.– 730 с.

18. Уилсон С.Ф., Мєйлс Б., Ленгрев Т. Принципы проектирования и разработки программного обеспечения. Учебный курс MCSD.–Пер. с анг.–М.: Из–во торговый дом «Русская редакция», 2000.–608с.

19. Domain-Specific Languages: An Annotated Bibliography. Arie van Deursen – Paul Klint – Joost Visser. CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands http://www.cwi.nl/arie,paulk,jvisser/ Domain-Specific Languages Languages:An Annotated Bibliography1.htm.

20. Корпорация SAP – http://www1.sap.com/ www.sap.ruvs.net

21. P&P – www.ibm.eebspherею.bisint

22. CASE "Oracle" – http://www.oracle.com

23. Компания ИВК (Россия) http://www.ivk.ru

24. IBM Sphere ПО portal www.01.ibm/com/ software/ru/webspere

25. Глушков В.М., Лаврищева Е.М., Стогний А.А. и др. Система автоматизации производства программ м (АПРОП). – К.: ИК АН УССР, 1976. – 134 С.

26. Guckenheimer S., Perez J.I. Software Engineering with Microsoft Studio Team System.– Crawfordsville, USA: Adison–Wesley, 2006.–304 P.

27. Бей И. Взаимодействие разноязыковых программ.–Диалектика.–М.:С-Петербург–Киев.– Изд. дом. "Вильямс".– 2005.–868с.

28. Гринфильд Дж. Фабрики разработки программ.–Диалектика.–М–С–Петербург–Киев. Изд.дом. "Вильямс".– 2007.–591с.

29. Grid – Distributed Computing at Scale.An overview of Grid and the Open Grid Forum.GWD- Mark Linesch, HP. Marketing April 23, 2007. http://www.ogf.org/ Public_Comment_Docs/Documents/Apr.

30. Lenz G.. Wienands. C. Practical Software Factories in .NET. From theory in practice – a primer reference and case study. – Apress, 2007. – 205 p.

31. Авдошин С.М. Фабрики программного обеспечения. – ttp://www/ewwek.com./on

32. Duval P., Matyas, Grover A. Continuous integration Împroving Software Quality and Reducing Risk, Addison Wesley, 2009. – 691 p.

33. EPAM– systems – разработчик информационных технологий, www.epam.com.

34. Лаврищева Е.М., Коваль Г.И., Бабенко Л.П., Слабоспицька О.А., Игнатенко П.П. Новые теоретические основы технологии производства семейств программных систем в контексте генерирующего программирования, ИПС НАНУ, ВИНИТИ,2011.– 277 с.

35. Аронов А. А. Дзюбенко А. И. Подход к созданию студенческой фабрики программ // Проблемы программирования, №3, 2011.– с.42–49.

36. Lavrischeva E., Aronov A., Dzubenko A. Programs factory – a conception of Knowledge Representation of Scientifical Standpoint of Software Engineering. Jornal of Computer Science, Canadian Senter of Science and Education, ISSN1913-8989, 2013, p.21 – 27.

37. Lavrischeva E., Ostrovski A., Radetskyi I. Approach to E–Learning Fundamental Aspects of Software Engineering.– // 8– th international Conf. ICTERI– 2011 “ICT Education, Research and Industrial Applications”.– Publ. springer –sbm.com/ocs/home/

38. Лаврищева Е.М., Зинькович В.М., Колесник А.Л. и др. Инструментально-технологический комплекс разработки и обучения приемаи производства программных систем.– Гос. Служба Интеллектуальной собственности Украины.–Свидетельство о регистрации авторского права на продукт – № 45292, от 27.08.2012.

39. Островський А.В. Подход к обеспечению взаимодействия программных сред JAVA и MS.Net.– Проблемы программирования, 2011.–№2.–с.37–44.

40. Радецкий ИА. Один из подходов к обеспечению взаимодействия сред MS.Net и Eclipse // Проблемы программирования, №2, 2011.– с.45–52.

41. Ekaterina Lavrischeva, Alexei Ostrovski. New Theoretical Aspects of Software Engineering for Development Applications and E-Learning, Journal of Software Engineering and Applications, 2013, 6, 34–40 –http://dx.doi.org/10.4236/ jsea.2013,.69A004 Published Online September 2013 (http://www.scirp.org/journal/jsea)

42. Lavrischeva E.: Generative and Composition Programming: Aspects of Developing Software System Families. In: Cybernetics and Systems Analysis, vol. 49, no.1, pp. 110–123. Springer, Heidelberg (2013) http://link.springer.com/article/10.1007%2Fs10559-013-9491-6.

43. Lavrischeva E.: Formal Fundamentals of Component Interoperability in Programming. In: Cybernetics and Systems Analysis, vol. 46, no. 4, pp. 639–652. Springer, Heidelberg (2010) http://link.springer.com/article/10.1007%2Fs10559-010-9240-z

44. Ekaterina Lavrischeva, Andrey Stenyashin, Andrii Kolesnyk, Object-Component Development of Application and Systems. Theory and Practice, Journal of Software Engineering and Applications, 2014, 7, Published Online August 2014 in SciRes http://www.scirp.org/journal/jsea

45. Лаврищева Е.М. Парадигмы программирования. УкрПро–2014, Спец. выпуск.– с.94– 112.

46. Lavrischeva, E.: Theory and Practice of Software Factories. Software–Hardware Systems. In: Cybernetics and Systems Analysis, vol. 47, no. 6, pp. 961–972. Springer (2011).m http://link.springer.com/article/10.1007%2Fs10559-011-9376-5

47. Лаврищева Е.М. Взаимодейсвие программ, систем и операционных сред // Проблемы программирования, №3, 2011.– с.13–24.

48. Аронов А. О. Дзюбенко А. І. Подход к созданию студенческой фабрики программ // Проблемы программирования, № 3, 2011. – с. 42–49.

49. Колесник А.Л. Модели и методы разработки семейства вариантных программных систем. – Автореф. – КНУ, 2013. – 22 С.

50. Лаврищева Е.М., Слабоспицька О.А., Коваль Г.І., Колесник А.А. Теоретические аспекты управления вариабельностью в семействах программных систем. – Весник КНУ, серия физ.–мат.наук. – 2011. – №1. – С. 151–158.

51. Гамма Э., Бек К. Расширение Eclipse: принципы, шаблоны и подключаемые модули / Пер. с англ. – М.: КУДИЦ–ОБРАЗ, 2005. – 384 с.

52. Protégé-Frames User's Guide: protégé.Stanford.edu/doc/sydex/php/prf-ug

53. Lavrischeva E., Ostrovski A. General Disciplines and Tools for E–Learning Software Engineering.– 9– th International Conf. ICTERI–2012 Springer.com, Communication in Computer and Information Sciences, ISSN 1865-0929.-http://senldogo0039.springer-sbm.com/ocs/

54. Kolesnik, A, Koval, G.: The Theoretical View for Software Family Variability Management [in Ukrainian]. In: Bulletin of University of Kiev. Series: Physics & Mathematics, vol. 1, pp. 151–158. Kiev (2011)

55. Lavrischeva, E., Slabospitcka, O.: An Approach for Expert Assessment in Software Engineering. In: Cybernetics and Systems Analysis, vol. 45, no. 4, pp. 638–654.

56. Kolesnik, A, Koval, G.: The Theoretical View for Software Family Variability Management [in Ukrainian]. In: Bulletin of University of Kiev. Series: Physics & Mathematics, vol. 1, pp. 151–158. Kiev (2011)

57. Kolesnyk A., Slabospitskaya O. Tested Approach for Variability Management Enhancing in Software Product Line. – In: Ermolayev V., Mayr H.C., Nikitchenko M. et al. (eds.): Proc. 8-th Int. Conf. ICTERI 2012, Kherson, Ukraine, June 6-10, 2012, CEUR-WS.org/Vol-848, ISSN 1613–0073, urn:nbn:de:0074-848-8. – P. 125–133. – Available at http://ceur- ws.org/Vol-848/ICTERI-2012-CEUR-WS- paper-31-p-155-162.pdf

58. Ekaterina Lavrischeva, Andrey Stenyashin, Andrii Kolesnyk, Object-Component Develop ment of Application and Systems. Theory and Practice, Journal of Software Engineering and Applications, 2014, 7, Published Online August 2014 in SciRes http://www.scirp.org/journal/jsea

59. Lavrischeva, E., Slabospitcka, O.: An Approach for Expert Assessment in Software Engineering. In: Cybernetics and Systems Analysis Cybernetics, 2008. vol. 45, no. 4, pp. 638–654.

60. Лаврищева Е.М., Петренко А.К. Технология сборки интеллектуальных и информационных ресурсов Интернет // Научный сервис в сети Интернет: труды XXI Всероссийской научной конференции (23-28 сентября 2019 г., г. Новороссийск). М.: ИПМ им. М.В.Келдыша, 2019. — С. 469-488. — URL: http://keldysh.ru/abrau/2019/theses/93.pdf doi:10.20948/abrau-2019-93

61. Лаврищева Е.М., Зеленов С.В., Рыжов А.Г.. Теория моделирование программно-технических систем с обеспечением безопасности и надежности в среде Web-services Интернет. Труды ИСП РАН, Том 30. № 6.- с.156-187.

62. Е.М.Лаврищева, И.Б.Петров. Моделирование технических и математических задач прикладных областей знаний на ЭВМ. Труды ИСП РАН 2020, Том 32 № 6 с.167-182.

63. Модельный подход к обеспечению безопасности и надежности Web-сервисов. Е.М. Лаврищева, С.В. Зеленов. Труды ИСП РАН 2020, Том 32 № 5. С.151-163.

64. Лаврищева Е.М. Теория графового моделирования сложных систем из модульных элементов для прикладных областей. Austria-science», 1 часть №28/2019. - p.12-30. http://austria-science.info

65. Е.М.Лаврищева, И.Б.Петров. Теория моделирования технических и математических задач предметных областей знаний. Евроазиатское Научное Объединение. 2021.№ 1- (1).-с.35-43.

3.6. Научный сервис и Е-наука Интернет для создания предметных областей знаний

Данный подраздел является дополнительным в проекте РФФИ и посвящен анализу состояния науки системного сервиса в Европейском физико-техническом Международном проекте, ставящего своей целью создания «Коллайдера» путем реализации задач исскуственного интеллекта и информационных систем в E-Science Интернет. Результатом анализа и исследования данной проблематики были доклады в МФТИ и опубликованы статьи:

Е. М. Лаврищева, Л. Е. Карпов, А. Н. Томилин. Подходы к представлению научных знаний в Итернет науке. Сб. XIX Всероссийский научной конференции «Научный сервис в сети Интернет», Новороссийск, 18-23 сентября 2017.-с.310-326.

Лаврищева Е. М., Карпов Л. Е., Томилин А. Н., Семантические ресурсы для разработки онтологии научной и инженерной предметных областей, Сб. "Научный сервис в сети Интернет" 19-24 сентября 2016, Абрау, с.126-138.

Используемые системные средства полезны полезны всем пользователям вычислительных машин и относятся к службам управления обработкой прикладных программ, занесения их в долговременную память, в стандартные библиотеки для последующего использования многими пользователями. В рамках Европейского проекта сформировалось понятие системного сервиса (или "службы"), которые со временем приобрели стандартное представление, содержащее информацию о реализованных системных функциях Семантик Веб и их применения при решении разных вычислительных задач и бизнес приложений. Одним из наиболее известных средств обеспечения системного сервиса по управлению процессом разработки объектных программ и систем являлась система CORBA (http://www.corba.org). Она предоставляет сервисы по управлению разработкой систем на разных языках программирования (ЯП), обеспечивает взаимодействие разных систем при решении некоторой задачи и передачи данных через интерфейсы IDL, обрабатываемые брокером объектных запросов ORB. Система находит применение как самостоятельно, так и внутри других общих систем.

В сети Интернет появилось огромное количество других видов сервисов, которыми управляют Web сервера. В e-sciences возник научный проект, под названием Grid (2000 г.) с

целью создания компьютерной инфраструктуры нового типа, обеспечивающей глобальную интеграцию информационных и вычислительных ресурсов, специального программного обеспечения и набора стандартизованных служб для обеспечения совместного доступа к географически распределенным ресурсам. Разработан принципиально новый подход к обработке огромных объемов экспериментальных данных, к моделированию сложнейших физических процессов и созданию бизнес-приложений с большими объемами вычислений. Grid включает набор систем по реализации разных научных задач. Одной из систем является ETICS. Для построения сетевых приложений (Web Application) в ней представлены сетевые службы (Web Services), доступные глобальным пользователям.

За последние 10 лет комитет W3C разработал набор стандартов (протоколов) и языков описания программ, процессов и технологий для решения разного рода задач и бизнес-приложений в среде Интернет.

Далее рассматриваются системные и прикладные сервисы, представленные в рамках систем CORBA, W3C, Grid, Etics и др. для решения физико-технических задач.

Системные сервисы. Системные сервисы обеспечивают решение разного рода прикладных, деловых и бизнес-задач. К ним относятся:

общие сервисы системных сред для поддержки процессов и функций обработки программ и данных (например, службы именования, каталогизации и др.);

объектные сервисы, которые управляют объектами, классами и услугами по формированию и обработке объектно-ориентированных систем (например, службы диспетчеризации объектными запросами, управления интерфейсом и др.);

сетевые сервисы стандартной модели OSI, моделей SOA (Service-oriented Architecture), SCA (Service-Component Architecture), как инструменты представления и обработки ресурсов в сети Интернет, которые реализуют деловые, финансовые, экономические и другие услуги при решении соответствующих задач;

готовые программные и информационные ресурсы (services, artifacts, reuses, assets и др.), используемые как многоразовые услуги при решении разных задач в e-science и других прикладных областях.

Некоторые из сервисов стали обязательной частью общесистемных средств (VS.Net, IBM, Intel, Linux и др.), другие используются в специальных областях (например, медицина, биология) в плане предоставления услуг при работе с современными данными (FDT, GDT, Big Data).

Каждая служба определяется именем, по которому осуществляется поиск в распределенной среде пространства имен через транзакции, устанавливающие соответствие “имя-объект” для организации и управления отдельными сервисными ресурсами глобальной сети, а также с помощью сообщений для визуального общения с требуемыми представителями отдельных ресурсов.

Перечисленные виды сервисов используются при моделировании программных систем (ПС) из готовых ресурсов сети Интернет. Для их использования при создании ПС требуется проводить поиск подходящего сервисного ресурса, его апробацию и встраивание в прикладную программу решения задачи, либо использовать его в динамическом режиме.

Системные сервисы CORBA. Полный набор сервисов для объектов создан в системе CORBA (см. 1-6). В ней впервые реализована объектная модель и системные сервисы для работы с объектами:

брокер объектных запросов (*Object Request Broker* — ORB), обеспечивающий взаимодействие объектов в разных языках программирования;

общие объектные сервисы (*Common Object Services* — COS), обеспечивающие сервис всем объектам по вопросам управления изменениями, реализациями, контролем,

транзакциями, подпроцессами и т.п.;

общие средства обслуживания (*Common Facilities* — CF) или общие услуги, предоставляющие ряд общих прикладных функций, которые могут объединяться в различные конфигурации с учетом заданных требований (например, средства печати, БД, электронная почта и др.);

объектные приложения (*Application Objects* — AO), к которым относятся приложения и их компоненты, реализующие задачи и объекты пользователя, функционирующие в объектной среде, и над которыми могут производиться операции типа - открыть, инсталлировать, переместить и поместить.

Общие объектные сервисы и общие средства обслуживания способствуют разбиению приложения на функции — базовые для большинства приложений, либо достаточно общие для широкого класса приложений. Объектные приложения и общие средства обслуживания обеспечивают функции и сервисы объектными интерфейсами, а также позволяют адаптироваться к новым поколениям сетей, языков и сред.

Система CORBA ориентирована на обеспечение взаимодействия удаленных объектов в распределенной среде типа OpenStep. Для задания взаимодействия объектов используется язык интерфейсов IDL (Interface Definition Languages) объектов. Интерфейсы в IDL запоминаются в репозитории интерфейсов (*Interface Repository*), а реализации объектов — в репозитории реализаций (*Implementation Repository*). Независимость интерфейсов от реализаций объектов позволяет использовать их разными приложениями статически и динамически.

Объект-клиент и объект-сервер обмениваются между собой с помощью запросов, каждый из которых исполняется брокером ORB с помощью компонентов, создаваемых на основе описания интерфейсов клиента, сервера и ядра ORB.

Интерфейс клиента (*Client Interface*) обеспечивает взаимодействие с объектом-сервером и состоит из трех базовых интерфейсов:

stub-интерфейса содержит описание внешне видимых параметров и операций объекта в IDL;

интерфейс динамического вызова (Dynamic Invocation Interface — DII) объекта, определяемого во время выполнения программы клиента посредством поиска описания интерфейса в репозитории интерфейсов или в репозитории реализаций;

интерфейса сервисов ORB (ORB Services Interface), содержащего набор сервисных функций, которые клиент запрашивает у сервера через брокер ORB.

Stub-интерфейс обеспечивает взаимосвязь клиента с ORB. Прикладная программа клиента через посредника (заместителя) *stub* — статической части программы клиента, посылает в запросе параметры, которым сопоставляются соответствующие описания из репозитория интерфейсов. Брокер ORB выполняет полученный запрос и пересылает результаты клиенту. Рассмотренная схема взаимодействия клиента с объектом соответствует схеме вызова удаленных процедур через RPC-механизм.

Интефейс DII обеспечивает доступ объектов и их интерфейсов во время выполнения. Этот интерфейс предоставляет механизм запроса к объектам, интерфейс которых становится известным во время выполнения. Он становится доступным с помощью вызовов брокера ORB. В каждом вызове указывается тип объекта, тип запроса и параметры. Эту информацию посылает прикладная программа, но она может извлекаться из репозитория интерфейсов или репозитория реализаций.

Объекты специфицируются средствами языков программирования и могут быть реализованы на разных платформах и в разных средах. Интерфейсы программ-посредников описываются на языке IDL. Заместитель клиента (stub) выполняет сервисные функции, связанные с преобразованием типов данных клиентских компонентов к стандартным системным типам, а заместитель сервера (skeleton) преобразует стандартное представление данных в типы данных сервера.

Описание интерфейса в IDL начинается с ключевого слова interface, за которым следует: имя интерфейса, описание типов параметров и операций (op_dcl) вызова объектов:

interface A { ... }

interface B { ... }

interface C: B, A { ... }.

Параметры операций (op_dcl) в задании интерфейсов это:

тип данных (type_dcl);

константа (const_dcl);

название исключительной ситуация (except_dcl), которая может возникнуть в процессе выполнения метода объекта;

атрибуты параметров (attr_dcl).

Описание типов данных (ТД) начинается ключевым словом typedef, за которым следует базовый или конструируемый тип и его идентификатор. В качестве константы может быть некоторое значение типа данного или выражение, составленное из констант. ТД и константы описываются как фундаментальные типы данных: integer, boolean, string, float, char и др.

Описание операций op_dcl передачи данных включает в себя:

наименование операции интерфейса;

список параметров (от нуля и более);

типы аргументов и результатов, иначе – *void*;

управляющий параметр или описание исключительной ситуации и др.

Атрибуты передаваемых параметров начинаются служебными словами: in – при отсылке параметра от клиента к серверу; out – при отправке параметров-результатов от сервера к клиенту; inout **–** при передаче параметров в оба направления (от клиента к серверу и обратно).

Описание интерфейса для одного объекта может наследоваться другим объектом и тогда это описание становится базовым, например:

const long l=2

interface A { void f (in float s [l]); }

interface B { const long l=3; }

interface C: B, A { }.

Интерфейс ***C*** использует интерфейс ***B*** и ***A***. Это означает, что интерфейс ***C*** наследует описание типов данных ***A*** и ***B***, которые по отношению к ***C*** являются внешними. При этом синтаксис и семантика остаются неизменными. Согласно приведенному примеру - операция функции *f ( )* в интерфейсе ***C*** наследуется из ***A***.

Механизм наследования интерфейса состоит в сохранении имен объектов без их переопределения. Это касается описания операций, которые должны иметь уникальные обозначения. Имена операций могут использоваться динамически во время выполнения интерфейса *Skeleton.*

Общая структура описания модуля с интерфейсом в языке IDL имеет вид:

Request Operations

module CORBA {

i**nterface** Request {

Status add-arg (

in Identifier name,

in Flags arg_flags);

Status invoke (

in Flags invoke_flags // invocation flags);

Status send (Status get_response (**out** Flags response_flags // response flags));};

Тип данных описывается в классе FDT, GDT, которые передаются через параметры операторов RPC, RMI, а также протоколами в WCF VS.Net и др.. ТД описывается на ООЯ (C#, VBasic, Pascal, и др.).

Входные и выходные интерфейсы (например) для программ P_1, P_2 имеют разную семантику, но одинаковое синтаксическое описание на некотором языке программирования. Передача данных от этих программ для P_3 осуществляется через функции $F_1(\ldots)$, $F_2(\ldots)$ и интерфейсы ***In***, ***Out***, с помощью которых осуществляется преобразование ТД переданных между P_1, P_3 и P_2, P_3 туда и обратно.

Данный аппарат интерфейса реализован в системах CORBA, ОС IBM, Microsoft и др. Его основу составляют библиотеки VS.Net (CRL, CTS, FCL, CIL и др.) преобразования ТД, которые применяется при интеграции разнородных программных объектов.

Общие объектные службы предоставляют набор операций для работы с разными категориями объектных приложений:

наименование и поиск объектов по именам, либо по свойствам и атрибутам;

управление жизненным циклом объектов;

параллельного обращения к объектам;

определение очередей запросов к объектам;

иерархия транзакций;

защиту объектов от несанкционированного доступа (авторизация и аутентификация клиентов и др.).

Интерфейсные средства IDL используются для обеспечения взаимодействия распределенных систем прикладного, коммерческого и бизнес типа, а также входят в состав общесистемных систем (VS.Net, Java и др.),

3.6.1 Сетевые технологии в W3C Интернет

Сеть Интернет базируется на стеке протоколов TCP/IP, за которые отвечает международная некоммерческая организация ISO/IEC (Internet Society). насчитывает более чем 20 тысяч представителей, свыше 100 организаций из 180 стран мира и предоставляет основу для других консультативных и исследовательских групп в том числе:

IETF открытое международное сообщество ученых, инженеров, провайдеров которое занимается развитием протоколов и архитектуры Интернет;

ICANN (Corporation for Assigned Names and Numbers) — международная некоммерческая организация, которая координирует предоставление имен и адресов в Интернете.

Структура TCP/IP базируется на четырехуровневой модели сетевого взаимодействия, которое разработано Министерством обороны США и в основном отвечает 7 - модели OSI.

В Интернете используется общий язык для передачи данных глобальной информационной среды, стандартные методики и форматы представления данных и обмен протоколами. Использовались технологии ISOC, W#C, OASIS, GGF.(Grid systems).

WEB технологии

Глобальная информационная сеть (WEB-среда) объединяет десятки миллионов документов по всему миру и развивается благодаря стандартам консорциум W3C. Консорциум W3C объединяет наиболее крупных производителей программного обеспечения для WEB технологий (Web серверов и WEB браузеров). W3C обеспечивает компьютерным программам взаимодействие в сети (так называемая «сетевая интероперабельность»).

Стандарты глобальной информационной сети можно подразделить на 4 группы.

1). Представление форматных данных пользователя

HTML язык разметки гипертекста. и документов. в виде .ASCII текста, фрагменты которого облагаются специальными пометками (тегами);

XHTML расширяемый язык разметки гипертекста (Extensible Hypertext Markup Language) на основе принципов и синтаксиса XML:

CSS каскадные таблицы стилей (Cascading Style Sheets) для писания внешнего вида документа (страницы);

MathML (Mathematical Markup Language) – язык математических формул, который использует формат XML для отображения математических формул. Используется для отображения формул в Web браузерах MathML;

SVG (Scalable Vector Graphics) – язык масштабируемой двумерной векторной графики, которая использует формат XML.

2). Представление структурированных данных

XML Schema – язык для определения правил, которым должен удовлетворять документ и состоять из набора тегов и их атрибутов для описания соответствия документа его предметной отрасли.;

XLink (XML Linking Language) – XML схема, которая рекомендована W3C для организации ссылок между ресурсами;

XInclude (XML Inclusions) – XML схема, которая рекомендована W3C и предоставляет механизм включения в XML-документы текстовых файлов и др.;

XSL (eXtensible Stylesheet Language) – семейство рекомендаций (стандартов) которое определяет методологию превращения XML документов для визуализации или другой обработки соответствующими программными средствами;

XSLT декларативный язык, который интерпретирует правила и применяет их к входным документам;

DOM (Document Object Model) – не зависимая от платформы и языка программирования модель, который позволяет создавать динамические Веб страницы посредством скриптовых языков (JScipt, JavaScript);

XML Encryption – определяет порядок шифровки и дешифровки содержания элементов XML документа;

XKMS XML Key Management Specification) – определяет порядок безопасной передачи и регистрации открытых ключей для шифровки и дешифровки XML документов за протоколом XML Encryption;.

PNG (portable network graphics) – формат для сохранения растровой графики;

SMIL (Synchronized Multimedia Integration Language) и язык мультимедийной интеграции на синтаксические основы формата XML.

3). Протоколы удаленного выполнения программных сервисов

SOAP (Simple Object Access Protocol) – определяет порядок обмена сообщениями (данными) в WEB среде.;

WSDL (Web Services Description Language) – язык для описания Веб сервисов и доступа к ним на основе XML.

4). Представление семантических данных

RDF (Resource Description Framework) – ряд стандартов, который определяет базовые методы формального представления знаний для машинной обработки;

OWL (Web Ontology Language) – язык, который определяет правила описания онтологии предметных отраслей и предназначен для обеспечения одинаковой и однозначной интерпретации документов разными агентами в распределенной среде.

Наиболее распространенным является протокол SOAP**.** При описании сервиса указывается адрес URI (Uniform Resource Identifier) и транспортный протокол (например, HTTP). Средством описания функциональности сервиса является язык WSDL (Web-service description language). Для представления данных, в особенности метаданных, используется модель RDF. Описание процессов представления и обработки запросов на сервисы в графическом виде осуществляется языками:

WSCI (Web Services Choreography Interface,

WSCL (Web Services Conversation Language,

BPMN (Business process and model and notation,
BPEL (Business Process Execution Language for Web Services) и др.

3.6.2. Сервисно-ориентированная архитектура SOA и SCA

Модель SOA (Service-oriented Architecture) задает сервисно-ориентированную архитектуру ПС, а ***модель SCA*** (Service-Component Architecture) – архитектуру ПС на основе сервисов и компонентов. К средствам моделирования сложных систем относится система WebSphere Integration Developer компании IBM. Она предоставляет сервис-ориентированную архитектуру SOA и SCA, в виде use case языка UML. Эта система обеспечивает интеграцию сервисов SCA через модель интерфейсов JAVA, задаваемую в языке WSDL и классах JAVA. Эта модель дает доступ к сервисным компонентам и определяет зависимость между ними через аппарат ссылок. Они упаковываются в модуль для выполнения на сервисном модуле **WebSphere Process Server**, который эквивалентен EAR-файлу J2EE и некоторым другим. Подмодули J2EE и артефакты упаковываются с модулем SCA, который затем запускает сервис и передает данные для их интеграции.

Механизмы, которые используются для вызова внешнего сервиса, названного *импортом* и *экспортом*, связаны с другими технологиями, такими как JMS, Enterprise JAVA Beans или веб-сервисы. SCA модуль может обратиться к существующему Enterprise JAVA Bean для обеспечения релевантного представления в *универсальной модели данных*, а также обмена данными друг с другом через SDO. В этой модели объекты данных представлены в JAVA common.sdo.DataObject и включают в себя метод, который позволяет пользователям получать свойства данных. WebSphere Integration Developer используется на платформе Eclipse 3.0.

Модель SOA сервисных объектов предоставляет набор принципов и средств создания системного ПО и прикладных ПС из совместимых и унифицированных сервисов.

Модель группируется на серверной стороне из некоторого количества согласованно реализованных сервисов и их служб. Группы задают открытый интерфейс, содержащий описание типов входных/выходных параметров каждого сервиса и портов обмена метаданными в языке WSDL. Для WSDL созданы компиляторы, позволяющие получать серверные и клиентские заместители и учитывающие особенности конкретных программных платформ, в том числе языки программирования на этих платформах, которые описывают реализуемые операции.

Унификация сервисов состоит в типизации функциональности сервиса и его характеристик, а также языков описания сервисов и их взаимодействия. Объект SOA обладает специфицированной функциональностью и качеством. В его модели используются две технологии, которые обеспечивают функциональность (Functions) и качество сервисов (Quality service). Эти технологии вынесены на уровень IT-стандартов комитета W3C.

Технология обеспечения *функциональности* веб-сервисов включают в себя:

1) транспортный уровень (transport layer) для обмена данными;
2) коммуникационный уровень (service communication layer) протоколов;
3) сервисный уровень (service description layer) и связанные с ним интерфейсы;
4) уровень бизнес-процессов (business process layer) для реализации бизнес-процессов и потоков работ через механизмы веб-сервисов;
5) уровень реестра сервисов (service registry layer), который обеспечивает библиотеку веб-сервисов для их публикации, поиска и вызова WSDL-описаний интерфейсов.

Технология обеспечения *качества* веб-сервисов имеет следующие уровни:

1) политики (policy layer) для описания правил и условий применения веб-сервисов;
2) безопасности (security layer) для описания вопросов безопасности веб-сервисов и функционирования (авторизация, аутентификация и распределение доступа);
3) транзакций (transaction layer) для установления параметров обращения к веб-сервисам и обеспечению надежности их функционирования;

4) управление (management layer) веб-сервисами.

Технологическую основу составляют: XML, SOAP, UDDI, WSDL. С их помощью осуществляется реализация базовых свойств веб-сервиса и механизмов взаимодействия между собой веб-сервисов в среде SOA. К ним относятся:

1) провайдер сервиса осуществляет реализацию сервиса, прием и выполнение запросов пользователей, публикацию сервиса, от из реестра сервисов;

2) реестр сервисов содержится библиотеку сервисов для поиска и вызова сервиса по запросам от поставщика или провайдера сервисов, предоставляющих сервисы;

3) потребитель или пользователь сервиса осуществляет поиск и вызов необходимого сервиса из реестра описания сервисов, а также использует сервис, предоставленный поставщиком в соответствии с его интерфейсом (рис. 3).

Посредником между этими службами и приложениям является *провайдер*, который обеспечивает взаимодействие меэжу поставщиками и провайдерами с помощью средств описания и передачи сервисов WSDL, SOAP, XML.

Для получения сервиса в архитектуре SOA выполняются следующие операции:

– публикация сервиса WSDL с целью обеспечения доступности пользователю сервиса и его интерфейса;

– поиск сервиса в реестре с помощью протокола SOAP и заданных критериев;

– связь с реестром UDDI через описание пользователем необходимого сервиса.

При этом предусматривается, что в реестре архитектуры SOA содержится описание сервиса с форматом запросов пользователя к провайдеру, содержащему в себе перечень описаний сервисов, которые могут быть вызваны соответственно с опубликованным интерфейсом сервиса.

К базовым функциям управления компонентами и службами относятся:

– поиск необходимых ресурсов (компонентов, reuses, assets, artifacts, служб и др.);

– доступ к названным ресурсам;

– организация обмена информации между компонентами, ресурсами и службами;

– динамическое управление функционированием заданной совокупности ресурсов.

Модель сервисов ПС базируется на унификации и совместимости, что позволяет рассматривать ПС как набор сервисов, функциональности и взаимодействия. Унификация достигается путем:

–типизации функциональности сервиса и их других характеристик;

–применения унифицированных языков для описания сервиса и их взаимодействия;

–использования стандартных базовых технологий.

Отдельный класс средств унификации составляет **онтология, включающая** модели и словари, которые обеспечивают согласование терминов и понятий языка описания сервисов на уровне семантики. В качестве стандартных базовых технологий при реализации сервисов используются: модель клиент-сервер, унифицированные коммуникационные протоколы, компонентные модели и т. д.

Смысл SOA состоит в том, чтобы создавать небольшие компоненты и собирать их в большой распределенный по глобальной сети комплекс под бизнес-задачи конкретного клиента. Функциональные сервисы, как обычные программные модули, могут быть распределены по вычислительным системам и обладать способностью к взаимодействию посредством локальных и/или глобальных сетей. Интерфейс таких модулей не зависит от технологии или платформы, в рамках которой они реализованы.

Бизнес-процесс определяется как набор взаимосвязанных задач, относящихся к деловой деятельности, имеющий начальные и конечные точки для повторения задачи.

Сервисно-компонентная архитектура SCA. Модель **SCA** предназначена для работы с компонентами со спецификациями, разработанными различными компаниями: компонентами EJB сервера приложений J2EE, сетевыми сервисами, объектами планирования, доступа к базам данных, к информационной системе предприятия и др.

Модель SCA обеспечивает доступ к сервисным компонентам и определяет зависимости между ними через аппарат ссылок. Механизмы, которые используются для вызова внешнего сервиса, называются импортом и экспортом. Элементы SCA могут компоноваться и обмениваться данными друг с другом, пересылая сервисные объекты данных, подготовленные в необходимом виде. Этот интерфейс включает определение метода получения и установления свойства данных. В рамках модели SCA сервисы могут собираться в различные образования (хореографии). Они используют архитектуру SOA и/или создают новые сервисы для их комбинирования и конфигурирования.

Характеристики приложения можно согласовывать со специфическими потребностями клиента (конфигурирование), используя такие элементы, как стандарты бизнеса, бизнес-правила, бизнес-сервис и параметры конфигурации (рис. 3).

Модель SCM представляет собой обобщение объектно-компонентной модели семейства программных продуктов. В ней каждый член является: системой удаленных reuses, которые обмениваются гетерогенными данными и предоставляют сервисы реализации с множеством общих свойств; композицией сетевых сервисов, которые поддерживают некоторый деловой процесс. Данная модель ориентирована на обеспечение адаптивности ПС к переменным условиям использования запросов к функциям и обрабатываемым ими гетерогенных данных. В интересах этой поддержки для программной реализации ПС используются механизмы сервисных объектов данных и сервисов доступа к ним. Они позволяют размежевать код ПС и код доступа к обрабатываемым данным. В состав модели SCM входят подмодель *абстрактных сервисов, интерфейсная* подмодель и подмодель объектных *ресурсов*.

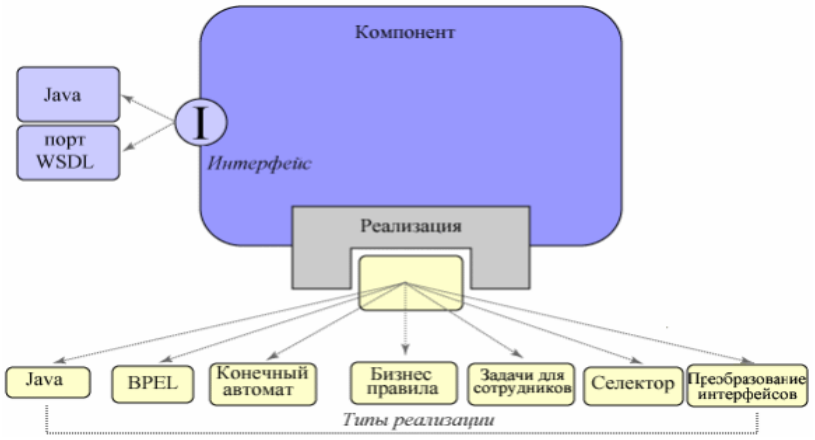


Рис. 3. Общая схема сервера SCA IBM

Сервер приложений Java Enterprise Edition содержит набор спецификаций на языке Java, которые необходимы при работе с сетевыми программами и средствами. К ним относятся:

– динамическая генерация серверных страниц (Java Server Pages);
– сетевые службы;
– компоненты повторного использования (Enterprise Java Beans);
– служба обмена сообщениями (Java Message Queue) и другие сервисные технологии.

Сетевая служба идентифицируется с помощью универсального ресурсного идентификатора URI, ее ресурсы (свойства и методы) описываются на специальном языке WSDL. Доступ к ресурсам осуществляется через протокол SOAP, который представляет собой XML-запросы, передаваемые посредством интернет-протоколов относительно высокого уровня (HTTP, SMTP). Сетевые службы соответствуют объектам объектно-ориентированных языков программирования с некоторыми важными отличиями.

Ключевым понятием сетевой службы **SCA** является сообщение, которое состоит из одной или нескольких переменных. Вместо методов классов в сетевых сервисах используются операции, которые определяются входным и выходным сообщениями. Для описания общедоступных ресурсов сетевых сервисов в язык WSDL, построенный на синтаксической основе языка разметки XML, введены возможности описания данных различных типов. В качестве переменных для сообщений можно использовать последовательности, созданные из фиксированного количества переменных простых типов, причем типы, которые будут использоваться в службе, декларируются заранее.

3.6.3. Среда взаимодействия систем WCF

Программная среда Windows Communication Foundation (WCF) входит в состав среды .NET Framework и является логическим развитием технологий сетевых служб, .NET Remoting и DCOM. В основе WCF лежит модель SOA, которая обеспечивает на сервере работу некоторого количества сервисов, определенных в интерфейсе для задания абстрактных входных/исходных параметров. Эти операции описываются на WSDL и могут быть сделаны доступными через, так называемые mex-endpoints (Metadata Exchange Endpoints), что позволяет получить "метаданные" сервиса. Подключаясь к этому интерфейсу; можно получить описание сервиса и всех его операций, а также сгенерировать соответствующий прокси-класс (класс-заместитель) для заданного языка или платформы. Сервис описывается в языке WCF и используется с языками Java / Python / Ruby и т. п. Клиенты в свою очередь имеют на своей стороне прокси-классы, которые содержат ссылку на операции к сервису.

В WCF MS.Net работает технология фабрик программ (AppFab), основанная на нескольких веб-службах и фабрике сервисов. Основу технологии составляют схемы, "рецепты", методы и средства построения программ разного назначения. Фабрика программ включает в себя набор ресурсов, блоков кода, документации, образцов приложений, инструментов и патернов VSIP для создания из них разных пакетов, которые накапливаются в глобальном хранилище Global Bank. Фабрика сервисов содержит рекомендации, схемы и методы, а также стандарты по проектированию и конструированию продуктов, Функционирование WCF определяется так называемыми конечными точками (endpoint), которые устанавливают "ABC" или "Address-Binding-Contract" (см. рис.4).

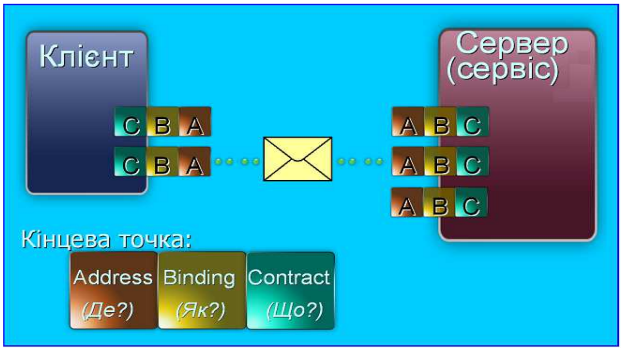


Рис.4. Точки в WCF

Каждая точка выполняет определенную роль:

"Address" задает место расположения конечной точки по абсолютному или относительному адресу;

"Buinding" задает связь с транспортным протоколом, на основе которого будет происходить взаимодействие. Связь в виде классов-привязок (BasicHttpBinding в HTTP, NetTcpBinding в TCP и т. д.);

"Contract" задает контракт, на основе которого будет происходить взаимодействие клиента и сервиса с помощью операций сервиса, который строит класс-прокси на стороне клиента.

На сервисной стороне задается множество конечных точек. В результате, сервисная сторона распределенного приложения будет выглядеть как совокупность конечных точек.

Инструменты распределенного действия WCF или .Net Remoting построены по принцип «слоенного пирога», каждый слой которого отвечает за свой конкретный уровень абстракции и не знает ничего о нижележащих уровнях. Инфраструктура WCF состоит из двух главных уровней: Service Model Layer и Channel Layer. Первый уровень расположен ближе к самому сервису клиента и обеспечивает превращение метода с параметрами в сообщение для передачи более низкому уровню. Канальный уровень (Channel Layer) инкапсулирует канал передачи данных, которых может быть много, такие как транспорт TCP, Http, Named Pipes и т. д. Каждый из этих уровней содержит подуровни, и может включиться в каждый из них.

Контракты представляют собой описание сообщений, переданных конечными службами с возвратом. Конечная точка должна специфицироваться и может выполняться в формате ожидаемых данных. Совокупность этих спецификаций и есть контракт.

WCF содержат три вида контрактов:

1) контракты сервисов для описания функциональных операций, реализованных сервисом. Внутри контракта имеются операции сервиса, которые реализуют функции;

2) контракты данных определяют формат данных, которыми будут обмениваться сервисы. Это относится к запросу на сервис и октету сервиса. Если используются примитивные типы – ***int***, ***string*** и др., то контракт не нужен, потому что в .Net имеется возможность сериализации и десериализации типов. Для комплексных типов – *Customers*, *Order* и др., необходимо указать принцип сериализации и десериализации;

3) контракты сообщений, как тип контракта, который используется для того, чтобы получить контроль над заголовком SOAP (рис.5).

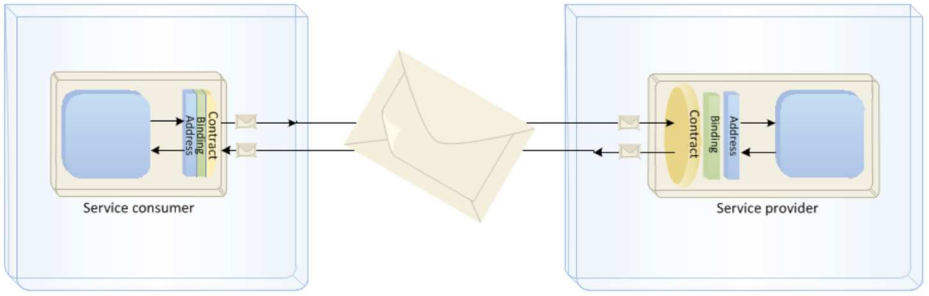


Рис. 5. Передача пакета сообщения между потребителем и поставщиком

В пакете передается протокол SOAP в виде:

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env="http://www.cbsystematics.com">
<!-Конверт протокола SOAP->
<env:Header>
<!- Заголовок протокола SOAP->
</env:Header> <env:Body>
<!-Тело протокола SOAP->
</env:Body> </env:Envelope>.
```

При взаимодействии систем общего назначения друг с другом через сообщения в конверте могут возникнуть разного конфликты передачи данных (рис. 6)

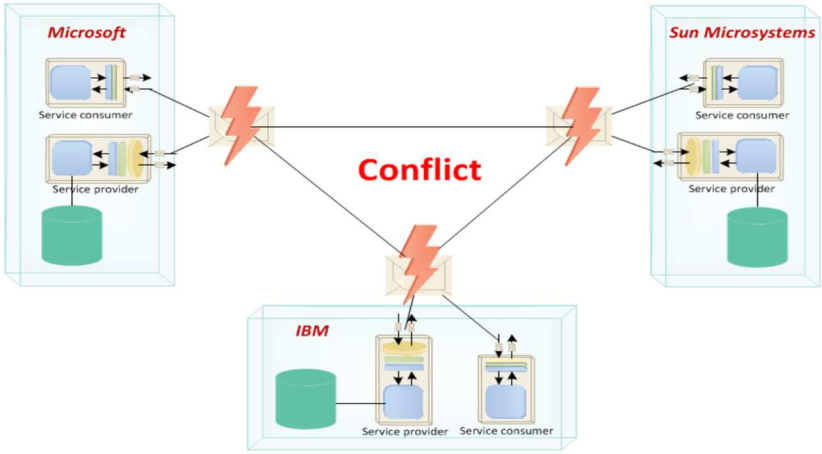


Рис.6. Схема взаимодействия систем с возникновением конфликтов

1) **Несовместность типов данных**, которые передаются в контрактных интерфейсах потребителя (вместо целого присылается символьный).

2) **Несоответствие порядка,** количества параметров в операторе передачи данных;

3) **Отличия в архитектуре платформ** систем Интернет для объектов клиента и сервера.

Для обеспечения интероперабельности контрактов в широком диапазоне систем, используются языки WSDL и XSD. При написании кода службы поставляется класс с определенными в WCF атрибутами [Service Contract], [OperationContract], [FaultContract], [MessageContract] и [DataContract] и утилита *svcutil.exe*, которая вызывает конечную точку сервиса для возврата данных после генерации WSDL-документа по атрибутам.

На этапе выполнения вызывается метод, определенный в интерфейсе сервиса, WCF сериализует типы CLR и вызов метода в формат XML и посылает сообщение в сеть для привязки к схеме кодировки, согласованной с WSDL. При этом участвуют четыре конструкции: две со стороны .NET и две со стороны XML. В .NET имеется тип CLR, который определяет структуры данных и функциональные возможности создания объект такого типа. Со стороны XML задается XSD-описание структуры данных, но сообщение осуществляется лишь после того, как будет создан экземпляр XML (XML Instance).

Язык WSDL. Для описания общедоступных веб-ресурсов используется язык WSDL (Web Service Definition Language) на основе XML. Среда программирования Eclipse позволяет автоматически создавать описания на основе классов Java. В языке определены следующие основные типы данных:

– строки (xsd:string);

– целые числа (xsd:int, xsd:long, xsd:short, xsd:integer, xsd:decimal), числа с плавающей-запятой (xsd:float, xsd:double);

– тип логический (xsd:boolean);

– последовательности байт (xsd:base64Binary, xsd:hexBinary);

– дата и время (xsd:time, xsd:date, xsd:g);

– объекты (xsd:anySimpleType).

В качестве переменных для сообщений могут использоваться последовательности, созданные из фиксированного количества переменных простых типов.

Типичный WSDL-файл имеет следующее описание.

```
<wsdl:definitions [.]>
<!-- Декларация типов, которые используются в сервисе —>
<wsdl:types>
```

<element name="someMethod">
<complexType>
<sequence>
<element name="arg0" type="xsd*:double"*/>
<element name="arg1" type="xsd*:boolean"*/>
</sequence>
</complexType>
</element>
<element name="someMethodResponse">
<complexType>
</wsdl: types> …

Приведенное WSDL-описание определяет веб-сервис MyService с единственным методом String someMethod(double arg0, boolean arg1). На его основе можно сгенерировать два типа данных, которые отвечают входным и исходным аргументам метода. Эти типы применяются в описаниях someMethodRequest и someMethodResponse – входного и выходного сообщений для операции someMethod.

Операции декларируются в описании интерфейса сервиса (декларация wsdl:portType) и дальше в описании привязки сервиса к SOAP (декларация wsdl:binding), причем во втором случае также оговаривается способ вызова (<wsdlsoap:body use ="literal"/>). За счет этого при вызове операции используются те же названия параметров, что и в методе класса. В конце WSDL-файла находится декларация веб-сервиса (<wsdl:service>), в которой содержится информация относительно его расположения (параметр location).

3.6.4. Технологии и языки Grid - OGSA

Документ Open Grid Services Architecture (OGSA), определяет концептуальную основу системы Grid и ее основных компонентов (рис.8): архитектуру, данные, инфраструктуру, компьютеры и приложения. Система Grid объединяет локальные ресурсы, которые настраиваются и администрируются автономно (нижний слой), объединяются, руководствуются и динамически отслеживаются за счет программного обеспечения промежуточного слоя (middleware).

Прикладное ПО через стандартизованные интерфейсы имеет доступ к локальным и распределенным ресурсам через средний слой, который скрывает от пользователя внутреннюю сложность и физическое отображение ресурсов. В основу стандарта OGF положена архитектура SOA, согласно которой вся функциональность промежуточного слоя реализуется путем комбинации взаимоувязанных Web сервисов (SOAP, WSDL). Служба OGSA (Basic Execution Service) определяет интерфейс к сервисам, которые инициируют вычислительные процессы в Grid, отслеживают и руководят вычислительной деятельностью. Кроме того определены модели жизненного цикла (состояния) процессов, а также информационные модели вычислительного процесса.

В Grid реализован программный интерфейс SAGA (Simple API for Grid Applications). Он определяет программную модель для описания компонентов Grid и сигнатуру методов для поддержки процесса выполнения заданий, управления ресурсами и др. Целью спецификации является предоставление стандартизировавшего доступа к функциям Grid на языках высокого уровня (C, Java, Python и т.п.) и тем самым обеспечение их совместимости с разными middleware.

Одной из Grid систем является ETICS. Она предоставляет е-инфраструктуру (Web Application и Web Services) для тестирования, интеграции и конфигурирования, занесения в репозиторий готовых программ и ПС (рис. 7). ETICS включает процессы сборки (builds) и управления тестами на компьютерах пользователей, обеспечивают отправку удаленных сборок на разные платформы пользователей. Программные объекты (пакеты, компоненты и системы) разрабатываются по модели CIM (Common Information Model).

Сценарии работы пакетов и систем задаются на языке моделирования UML. Сервисный набор процедур Etics обеспечивает построение и тестирование новых пакетов и систем с помощью механизмов плагинов. Каждый плагин включает в себя публичный интерфейс и описание услуг.

В набор характеристик системы входят механизмы спецификации зависимостей между разными пакетами и их тестами. Функциональные плагины обеспечивают проверку договоров, тестов для выполнения разных элементов систем, генерацию документации и ведения готовых объектов в оперативном или постоянном репозитории.

Рис. 7. Портал ETICS

ETICS обеспечивает разработку наборов пакетов из комбинаций перекомпилированных двоичных элементов, размещенных в репозитории. Компоненты регистрируются в репозитории в стандартном языке WSDL консорциума W3C (рис.8). Стандартное описание компонентов используется при проведении сборки и тестирования (Build and Test Web Application) групп компонентов. При конфигурации совокупности компонентов в результирующий файл поступают команды контроля версий (Command-Line Client), тестирования свойств и зависимостей в выходном коде системы. Этот сервис можно задать командной строкой задания сборки и тестирования (Build and Test Service). *Удаленная сборка/тестирование* выполняется на разных платформах с генерацией ряда отчетов о сборке, статическом и динамическом тестировании отдельных компонентов в разных средах глобальных пользователей.

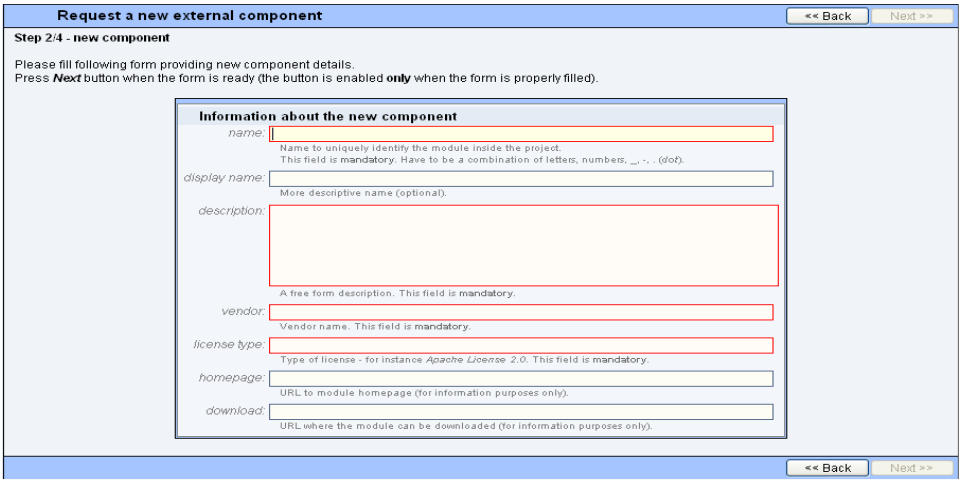


Рис. 8. Стандарт описания программ в Etics, Grid

Каждый новый компонент содержит сведения (имя, лицензия, URL репозитория), глобальный уникальный идентификатор – ID (GUID); информацию о версиях, GUID (свойства, среде выполнения и зависимости), тестовых команд и GUIDs.

Типы данных пакетов и систем объединяется плагинами, в которых задаются услуги для потребителей или поставщиков, доступ к OCAM, архитектуре CPU, компиляторам с языков программирования и средствам спецификации зависимостей между разными пакетами, используемым при сборке программ и их развертывании. Плагины обеспечивают проверку контрактов, тестов выполнения разных элементов систем, генерацию документации, ведения готовых КПИ в репозиториях.

Главная проблема в ETICS – преобразование некоторых компонентов систем для альтернативной платформы гетерогенной среды компьютеров с 16-, 32-разрядной платформы на 64-разрядную платформу среды Grid.

Заключение по SOA SCA и системным сервисам

В работе рассмотрены сервисы, которые являются системной поддержкой процессов разработки программных систем и веб-сервисы Интернет как информационные ресурсы, используемые для решения научных и бизнес-задач в среде Интернет. Дано описание набора системных сервисов CORBA для управления объектами при обмене данными (через stub и skeleton) между клиентом и сервером. Представлены модели сервисов SOA и SCA, используемые для композирования систем из сервисов и последующего их выполнения в распределенной среде. Дано описание системы Grid, в состав которого входит ряд систем поддержки физического эксперимента и реализации прикладных систем в e-sciences (физики, биологии, математики, медицины и др.). В ней реализованы операций сборки (building), тестирования (testing) и конфигурации (configurating) и работа с данными, согласно универсальной модели данных. Приведено описание ряда стандартов W3C, включая протоколы и языки описания сервисов для функционирования в среде Интернет (XML, SOAP, BPMN, WSDL, BPEL и др.). Приведено описание протоколов (Icontract) в системе WCF для передачи сообщений между распределенными системами и обеспечения взаимодействия между разнородными системами в среде Интернет.

Литература к подразделу 3.6

1. Андон Ф.И., Лаврищева Е.М. Методы инженерии компьютерных распределенных приложений.-К: Наук.думка.- 1997. 271 с.

2. Лаврищева Е. М. Software Engineering компьютерных систем. Парадигмы, Технологии, CASE-средства программирования. – К.: Наук. Думка, 2014 – 284 с. / Программная инженерия компьютерных сстемю www.urait.ru

3. Лаврищева Е. М., Грищенко В.. Сборочное программирование. Основы индустрии программных систем.- Наук.Думка, К.: 2009.-370с.

4. Карпов Л. Е. Архитектура распределенных систем программного обеспечения – М., МАКС Пресс, 2007. – 130 с.

5. Карпов Л. Е., Юдин В. Н. Обмен данными в распределённой системе поддержки решений. Труды Института системного программирования, т. 19, М., Институт системного программирования РАН, 2010, стр. 71-80, ISBN 978-0-543-57630-9, ISBN 978-5-4221-0085-9, ISSN 2220-6426 (Online), ISSN 2079-8156 (Print), http://www.ispras.ru/ru/proceedings/docs/2010/19/isp_19_2010_71.pdf

6. Jon Siegel. "Quick CORBA™ 3". Wiley Computer Publishing, John Wiley & Sons, Inc., 2001 (Джон Сигел, "CORBA 3", М., МАЛИП, 2002).

7. Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju. Web Services. Concepts, Architectures and Applications. Springer-Verlag, 2004.

8. А.П. Демичев, В.А. Ильин, А.П. Крюков, Введение в Грид-технологии. Препринт НИИЯФ МГУ - 2007 - 11/832 2007, http://www.sinp.msu.ru

9. А.М. Ходжибаев, Ф.Т.Адылова, Новейшие информационные ГРИД-технологии в электронной медицине, Донецк, журнал телемедицинита медичної телематики том 3, №1, 2005

10. Lim Gray. A transformed scientific method. http://research.microsoft.com/en-us/collaboration/fourthparadigm/4th_paradigm_book_jim_gray_transcript.pdf

11. Дрёмин И.М. Физика на большом адронном коллайдере. УФН. Июнь 2009. т. 179 № 6 Fusion for energy. Annual report 2011. av. At http://fusionforenergy.europa.eu/downloads/mediacorner/publications/ reports/ANNUAL_2011.pdf

12. http://www.w3.org/TR/2008/REC-xml-20081126/

13. Гладцын В. А., Кринкин К. В. Яновский В. В. Сервис-ориентированная архитектура: стандарты, алгоритмы, протоколы – Санкт-Петербург: СПб ГЭТУ ЛЭТИ, 2006 – 108 с.

14. Papazoglou M. P., Dubay J.-J. A Survey of Web Service Technologies, Technical Report DIT-04-058, Ingegneria e Scienza dell'Informazione, University of Trento, 2004.

15. http://www.w3.org/TR/soap/

16. http://www.w3.org/TR/soap12-part1/wsdl20/RDF/wsci

17. http://www.omg.org/spec/BPMN

18. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

19. http://www.oasis-open.org/specs/index.php#wsbpelv2.0

20. http://uddi.org/pubs/uddi-v3.0.2-20041019.htm

21. http://www.oasis-open.org/specs/index.php#uddiv2

22. http://www.oasis-open.org/specs/index.php#uddiv3

23. http://www.ibm.com/developerworks/websphere/techjournal

24. Джим Амсден, Моделирование SOA: часть 1. Идентификация сервисов, IBM® Rational® Application

25. http://127.0.0.1:4000/IContract

26. https://web.infn.it/etics-support/index.php/documentation

3.7. Средства Интернет для использования сервисных ресурсов

Семантическая сеть – эта новая информационная среда глобальной информационной сети Интернет, которая содержит семантические ресурсы, языки и инструменты разработки онтологий, систем и бизнес-процессов с использованием накопленных в среде знаний. Термин «семантическая сеть» (semantic Web) ввел Тим Бернес-Ли (2001) в журнале «Scientific

American», где им были определены семантические основы глобальной сети. С тех пор это понятие развивается исследовательскими программами институтов США и Евросоюза, множеством больших и малых производителей бизнес систем, проектами с открытым кодом, а также конференциями по семантическим ресурсам и технологиям (см., например, http://semwebprogramming.org).

Семантическая сеть предоставляет научный и прикладной сервис для автоматизированной обработки научных задач (биологии, физики, математики, бизнес-задач и др.), а также больших данных в разных форматах, интеграцию данных из коллажей (Mash-Ups), поиск и композицию сетевых служб, управление интеллектуальными агентами в мобильных приложениях и др. Эта сеть используется для решения задач с использованием многочисленных сервисов и научных достижений в области предоставления бизнес-услуг, предлагает новые подходы к использованию больших объемов информации, основанных на новых словарях и концептуальных моделях, методах онтологизации новых научных задач и приложений с помощью предметно-ориентированных языков (OWL, DSL и др.) и инструментов (FODA, Protégé, DSLTool MS.Net и др.).

Далее описываются языки и инструменты инструмента онтологии в глобальной информационной сети для представления научного домена «Вычислительная геометрия» и инженерного домена «Жизненный цикл разработки программных систем» в глобальной семантической среде.

Базовые ресурсы семантической сети

Глобальная семантическая сеть включает в себя множество ресурсов и инструментов (рис.1), основные из которых приведены далее.

Языки семантической сети включают набор ключевых слов, которые позволяют описывать компоненты и утверждения. Применяются такие языки для описания онтологий, бизнес процессов и данных (OWL, RDF, BPMN, DSL и др.), а также для задания уникальных идентификаторов ресурсов – Uniform Resource Identifier (URI) и др. Уникальные имена элементов семантической сети образуют пространство имен. Доступ к различным ресурсам осуществляется с помощью уникальных имен ресурсов URN (Uniform Resource Name).

Онтология определяет понятия, отношения и ограничения объектов в концептуальной модели предметной области. Многие онтологии могут быть включены в любое приложение путем адаптации к специфическим потребностям этой области. Онтология поддерживает коммуникации между приложениями и является способом описания доменов таких прикладных областей, как финансы, медицина, экономика, бизнес приложения и др.

Инструменты могут быть четырех типов: конструирования и развития приложений семантической сети, справочные инструменты для изучения сети, инструменты-резонеры, добавляющие механизмы вывода для семантической сети, а также машины правил для расширения семантической сети.

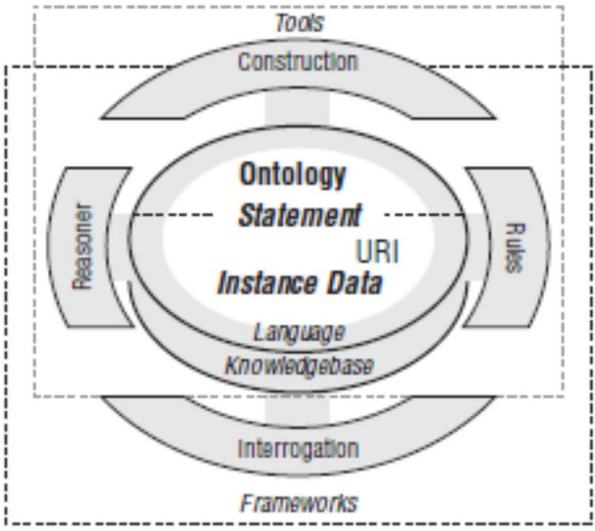


Рис. 1. Основные компоненты глобальной Семантической сети.

Инструменты семантической сети позволяют конструировать и интегрировать средствами онтологии Жизненный Цикл из отдельных экземпляров ресурсов. С помощью графического инструмента (GUI) осуществляется просмотр и исследование данных Семантик Веб, являясь полезным редактором сети.

Справочные инструменты обеспечивают навигацию по семантической сети в поисках ответа на вопрос. Существуют различные справочные методы, начиная от простой навигации по графу при поиске вплоть до полного применения языка запросов. К ним относятся:

Механизмы-резонеры, которые добавляют к семантической сети новые понятия, необходимые пользователям. Компоненты создают логические дополнения с помощью классов. Классификация заполняет структуру класса, позволяя должным образом соотносить понятия и отношения с другими классами. Имеется несколько типов резонеров, предлагающих различные уровни рассуждений. Резонеры включаются в другие инструменты и каркасы. Они служат рычагами для создания логически правильных вспомогательных утверждений.

Машины вывода основаны на правилах дескрипторной логики, позволяющей добавлять средства измерения конструкций знаний с помощью правил слияния онтологий и других логических задач, в том числе поиск с подсчетом и по строке (count and string searches). Машины на правилах могут рассматриваться как часть общего представления знаний. заданных языком описания правил.

Семантический каркас объединяет перечисленные инструменты и позволяет им работать как единое целое. Утверждения, универсальные идентификаторы, языки, онтологии, и данные конкретных экземпляров классов составляют семантически связанную информацию в семантической сети, которой манипулирует семантический каркас, создавая новые инструменты.

Сетевые данные отражают смысловое значение типов данных и сборку данных для совместного использования путем доступа и обмена богатыми информационными ресурсами глобальной сети, включая использование многих существующих источников данных.

Динамические данные сети позволяют в динамике выполнения изменять структуру и содержание обрабатываемой информации.

Фреймворки Семантической Сети включают наборы программных средств среды, Библиотек ресурсов для создания, манипулирования и обогащения семантической сети. Они помогают создавать выражения, онтологии, сетевые приложения и публиковать их в глобальной сети. Примерами таких средств могут служить среда программирования Eclipse, язык RDF и Java, набор библиотек для работы с Apache Jena и средства создания онтологией Protégé систем представления знаний.

Язык RDF был утвержден в качестве стандарта консорциумом W3C в 2004 году. Он предназначен для систематического описания сетевых ресурсов, понятных компьютеру. Формат RDF предназначен для сохранения метаданных (данные о данных), описания семантических ресурсов, каркасом для создания отдельных компонентов глобальной семантической сети. Документы в RDF обрабатываются компьютером автоматически. Схема RDF, обозначаемая часто как RDFS (от английского *RDF Schema*) является надстройкой над RDF, которая позволяет создавать классы и свойства объектов.

Язык OWL (Web Ontology Language) используется с 2004 года, он построен на форматах RDF и RDFS и предназначен для обработки информации в сети. Язык OWL имеет 3 степени детализации, легко масштабируется и согласовывается с современными сетевыми стандартами. В 2008 году был принят новый стандарт OWL 2, включающий описание логики.

RDF Query Language - это новый язык запросов для быстрого доступа к данным RDF. Используется обычный протокол и язык SPARQL, программы могут анализировать RDF-описания ресурсов и получать из сети необходимую информацию.

3.7.1. Онтологии для представления существующих знаний

Онтология является основой стандартизации мировой системы знаний, в том числе языковой, инженерной, системной и другой деятельности. Появились международные профильные стандарты терминов и определений, а также ряд международных органов, которые ответственны за их ведение (ISO, W3C и некоторые другие).

В основе представления любой системы знаний лежит понятийная база, совокупность концептов (понятий) и отношений между ними, классификация понятий и их таксономия в виде тезаурусов, а также методы создания профильных онтологий.

В мировой практике можно наблюдать целый ряд профессиональных онтологий:

- онтология Sensus базируется на понятиях естественного (английского) языка и содержит более чем 70 000 терминов и их дефиниций;
- онтология понятий электронной коммерции;
- глобальная онтология продуктов и услуг (стандарт Объединенных Наций);
- коммерческая онтология SCTG транспортных потоков товаров компаний;
- онтология e-cl@ss поддерживает обмен данными и материалами между продавцами и пользователями крупных компаний Германии;
- онтология товаров RosettaNet поддерживается более, чем 400 коммерческими компаниями.

Существует и широко используется круг медицинских онтологий (Galen для определения клинической картины; UMLS для Национальной медицинской библиотеки Соединенных Штатов; ON9 для аттестации известных медицинских систем по определенным параметрам. Одновременно существуют также инженерные онтологии, онтологии деятельности предприятий, химические и биологические онтологии и многие другие.

К основным направлениям создания онтологий относятся:

- разработка онтологических моделей, методов и средств доступа к мировым информационным ресурсам;
- создание моделей описания готовых ресурсов разработки как повторно используемых знаний;
- создание библиотек онтологий по профилям знаний;

- разработка методов и средств создания комплекса инструментов ведения онтологий;
- разработка методических и учебных материалов по применению семантической сети для описания профессиональных знаний.

Онтология в Семантической глобальной сети содержит аппарат построения концептуальной модели предметной области, которая включает понятия, отношения и ограничения ее элементов. Средствами уже перечисленных языков семантической сети формализуется любая предметная область из накопленных знаний. Концептуальная модель включает структуры данных, содержащие релевантные классы объектов, их связи и правила (теоремы, ограничения), принятые в этой области (рис.2).

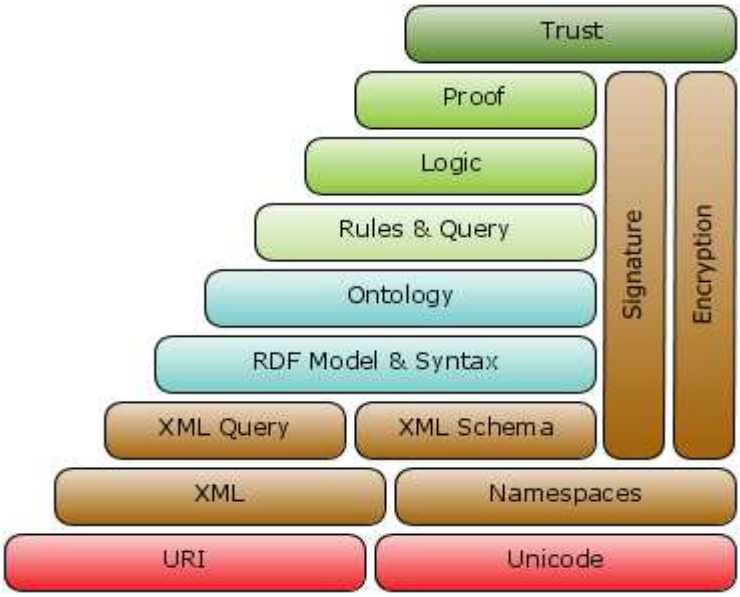


Рис.2. Набор понятий семантической сети.

При описании онтологии некоторой предметной области используются индивиды, понятия, объекты, атрибуты и отношения.

Индивиды – это основные, низко уровневые компоненты онтологии. Индивиды могут представлять собой как физические объекты (люди, дома, планеты), так и объекты абстрактные (числа, слова).

Понятия – абстрактные группы, коллекции или наборы объектов, которые включают в себя индивиды, классы или же их сочетания.

Объекты могут иметь атрибуты. Каждый атрибут задает имя и значение, которые используются для сохранения информации, специфической для каждого объекта.

Описание модели предметной области выполняется с помощью языков OWL, RDF, RDFS и KIF (Knowledge Interchange Format), а также логических языков. Для работы с языками онтологий применяются самые разные программные инструменты: редакторы онтологий (для создания онтологий), системы управления Базами Данных онтологий (для хранения и обращения к онтологиям) и хранилища онтологий (для работы с несколькими онтологиями).

Описание онтологии на языке OWL – это последовательность аксиом и фактов, информация о классах, свойствах, ресурсах и документах сети, которые импортируются через URI и могут ссылаться на XML-схему по форме: <Datatype ID>:: = <URI ссылки>

3.7.2. Элементы онтологии для задания доменов и систем

К языкам описания онтологии доменов относится языки DSL, FODA (Feature-Oriented Domain Analysis), DSSA (Domain-Specific Software Architectures) и инструменты реализации DSL – Eclipse-DSL, а также ODSD (Ontology-Driven Software Development) для описания классов, отражающих понятия домена. Основной системой представления конкретных онтологии некоторой предметной области является Protégé.

Онтология доменов описывается с помощью метапонятий – **классы, слоты, факты, аксиомы, фасеты** и другие.

Классы описывают понятия предметной области, а **слоты** – свойства (атрибуты) классов. **Фасеты** предназначены для описания свойств слотов (конкретные типы и возможные диапазоны значений). Класс может быть подмножеством или пересечением более общих классов с ограничениями. Абстрактные классы являются контейнерами конкретных классов и могут содержать *абстрактные* атрибуты (которые не содержат конкретных значений). Атрибуты понятий предметной области являются ***слотами,*** образующими классы, которыми могут быть значения (экземпляров атрибутов). Согласно фреймовой модели представления знаний, которая используется в Protégé, слот является фреймом. Слоты определяются независимо от какого бы то ни было класса, и один и тот же слот может принадлежать разным классам.

Аксиома содержит совокупность понятий, которые могут относиться к обобщенным классам и включать ограничения, наборы ресурсов, булевы комбинации описаний и прочее. Каждая аксиома задается с помощью правил и ограничений.

Фасет позволяет вводить ограничение на типы данных и диапазоны значений их экземпляров (значений атрибутов), подобно понятиям XML-схемы. Фасеты задают ограничение на присоединение слота к фрейму класса. Слоты-образцы **(***template slot*) и собственные (*own*) *слоты* можно присоединить к фрейму (класса) одним из двух способов: как *слот-образец* или как собственный слот, который присоединяется к фрейму. Классы также могут иметь собственные слоты. Например, документация класса является собственным слотом, присоединенным к классу, поскольку описывает сам класс, а не экземпляры класса.

Факты – это информация о специфических ресурсах в форме классов и свойств со значением этого ресурса. Аксиомы используются, чтобы сопоставить класс и свойство идентификации с частичными или полными спецификациями характеристик, поставщики могут предлагать другую логическую информацию о классах и свойствах. Каждая аксиома класса содержит совокупность более общих классов и совокупность локальных ограничений свойства.

Элементы онтологий могут вступать в следующие отношения:

– обобщение для сужения существенных признаков понятия с расширением круга понятий объектов и их объема;

– конкретизация, как добавление существенных признаков, благодаря чему содержание понятия расширяется, а объем его сужается;

– агрегация, как объединение ряда понятий в новое понятие, существенные признаки нового понятия могут задаваться суммой признаков;

– ассоциация, как наиболее общее отношение, которое утверждает наличие связи между понятиями, не уточняя зависимости их от содержания и объемов.

3.7.3. Описание домена – Вычислительная геометрия

Вычислительная геометрия (computational geometry) – отрасль компьютерных наук для изучения алгоритмов в терминах геометрии. Основным стимулом развития вычислительной геометрии как дисциплины была компьютерная графика и системы автоматизированного проектирования. Многие задачи вычислительной геометрии являются классическими и могут появляться при математической визуализации.

Другим важным применением вычислительной геометрии является робототехника (задачи распознавания образов, планирование их движениями), геоинформационные системы (геометрический поиск, планирование маршрута), дизайн микросхем, программирование станков с числовым программным управлением и др.

Основными разделами вычислительной геометрии является:

- комбинаторная вычислительная геометрия или алгоритмическая геометрия, которая рассматривает геометрические объекты как дискретные сущности. Основополагающим материалом является книга Препарати и Шеймоса, в которой впервые в 1975 был использован термин "вычислительная геометрия".
- численная вычислительная геометрия или машинная геометрия, геометрическое моделирование, которое имеет дело в основном с представлением объектов реального мира в форме пригодной для дальнейшей компьютерной обработки (1971). Этот раздел можно рассматривать как дальнейшее развитие вычислительной геометрии и часто рассматривается как раздел компьютерной графики.

Комбинаторная вычислительная геометрия содержит разработанные эффективные алгоритмы и структуры данных для решения задач в терминах базовых геометрических объектов: точек, отрезков, многоугольников, многогранников и других.

Вычислительная геометрия предназначена для работы над очень большими наборами данных с десятком или сотен миллионов точек. Основные задачи вычислительной геометрии можно классифицировать разными способами, и с различными критериями.

Пример задачи. На множестве *n* точек на плоскости найти пару точек, расстояние между которыми наименьшее. Расстояние между каждой парой точек вычисляется в виде *(n(n-1)/2)*. Затем выбирается пара с наименьшим расстоянием. Полный перебор имеет сложность $O(n^2)$, то есть время его выполнения пропорционально квадрату количества точек. Имеется также алгоритм со сложностью $O(n \cdot log(n))$.

В вычислительной геометрии имеются статические задачи и задачи геометрического поиска.

Статические задачи включают фундаментальные задачи такого рода:

- *выпуклая оболочка*: имея набор точек необходимо найти наименьший выпуклый многоугольник, который содержит все точки;
- *пересечение отрезков*: найти все пересечения в наборе отрезков;
- *триангуляция Делона*;
- *диаграмма Вороного*: имеется множество точек на плоскости, требуется разделить это множество на области, чтобы каждая такая область образовывала множество точек, более близких к одному из элементов исходного множества, чем к любому другому его элементу;
- *задача ближайшей пары точек*: имея набор точек найти кратчайшее расстояние;
- *евклидов кратчайший путь*: соединить две точки Евклидова пространства (с полигональными препятствиями) кратчайшим образом.
- *триангуляция многоугольника*: имея многоугольник, разбить его на внутренние треугольники.

Задачи геометрического поиска. В задачах геометрического поиска входные данные состоят из двух частей: пространств поиска и запросов, которые разнятся в различных видах задач. Для обеспечения эффективного выполнения нескольких запросов пространство поиска требует предварительной обработки, включая:

- *региональный поиск*: обработать набор точек, с целью эффективного поиска набора точек, содержащихся в заданном регионе.
- *локализация точки*: имея разбиение пространства на регионы, создать структуру данных, которая позволит эффективно определить, в каком регионе находится данная точка.
- *поиск ближайшего соседа*: обработать набор точек чтобы иметь возможность эффективно найти точки, близкие к заданной.

- *трассировка лучей*: для заданного набора объектов в пространстве создать структуру данных, которая позволит эффективно определять объекты, пересекающие заданный луч.

Динамические задачи – это тип задач, входные данные которых постепенно меняются (добавляются или удаляются). Алгоритмы решения таких задач включают в себя поддержку динамических структур данных. Любую задачу вычислительной геометрии можно решать динамически, но за счет дополнительных вычислительных ресурсов. Региональный поиск, построение выпуклой оболочки можно проводить над множеством точек, которые меняются.

Вычислительная сложность для этого класса задач задается параметрами, к которым относятся:

– ресурсы, необходимые для определения структур данных при поиске;

– ресурсы, необходимыми для модификации построенной структуры и задания ресурсов для ответа на запросы.

Некоторые задачи могут рассматриваться как такие, которые принадлежат нескольким категориям в зависимости от контекста.

Вычислительная сложность этого класса задач задается такими параметрами

– ресурсы, необходимые для определения структуры данных при поиске;

– ресурсы, необходимые для модификации построенной структуры и задания ресурсов для получения ответа на запросы.

К динамическим методам вычислительной геометрии относятся методы регионального поиска, основанные на методах локализации точек на простых и выпуклых многоугольниках, а именно:

– **метод Грехема и Джарвиса** для обхода точек в простых и выпуклых многоугольниках, пересеченными прямыми линиями для решения задач с использованием арифметических операций и уравнений;

– **метод быстрой сортировки** на множестве точек и линий пересечения выпуклых и невыпуклых фигур с системой координат;

– **метод М.Шеймоса** по структурной организации данных для выпуклых многоугольников на множестве точек, а также использования описания данных по методу Препарата, в соответствующих точках внутри и вне выпуклых оболочек. Произошел переход к новым графовым структурам Штейнера для решения разных задач триангуляции и поиска наиближайшего соседа и др.

– **метод локализации точек в простых и выпуклых многоугольниках. Он показан на** примере разбития плоскости многоугольника прямолинейными отрезками. Можно сформулировать задачу определения принадлежности точки z к простым или выпуклым многоугольникам P. Многоугольник P содержит точку z. Надо определить, находится ли точка z внутри P. Алгоритм решения этого запроса через z проведена горизонталь l (рис. 3).

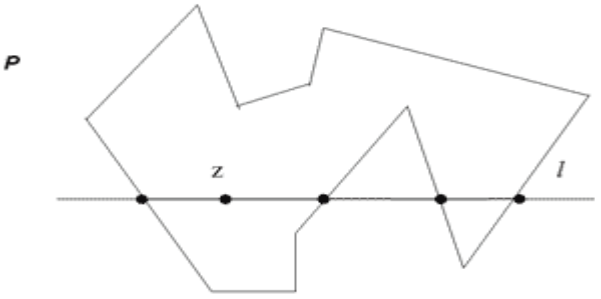


Рис. 3. Многоугольник P

По теореме Жордана внешняя и внутренняя области P хорошо определены: если l не пересекает P, то z внешняя точка. Пусть L пересекает многоугольник P:

а) случай, когда l не проходит ни через одну из вершин P. Пусть L-число точек пересечения l с границей P слева от z. Точка z лежит внутри тогда и только тогда, когда L нечетное.

б) вырожденный случай, когда l проходит через вершины P. Фрагмент программы будет выглядеть так:

begin L:=0;
for i:=1 *to* N *do if* ребро (i) - не горизонтально
then if (ребро (i) пересекает l нижним концом слева от z
then L:=L+1;
if (L непарное) *then z* в середине *else z* внешнее
end

Этот многоугольник P содержит по крайней мере одну точку q такую, что [q, pi] лежит полностью внутри этого многоугольника для любой вершины p_i из P, i=1,...,N. Множество искомых центров внутри P является ядром ЗМ.

Вариации. Во многих программах эта задача рассматривается как задача первого класса. Тем не менее, во многих случаях нужно определить положение курсора «мыши» внутри данного многоугольника. Курсор постоянно перемещается, а многоугольник не меняется. Аналогично можно проверять изображенный на экране радара подозрительный летательный аппарат, не пересек ли он охраняемую границу страны. Такие задачи можно считать задачами геометрического запроса. В CAD-системах сам многоугольник может варьироваться, поэтому задача может считаться динамической. Таким образом в описание онтологической схемы комбинаторной вычислительной геометрии входят (рис.3):

1. Классы задач вычислительной геометрии
1.1. Статистические задачи.
1.2. Задачи геометрического поиска.
1.3. Динамические задачи.
1.4. Вариации.

Реализация онтологии «Вычислительная геометрия». Онтология домена «Вычислительная геометрия» для заданного описания выполнена в системе Protégé 4.1 b представлена на (рис.4).

Protégé является открытым редактором онтологий и фреймворком для построения баз знаний предметной области. Ее платформа поддерживает два основных способа моделирования онтологий с помощью редактора Protégé-Frames и Protégé-OWL. Онтологии, построенные в Protégé, могут быть экспортированы в другие форматы, включая RDF (RDF Schema), OWL и XML.

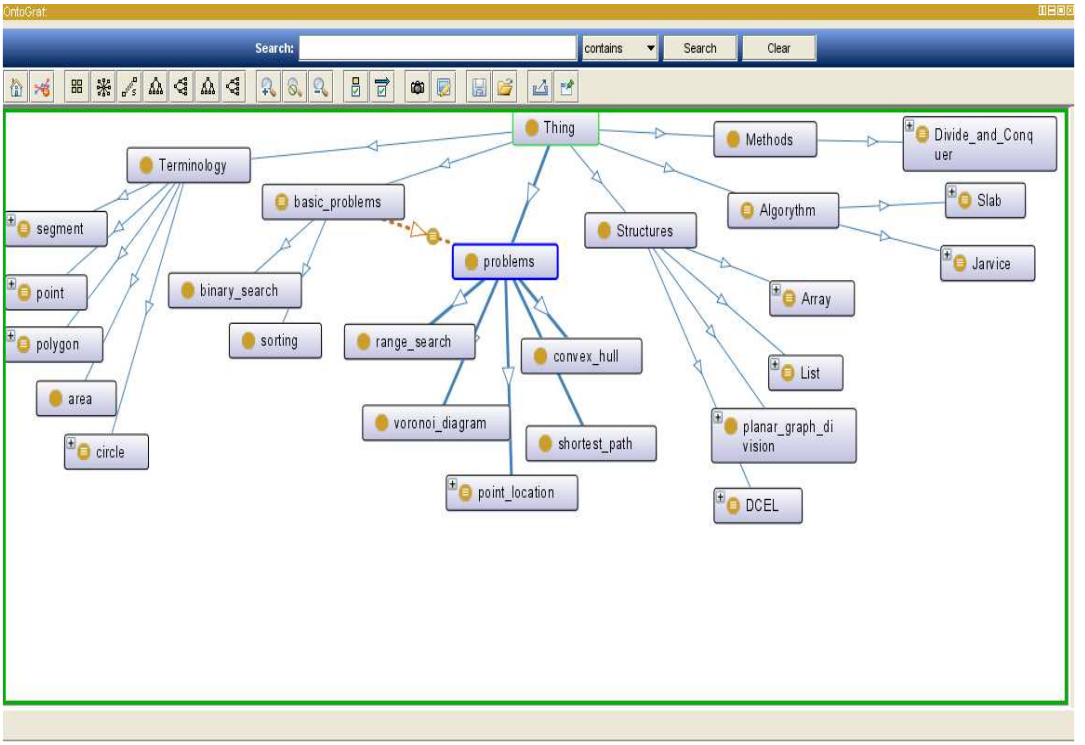


Рис.4. Онтологическая структура вычислительной геометрии.

Платформа Protégé поддерживается значительным сообществом, состоящим из разработчиков и ученых, правительственных и корпоративных пользователей, которые используют его для решения задач, связанных со знаниями, в таких разнообразных областях, как биомедицина для сбора знаний и корпоративного моделирования. Онтология домена «Вычислительная геометрия» включает набор понятий и связей между ними, которые зафиксированы в схеме онтологии (рис.2) и в XML. Реализован экспериментальный вариант онтологии для вычислительной геометрии и имеется е-учебник для обучения этой дисциплины. В подготовленном электронном учебнике описаны алгоритмы геометрии. В работе принимал участие нс. Подлесный И.В., который реализовал геометрию в МФТИ.

3.7.4. Разработка онтологии стандарта жизненного цикла 12207

При проектировании разного рода прикладных систем используются модели жизненного цикла или стандарт жизненного цикла программного обеспечения ISO/IEC 12207. Стандарт ЖЦ является общим механизмом регламентированного построения разных систем. Мои коллективом сотрудников проведена техническая работа по созданию онтологии ЖЦ для разных предметных областей.

Жизненный цикл в данном стандарте представлен процессами (рис.5, 6):

1) основные процессы;
2) дополнительные процессы, включающие процессы поддержки и организационные процессы.

Для каждого из процессов определены виды деятельности (действия – activity), задачи, совокупность результатов (выходов) деятельности для решения задач и др. В стандарте приведен перечень работ для этих процессов, но не задан способ их выполнения и форма представления результатов. Основные процессы – это разработка, эксплуатация и сопровождение программных систем.

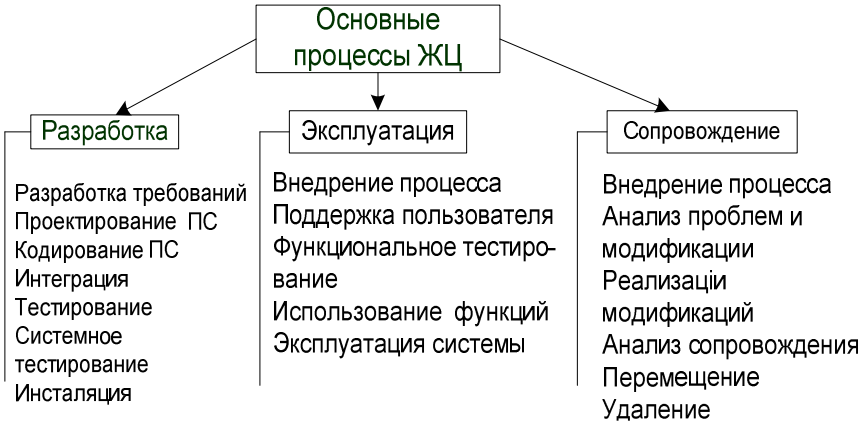


Рис. 5. Схема основных процессов ЖЦ ПС для онтологии

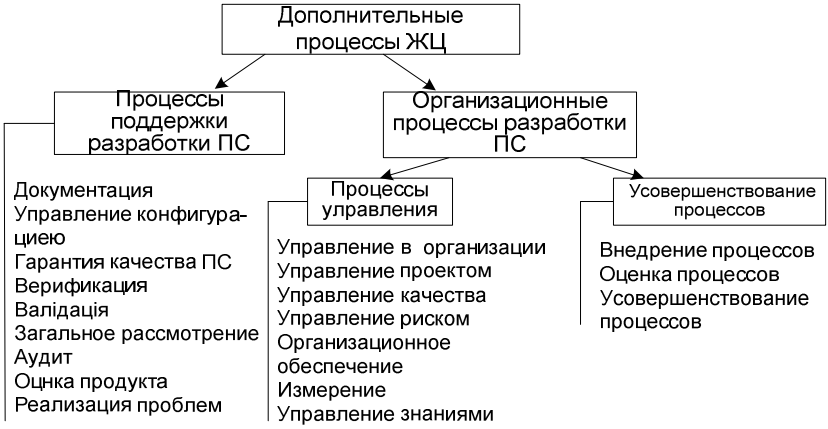


Рис. 6. Схема вспомогательных процессов ЖЦ ПС

Стандарт жизненного цикла содержит в себе ие вспомогательные процессы, которые регламентируют дополнительные действия при проверке продукта, управлении проектом и качеством. Как правило, в зависимости от целей конкретного проекта главный разработчик и менеджер проекта выбирают процессы, действия и задачи, выстраивая определенную модель жизненного цикла для этого проекта. Данный стандарт (в варианте 2007 года) включает в себя (таблица 1) 17 процессов, 74 подпроцесса и 232 технологические операционные задачи (действия). Их необходимо и достаточно для проектирования систем с помощью процессного подхода. Некоторые системные фирмы производители программных пакетов реализовывали отдельные фрагменты или варианты этого стандарта.

Таблица 1. Процессы, подпроцессы и задачи жизненного цикла

Классы	**Процесс**	**Действие**	**Задача**
Основные процессы	5	35	135
Процессы поддержки	8	25	70
Организационные процессы	4	14	27
Всего	17	74	232

Концепция автоматизации стандарта ЖЦ средствами онтологии является новой. Базовыми понятиями онтологии являются процессы жизненного цикла, их взаимосвязи по передаваемым результатам проектирования программных систем, а также функции

процессов для их реализации и выполнения. Если все 17 процессов будут реализованы, то можно из них генерировать некоторые подмножества жизненного цикла, как вариант рабочего жизненного цикла для конкретного применения. Нами рассмотрена онтология процесса тестирования.

Для проектирования онтологии жизненного цикла могут использоваться языки BPMN и DSL. Жизненный цикл представляется с помощью словарей понятий, концептов и отношений между ними в среде Protégé и DSL Tool VS.Net. Полученное онтологическое описание жизненного цикла трансформируется на язык XML, который при реализации размечает данные домена жизненного цикла и устанавливает связи и обмен данными между процессами.

Домен жизненного цикла занимает центральное место в программной инженерия, основным назначением которого является поддержка методов и средств изготовления сложных программных систем. Изучающие эти методы и средства, а также современные стандарты жизненного цикла ISO/IEC 12207–2007 и ISO/IEC 11404–2007 GDT (General Data Types) приняли участие в разработке экспериментального вариант онтологии жизненного цикла с помощью доступных открытых инструментов – DSL Tools VS.Net и Protégé. Концепция онтологизации жизненного цикла обсуждалась в Киевском Национальном университете (КНУ) на научных семинарах кафедры теории программирования и технологии, кафедры информационных систем и МФТИ в группах студентов, которым ЛЕМ читает лекции по предмету "Программная инженерия" с 1992.

Участники разработки онтологии ЖЦ изучили современные онтологические средства в глобальной семантической сети, а также средства визуального представления процессов ЖЦ – DSL Tools VS.Net, Protégé и другие. На основе этих средств было получено первое описание онтологии ЖЦ в графическом виде и в XML. На конференции “Science and Information -2015” в Лондоне был сделан доклад Лаврищевой Е.М. (см. ниже).

После доклада Комитет IEEE предложил подготовить патент по реализации идеи автоматизации жизненного цикла стандарта ISO/IEC 12207. По приезду в ИСП Директор В.П.Иванников преложил сделать программный сертификат, так как реализация для IEEE стоит очень дорого.

Для представления онтологической структуры жизненного цикла (рис. 4 и 5) используется графический язык DSL, который содержит общие абстракции для отображения

классов объектов предметной области, типов процессов и действий, а также отношений между ними. Описание в этом языке данные преобразуется к языкам HTML, XML, WSDL и другим.

Модель жизненного цикла стандарта ISO/IEC 12207 была представлена группой студентов на МФТИ на языке DSL. Это описание было трансформировано к промежуточной модели с более низким уровнем DSL. Такой подход позволяет интегрировать между собой разные части процессов ЖЦ, написанные на языках DSL.

Описание модели характеристик процессов ЖЦ стандарта 12207

Для проведения доменного анализа стандарта жизненного цикла используется FODA (*Feature-Oriented Domain Analysis*). Отличительной особенностью представления процессов жизненного цикла является *диаграмма* зависимостей между характеристиками. Нотация диаграмм характеристик процессов выполняется на языке FDL (Feature Definition Language), позволяющем описывать:

– атомарные и композиционные характеристики, имена которых определяются и используются в разных процессах;

– необязательные (*optional*) и обязательные характеристики (*mandatory*) выражений, которые относятся к замкнутым конструкциям *all*();

– альтернативные характеристики (*exclusive* – выбор) через *one-of();*

– характеристики по умолчанию (default).

Результат трансляции характеристик в FDL может быть выполнен на XML для обмена информацией (XML Metadata Information Exchange format) и может быть импортирован в систему моделирования UML при генерации классов. Подход к описанию модели характеристик предметной области использован при разработке вариантов жизненного цикла программной системы и конфигурировании разных процессов. С помощью данной модели генерируются необходимые варианты жизненного цикла для реализации различных классов программных систем.

Описание процессов жизненного цикла средствами DSL, Protege

Для описания онтологии домена жизненного цикла взят Eclipse DSL. В этом языке имеются средства разработки графических моделей жизненного цикла. Описание основных процессов домена жизненного цикла выполнено с помощью инструментария DSL Tools VS. На рис. 4 приведена схема в языке DSL основных процессов.

Типы отношений в данном графическом представлении задают основную логику процессов домена жизненного цикла. В каждом классе заданы методы и поля, необходимые для функционирования.

Процессы поддержки DSL Tools VS включают в себе процессы, которые выполняются после построения системы и поддержки его работоспособности. Их онтологическая структура отражает структуру основных процессов, процессов поддержки и организационные процессы. Пример жизненного цикла для основных процессов приведен на рис.5, поддержки на рис.6 и организационные – рис.7, 8,9.

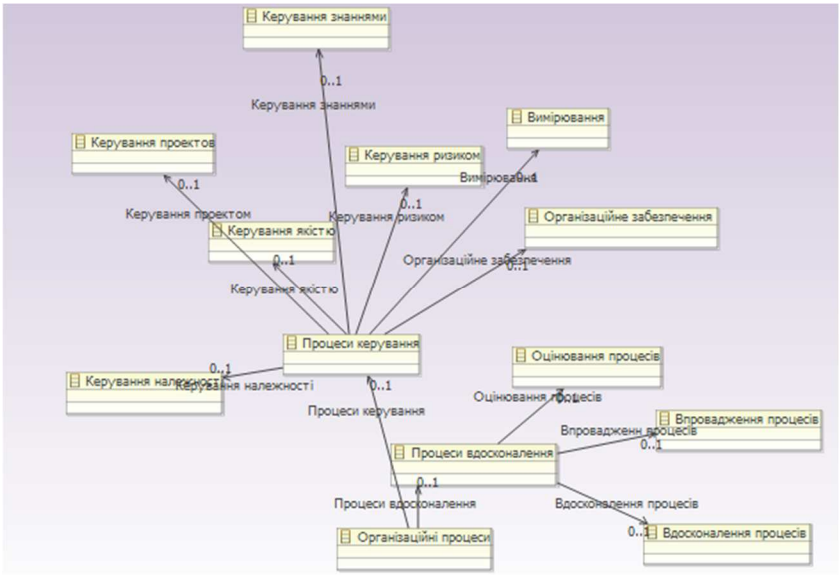


Рис. 7. Онтология основных процессов ЖЦ

Рис.8. Онтология процессов поддержки ЖЦ

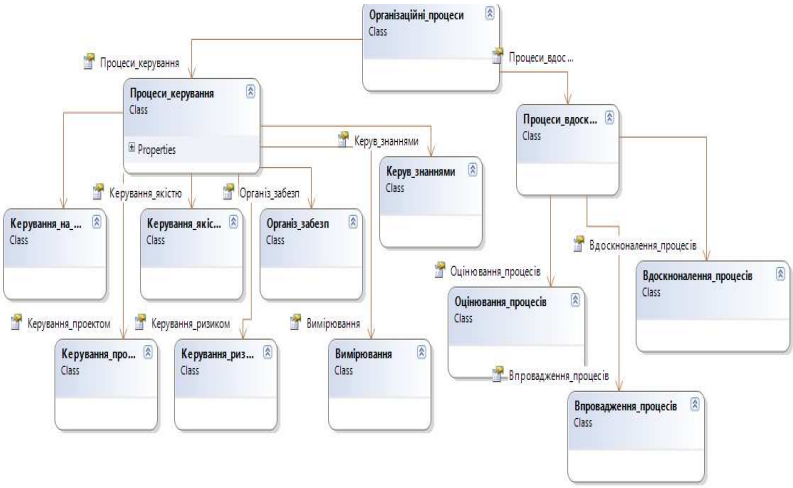


Рис. 9. Онтология организационных процессов ЖЦ

Приведенное описание схем онтологии основных процессов ЖЦ, процессов поддержки и организационных генерируются в текстовое представление языка XML. Текстовое описание процессов жизненного цикла на XML тестируется для нахождения ошибок в графическом описании DSL с помощью соответствующего редактора. Результаты других процессов также выдаются в XML

Далее приведен пример фрагмента описания основных процессов ЖЦ на языке XML:

```
<?xml version="1.0" encoding="utf-8"?>
<AssociationLine Name ="Определение требований"
      Type="Main. Определение требований"
      ManuallyRouted = "true" FixedFromPoint = "true"
</AssociationLine>
<AssociationLine Name = "Интеграция_ПС" Type = "Main.Интеграция_ПС" />
<AssociationLine Name = "Инсталляция" Type = "Main.Инсталляция" />
<AssociationLine Name = "Эксплуатация" Type = "Main.Эксплуатация" />
<Property Name = "Интеграция_ПС" />
<Property Name = "Инсталляция" />
<Property Name = "Анализ требований" />
<Property Name = "Эксплуатация" />…
```

Это описание преобразуется в машинный код при обработке процессов.

Публикации по ЖЦ:

– Lavrischeva Ekaterina. Ontological Approach to the Formal Specification of the Standard Life Cycle, Conference "Science and Information Conference-2015, July 28-30, London, UK, www.conference.thesai.org.- p.965-972.

– Ekaterina M. Lavrischeva. Moscow Physics-Technical Institute, Dolgoprudnuy, Russia. Ontology of Domains. Ontological Description Software Engineering Domain—The Standard Life Cycle", Journal of Software Engineering and Applications, 2015, 8, 324-338

– The Operating Computing Complex «Dnepr-2», Ekaterina Lavrischeva, Institute of the System Programming of Russian Academy of Sciences, ispras.ru.-SoRuCom-2014, IEEE Springer-2015, c.102-104 (Lavrischeva Dnepr-2 -6.12.2014).

В рамках реализации онтологии ЖЦ разработана онтология процесса тестирования средствами Protégé. Задачи этого процесса реализованы студентами на языке Python. На этом реализованном процессе проводилась проверка программ реализации.

Элементы онтологии стандарта жизненного цикла и домена «вычислительная геометрия» представлены на сайте ИТК. Сотрудник ИСП РАН Рыжов А.В. разместил этот сайт на www.ispras.ru//7dragons/ru/ru в разделе «Онтология». На этом сайте была сделана новая онтология с применением Protégé и DSL, средствах Семантик Веб для разработаки сервисов сетевых служб.

Заключение по подразделу 3.7

В разделе 3.7 описаны методы и средства онтологии для проектирования предметных областей в глобальной семантической сети. Приведено описание и реализация конкретных онтологий для научной математической области – «Вычислительная геометрия» и инженерного домена дисциплины «Программная инженерия» – жизненный цикл стандарта ISO/IEC 12207-2012. Соответствующие онтологии описаны в статьях и размещены на сайте http://sestudy.edu-ua.net, http://7dragons.ru/ru.

Эти сайты широко используются при преподавании предмета «Программная инженерия» в КНУ и МФТИ. В первом сайте находится учебник «Програмна інженерія» на укр. языке и рус.

Литература к 3.6 - 3.7

1. http://semwebprogramming.org
2. http://www.omg.org
3. OASIS/ebXML Registry Information Model http://www.oasis–open/org/cojm-metees/specs/ ebrim
4. D. Roman, H. Lausen, U. Keller (eds.): Web Service Modeling Ontology (WSMO: http://www.w3.org/Submission/wsmo
5. J. de Bruijn, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, and D. Fensel: The Web Service Modeling Language (WSML). http://www.w3.org/Submission/wsml
6. www.swebok.org
7. Web Service Semantics – WSDL–S.W3C Member Submission. http://www.w3.org/Sub-mission/WSDL–S/
8. Web Service Glossary//http://www.w3.org/TR/ws-gloss
9. Mens., Van Gorp P, Czarnecki K. A. Taxonomy of Model Transformation.– http://drops.dagstuhl.de/2—5/11
10. D. Roman, H. Lausen, U. Keller (eds.): Web Service Modeling Ontology (WSMO: http://www.w3.org/Submission/wsmo
11. J. de Bruijn, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, and D. Fensel: The Web Service Modeling Language (WSML). http://www.w3.org/Submission/wsml
12. Semantic Annotations for WSDL and XML Schema. W3C Recommendation. http://www.w3.org/TR/sawsdl/
13. Web Service Semantics – WSDL–S.W3C Member Submission. http://www.w3.org/Submission/WSDL–S/
14. Web Service Glossary//http://www.w3.org/TR/ws–gloss.
15. Франко Р., Препарата Ф., Шеймос М. Вычислительная геометрия. Введение. — М. : Мир, 1989. — 478 с. — ISBN 5-03-001041-6.
16. A.R. Forrest, «Computational geometry», Proc. Royal Society London, 321, series 4, 187–195 (1971)

17. Анисимов А.В., Терещенко М.В. Навчально-методичний посібник з курсу "Обчислювальна геометрія та комп'ютерна графіка" факультету Кібернетики КНУ імені Тараса Шевченка.

18. Computational Geometry

19. Computational Geometry Pages/ Geometry In Action

20. "Strategic Directions in Computational Geometry—Working Group Report" (1996), Journal of Computational Geometry

21. Protégé-Frames User's Guide. http://protégé.stanford.edu/doc/frames/

22. Protégé-OWL API Programmer's Guid - http://protege.stanford.edu/doc/owl/guide.html.

23. Standart ISO/IEC 12207-2007 Life Cycle

24. Lavrischeva Ekaterina. Ontological Approach to the Formal Specification of the Standard Life Cycle, "Science and Information Conference-2015", Jule 28-30, London, UK, www.conference.thesai.org.- p.965-972.

25. Лавріщева К.М. Підхід до формального подання онтології життєвого циклу програмних систем Вісник КГУ, серія фіз.-мат.наук. – 2013. – №4. – С. 140–149.

26. Лавріщева К,М. Отологічне подання життєвого циклу ПС для загальної лінії виробництва програмних продуктів.- Теоретичні та прикладні аспекти побудови програмних систем.-TAAPSD-2013.- 25.05-02.06.2013, Ялта.-с.81-90.

27. Лаврищева Е.М. Software Engineering компьютерных систем. Парадигмы, Технологии, CASE –средства. Наук. Думка, 2014. 282 с.

28. Лаврищева Е.М., Карпов Л.В., Томилин А.Н. Системная поддержка бизнес задач в глобальной информационной сети. - Труды конференции «Научный сервис в сети Интернет -2015», 21-26 сентября 2015, Новоросийск, с. 193-218.

Общее заключение по Методам и подходам информатизации в проекте РФФИ 352

1. По методу ОКМ моделирования ПС с обеспечения вариабельности систем (2010-2013) был создан вариант ОКМ метода на компонентной основе и защищен в докторской диссертации Грищенко В.Н.*, а также отдельные аспекты объектно-компонентной теории развита аспирантами в диссертациях и магистерских работах студентов МФТИ и статьях сотрудников отдела:

*Грищенко В.Н. Теоретические и прикладные основы компонентного программирования, доктор. диссертация, 2007, ИК.-36с.;

– канд. диссертации: Задорожная Н.Т., Коротун Т.М., Коваль Г.И. Колесник А., Стеняшин А. и др.;

– Лаврищева Е.М., Колесник А.Л., Стеняшин А.Ю. Объектно-компонентное проектирование программных систем. Теоретические и прикладные вопросы – Весник КГУ, серия физ.-мат. наук, 2013. – №4. – С. 103 – 117;

– Lavrischeva E.M., Kolesnik A.L., Stenyashin A.U. Object-component Development of Application and systems. Theory and Practice» в журнале USA "Software Engineering and Application" (2014). Данная статья получила признание в Украине, в США, в Лондонском журнале "London press" 4 .11.2016. Главный автор статьи Лаврищева Е.М. стала членом «Quarterly Franklin Membership» (ID #5134421UK) и получила право бесплатно публиковать по этой тематике статьи в журнале «London Journal of Research in Computer Science and Technology» (LJCST) и рецензировать статьи по программной инженерии европейских авторов;

– усовершенствован вариант ОКМ под задачи проекта РФФИ 16-01-00352 и опубликован в препринте «Теория объектно-компонентного метода моделирования систем» в ИСП РАН, 2016.- с. 1-48. Препринт переводится на английский язык и будет опубликован в LJCST UK в 2017. ОКМ признан в США, Англии, Украине и России.

2. Использование средств Data Mining для моделирования ОС в отчете РФФИ 2017.

Написаны 5 магистерские диссертации студентами МФТИ (Р. Губайдулина, Б А.Левин, А.Иванов, А.Гариров, Козин В.И. и др. на тему моделирования прикладных систем с использованием метода Data Mining, Reusebility Mining для фрагментов OS Linus в медицине методом сборки (2014-2019)1.

3. Исследована вариантность систем и ОS Linux. Определены сформировавшиеся подходы к организации процесса генерации новых вариантов ОС; методы проведения анализа систем с целью извлечения артефактов с действующих Legacy Systems (OS Linux, Intel Web-systems и др.); сформулирован метод верификации моделей и вариантов ОС, а также сформулирована специфика конфигурационной сборки функциональных артефактов ядра системы Linux. Проведены эксперименты по обеспечению статического анализа и верификации отдельных артефактов, фрагментов ядра этой системы, подготовлены бакалаврская и магистерская работы студентов по данной тематике. Развита теория применительно к задачам ОС и готовятся публикации. Отпечатана статья по проведенным исследованиям задач обеспечения вариабельности и верификации «Верификация и анализ вариабельных операционных систем» в трудах ИСП РАН с участием членов группы проекта (Кулямина В., Мутилина В., Петренко А. и др.). С использованием этих концепций проведена разработка экспериментального варианта ОС Linux для конкретного применения в прикладных областях с участием Козина В.И.

4. Дано описание основ моделирования систем и их семейств с использованием конфигурационной сборки.

К ним относятся модели архитектуры систем и свойства моделей: UML (Unified Modelling Language, 1994), модели: MDA (Model Driving Architecture) OMG, MDD (Model Driving Development) IBM Rational, MDE (Model Driving Engineering) MBE, SOA (Service Oriented Architecture) IBM и др. В результате сформировался общий подход к моделированию систем по моделям или «каркасам» и трансформация объектов к выходному коду. Основу ОКМ моделирования сложных систем составляет логико-математический аппарат спецификации объектной модели (ОМ) и определения функциональных характеристик модели MF (Feature Nodel). Моделирование состоит в построении графовой ОМ, в вершинах которой находятся отдельные объекты системы, а на дугах отношения (связи) между ними. Элементы модели верифицируются с помощью аппарата Model Checking, а затем трансформируются к программному коду. Выходной код тестируется и документируется для заданной платформы. ОКМ относится к теоретическим методам разработки сложных систем по сравнению с инженерными методами производства систем на основе международных стандартов жизненного цикла (ISO/IEC Life Sycle 12207-1996, 2007: ISO/IEC Life Sycle 9000(1-4) и др. Инженерные подходы к разработке массовых систем представлены в монографии Лаврищевой Е.М. «Программная инженерия. Парадигмы, технологии и CASE-средства».-Учебник, 2 издание.-Москва-Юрайт- 2016, www.urait.biblio-online.ru.

Международные стандарты в Software Engineering (SE) используются среди разработчиков деловых и бизнес-проектов. Созданные на инженерной основе Legacy System, требуют большой их доработки в связи с возникновением разного рода ошибок и неточностью определения отдельных функций таких систем.

Начиная с 2009 года, международные комитеты ACM и IEEE предложили развивать теорию и методы в рамках нового направления SEMAT (Software Engineering Theory and Methods). Фактически с этого времени нами применяется математическая теория множеств, логика, алгоритмов и алгебра компонентного проектированию прикладных и компьютерных систем.

Основные математические положения метода ОКМ (логико-алгебро-математический аппарат), теория доказательства и верификации также нашли отражение в этом методе. Отдельные аспекты метода ОКМ опубликованы в ряде статей на укр., рус., англ. языках в соответствующих мировых журналах. ОКМ признан как метод математического

проектирования сложных систем с доказательством правильности реализованных функций системы. В этом и есть его новизна и оригинальность. В дальнейшем в рамках SEMAT реализованы новые теории и методы, особенно для среды Интернет (см. ниже публикации).

Научные работы, опубликованные по проекту РФФИ 352 форма 509.

1. Е.М. Лаврищева. Теория объектно-компонентного моделирования программных систем. Препринт ИСП РАН 29, 2016 г.- с. 1-48. (www.ispras.ru/preprintts/docs/prep_29_2016.pdf)

Аннотация. Представлен формальный аппарат объектно-компонентного моделирования предметной области, включающий логико-математическое описание на четырех уровнях проектирования объектной модели. На каждом уровне моделируются и уточняются объекты, устанавливаются их связи (интерфейсы) и отношения, базовые характеристики функций и логика поведения объектов. Объекты и их интерфейсы группируются в классы и отображаются в графе объектной модели. Объекты переводятся к виду компонентов с сохранением связей. Предложены формальные модели (компонента, интерфейса, среды и системы) компонентной среды, обеспечивается технический перенос объектов в компонентную среду и сборка объектов и компонентов в конфигурационный файл системы для выполнения.

Ключевые слова: моделирование, логико-математический аппарат, предметная область, уровни моделирования, модели, объекты, граф, компоненты, изменяемость, многоразовые компоненты, сборка, конфигурация.

2. В.В. Кулямин, Е.М. Лаврищева, В.С. Мутилин, А.К. Петренко. Верификация и анализ вариабельных операционных систем. Труды Института системного программирования РАН**,** том 28, вып. 3, 2016, стр. 189-208. ISSN 2220-6426 (Online), ISSN 2079-8156 (Print). DOI: 10.15514/ISPRAS-2016-28(3)-12

Аннотация. В данной работе рассматриваются проблемы верификации и анализа сложных операционных систем с учетом их вариабельности, или наличия большого количества разнообразных конфигураций. Исследуются методы, позволяющие преодолеть эти проблемы, проводится их обзор и классификация. Выделены классы методов, использующих для анализа инструменты, не учитывающие вариабельность, и выборки вариантов системы и методов, использующих специализированные инструменты, учитывающие вариабельность. Как наиболее перспективные с точки зрения масштабируемости, выделены техники анализа, использующие выборки вариантов системы, обеспечивающие покрытие ее кода и комбинаций значений конфигурационных параметров, а также специализированные, учитывающие вариабельность кода техники анализа с итеративным уточнением модели поведения системы на основе контрпримеров.

Ключевые слова: анализ, верификация, вариабельность, конфигурационная сборка.

3. Лаврищева Е.М. Петренко А.К. Моделирование семейств программных систем. Труды ИСП РАН, 2016, том 29, вып. 3, 2016, стр. 189-208.ISSN 2220-6426 (Online), ISSN 2079-8156 (Print). DOI: 10.15514/ISPRAS-2016-28(3)-12

Аннотация. Анализируются подходы к моделированию систем, начиная с момента появления языка UML и ряда моделей MDA, MDD, MDE, PIM, PSM, SOA и др. Борьба со сложностью созданных больших прикладных систем привела к определению моделей характеристик MF (Feature Model), отображающих функциональные свойства элементов системы и возможность их изменять, удалять и заменять новыми. Дается конкретизация современных подходов к моделированию сложных систем с помощью приведенных моделей. Рассмотрена сущность объектной теории моделирования изменяемых программных, операционных систем, их семейств, а также метод конфигурационной сборки программных продуктов (ПП) в (Product Lines/Product Families). Определены задачи управления вариабельностью и созданием вариантами систем и их семейств с учетом требований заказчика. Проводится анализ

моделей операционных (на примере Linux) и моделей программных систем (на примере генеративной GDM модели). Определены задачи верификации, трансформации и Minning элементов, а также цели проекта РФФИ по моделированию операционных и Веб-систем.

Ключевые слова: моделирование; проверка моделей; уточнение моделей; вариабельность; верификация; тестирование; семейство систем; управление моделями; конфигурационная сборка, управление вариабельностью.

4. Ekaterina M. Lavrischeva. Assemblling Paradigms of Programming in Software Engineering.- 2016, 9, 2016.- p.296-317, http://www.scrip.org/journal/jsea, http://dx.doi.org/10.4236/jsea.2016.96021 Received 20 April 2016; accepted 24 June 2016; published 27 June 2016 Journal of Software Engineering and Applications, 2016, 9, 296-317

Abstract Assembling paradigms programming are based on the reuses in any programming language (PL) with the passport data of their settings in WSDL. The method of assembling is formal and secures co-operation of the different reuses (module, object, component, service and so on) being developed. A formal means of these paradigms creation with help of interfaces is presented. Interface IDL (Stub, Skeleton) is containing data and operations for transmission data to other standard elements linked and describes in the standard language IDL. Assembling will be realized by integration of reuses elements in these paradigms on the instrumental-technological complex (ITC).

Keywords: Paradigm, Assembling, Theory, Interface, Reuses, Object, Component, Service, Program Systems, Interoperability, Multilanguages, Reengineering

5. Лаврищева Е.М. Парадигмы программирования сборочного типа. Труды Конференции УКРПрог-2014, № 2-3. – с. 121-134.

Аннотация. Представлен формальный аппарат парадигм программирования сборочного типа – модульного, объектного, компонентного и сервисного программирования. Теоретический аппарат каждой парадигмы обеспечивает формальную разработку отдельных программных элементов этих парадигм, определение интерфейсных элементов в стандартном виде и их сборку на единой технологической основе в среде комплекса ИТК сайта http://sestudy.edu-ua.net.

Ключевые слова: парадигма, сборка, теория, интерфейс, повторное использование, объект, компонент, сервис, программа, система, взаимодействие, мультиязыки, реинжиниринг.

6. Лаврищева Е.М. Программная инженерия. Парадигмы, технологии и CASE-средства программирования. Учебник, 2 издание. Москва, Юрайт, 2016, biblio-online.ru

Участие в 2016 году в научных мероприятиях по тематике Проекта

1. Доклад на XVIII Всероссийской научной конференции (19-24 сентября 2016 г., Новороссийск) Научный сервис в сети. Е.М. Лаврищева, Л. Е. Карпов, А. Н. Томилин. Семантические ресурсы для разработки онтологии научной и инженерной предметных областей (отпечатан).
2. Доклад на Международной научно-практической конференции «Теория активных систем (ТАС-2016), 16-17 ноября 2016. ИПУ РАН им. В.А.Трапезникова. Лаврищева Е.М. Теория объектно-компонентного моделирования вариабельных программных систем, опубликован в трудах данной конференции.
3. Доклад на научно-практической Открытой конференции ИСП РАН, с 1 по 2 Декабря 2016 года в Главном здании Российской Академии наук, г.Москва. Е.М. Лаврищева, А.К. Петренко (ИСП РАН, Россия) "Моделирование семейств программных систем"
4. Доклад на XI Научно-практической конференции «Современные информационные технологии и ИТ-образование», 25-26 ноября 2016, МГУ им. М.В.Ломоносова. Лаврищева Е.М. Научные основы построения программных и информационных систем. Е-обучение теории и методам. С.50-55.

Аннотация. Представлены научные основы – теория и методы построения

информационных систем (ИС) и программных систем (ПС). Приведен краткий обзор развития теории программирования, начиная с Ляпунова, Ершова, Дийкстры, Бьернера, Буча и др. Показаны пути развития теории в технике и в технологии систем. Представлена сущность научных теорий и методов разработки и доказательства программ и поданы базовые основы теории и технологии систем (1963-1990), программной инженерии продуктов SE (2001-2014) и SEMAT 2009).

Ключевые слова. Теория; методы; научные основы; информационные системы; программные системы; построение; моделирование; верификация; доказательство; интероперабельность.

Практическое использование результатов проекта РФФИ 325 -2016

Научная продукция – теория и методы формального создания корректных вариабельных систем разного назначения будет преподаваться в основных ВУЗах страны (МФТИ, МГУ, МИФИ, ВШЭ и др.) для поднятия теоретического уровня студентов в области Research Computer Science, как это планируется в SEMAT.

Ожидаемые научные результаты в конце 2018 г.

1) Развитие теории формального моделирования вариабельности конфигурируемых программных и операционных систем.

2) Масштабируемый метод описания моделей вариабельности и конфигурационной сборки программных и операционных систем.

3) Масштабируемые методы разработки новых систем и управления конфигурациями программной и операционной системы, позволяющие проверять корректность артефактов, создаваемых и извлекаемых из них вариантов конфигураций новых систем.

4) Метод извлечения вариабельной модели операционной системы на основе исходного кода и конфигурационных артефактов.

Раздел 2. Форма 502. Развернутый научный отчет по проекту 16-01-00352-2017 (№ 35202018). Методы формального моделирования программных, технических и операционных систем

В.В. Кулямин, Е.М. Лаврищева, В.С. Мутилин, Рыжов А.Г.

Цель проекта РФФИ 352 в 2017 г. состояла в разработке теоретических положений по формальному моделированию программных (ПС) и операционных (ОС) систем. Эти положения включают логико-математический аппарат проектирования графовой объектной модели (ОМ), методы верификации моделей, конфигурирования готовых ресурсов в систему и их тестирования для получения качественной системы.

Сущность этих положений состоим в следующем:

– определение с теоретической точки зрения характеристической модели FM (Feature Model) с помощью метода ОКМ, который является развитием начальных теоретических основ отечественных ученых в области программирования программ и систем для ЭВМ;

– разработка формализмов для верификации и управления конфигурационной сборкой программных систем из программных ресурсов (модуль, компонент, объект) и новых ресурсов - сервисов и сервисных компонентов Интернет.

– представить описание процесса тестирования, моделей MF и систем для управления процессом сборки ПС и конфигурирования готовых программных ресурсов (ГОР), ПС и Веб-систем;

– разработать средства представления и обработки данных (в том числе Big Data) с применением механизмов определения фундаментальных типов данных в интерфейсных объектах ПС и их семейств;

– разработать подход к описанию программных ресурсов ПС и Web-system из сервисных ресурсов Интернет на основе языков WSDL, FODA, BPML и др. для их конфигурирования и генерации из них Веб –приложений разного прикладного характера.

Для достижения поставленных целей и задач по проекту в 2017г. было выполнено:

1. Анализ отечественных теоретических основ программ и систем с целью поднятия теоретического уровня подходов к разработке и моделированию новых сложных систем в нашей стране. Анализ начинается с теории программ и технологий, разработанных первыми ученых СССР (А.А. Ляпунова, А.П.Ершова, Ю.И. Янова, В.М.Глушкова и др.) в начальный период появления первых ЭВМ, метода сборки разноязычных модулей, описываемых в широко используемых языках программирования (ALGOL, COBOL, FORTPAN, PL-1 MODULA-2 и др.) вплоть до 1991г., развития теории ООП Г.Буча, появившейся в 1987г. в направлении формирования логико-математического аппарата для графового моделирования систем и вариабельных моделей MF (Feature Model) для обеспечения трансформации созданных и создаваемых программных (ПС) и операционных систем (ОС) и метода взаимодействия разных систем в современных средах.

2. Разработка подходов к верификации моделей создаваемых систем, к конфигурированию используемых готовых объектов, компонентов и интерфейсов - программных ресурсов (ГОР), к тестированию отдельных ресурсов и систем из ГОР и модификации систем и MF (добавление, удаление, создание новых ГОР);

3. Исследование и разработка вариантов операционных систем, OS Linux, путем изучения системных и функциональных модулей с целью выделения из них базовых элементов для построения моделей ОС и MF и генерации по ним вариантов ядра ОС для конкретного применения в экономике, промышленности, медицины и др. Решается задача управления процессом генерации, верификации моделей, тестирования модулей и интерфейсов сгенерированных вариантов ОС.

4. Разработка подхода к преобразованию типов данных компонентов, задаваемых в их интерфейсах, с целью обеспечения взаимодействия отдельных ГОР, систем ПС и семейств систем СПС в современных средах.

5. Определение подхода к решению поставленной в данном проекте задачи создания Веб-систем, Веб-приложений с нового вида ресурсов ГОР – сервисов (SOA) и сервисных компонентов (SCA) Интернет с использованием новых языков W3C.

Далее дается описание задач 1-5 в проекте РФФИ 352-2017г.

1. Анализ теоретических основ формального описания программ и объектов для сложных систем

На начальном этапе создания ЭВМ в СССР сформировались отечественные теории построения программ (А.А.Ляпунова, А.П.Ершова, Е.Л.Ющенко, Э.Х.Тыугу, и др.) и систем АС, АСУ, АСНИ, АСУ ТП (В.М.Глушкова). Системы разрабатывались с использованием готовых модулей, которые накапливались в Фондах алгоритмов и программ (1976-1992), а позднее в библиотеках и репозиториях систем международного сообщества.

Модуль –это программный элемент, который преобразует множество исходных данных X во множество выходных данных Y методом отображения $M : X \rightarrow Y$. Система из модулей - это пара $S = (T, \chi)$, где T – модель системы; χ – характеристическая функция, определяется на множестве вершин X графа модулей G. Две модульные системы $S_1 = (T_1, \chi_1)$ и $S_2 = (T_2, \chi_2)$ тождественны, если $T_1 = T_2$ и $\chi_1 = \chi_2$, а S_1 и S_2 являются *изоморфными*, если T_1 изоморфна T_2 и $\chi_1 = \chi_2$.

Разработка программ и модулей на первых ЭВМ осуществлялась сначала интуитивно, а потом формализовано на языках программирования (Алгол, Кобол, ФОРТРАН и др.), для которых на новых ЭВМ разрабатывались трансляторы. Появилось новое направление к разработке программ – Software Engineering (1968), основанное на трех «китах»: производительность, качество и индустрия. Главное внимание в процессе разработки стало уделяться методам обеспечения качества продуктов (например, стандарт ISO/IEC 9000, 1-4, Guality) и индустриальным методам сначала с помощью моделей жизненного цикла – ЖЦ (водопадной, спиральной, интеграционной и др.). За рубежом сформировались новые формальные методы спецификации программ (VDM, RSL, Z, B и др.) и подходы к доказательству их правильности (Флойд, Хоар, Дейкстра, Грисс и др.). Эти методы использовались и в экспериментальных отечественных разработках (в университетах, в том числе).

После модуля новым элементом программирования стал объект (1987) в ООП Г.Буча и связанные с ним математические понятия - наследование, полиморфизм, инкапсуляция и др. Разработаны были CASE-средства моделирования объектных систем (Rational Rose, UML1, UML2, MDA, MDD, PIM, PSM, SOA и др.).

Следующей важной задачей программирования стала задача обеспечения изменений в действующие Legacy системы (OS IBM, .NET, Intel и др.) и вновь разрабатываемые системы. Эта задача обсуждается на международных конференциях (VAMOS, SPLE, Reusebility и др.).

В результате предложена модель характеристик MF (Feature Model), 2004 и метод создания вариабельных систем из готовых программных ресурсов (модулей, объектов, компонентов и др.), а также из сервисов Интернет в рамках нового направления Семантик-Веб (2008). Первые вариабельные модели систем и продуктов определены в работах K.Czarnetsci, K.Pohl (ProductLine/ProductFamily), GDM, Grid и др. Их основу составляет модель MF и конфигурационная модель (СМ) для сборки базовых артефактов и готовых ресурсов (reuses, assets, services и др.) в новые системы.

Разработан отечественный логико-математический метод ОКМ (2011-2016) для графового моделирования систем, в вершинах которого размещаются готовые ресурсы (ГОР) и *интерфейсы. На основе спроектированного графа и модели MF проводится сборочное конфигурирование элементов графа в конфигурационный файл ПС и семейство систем СПС.

Далее дается последовательное описание теорий программ и систем, как путь к формированию новых перспективных теорий математического моделирования систем из готовых ресурсов.

1.1. Теория программ для первых ЭВМ

Теория программ - это новый раздел математической науки, объектом изучения которого являются математические абстракции схем программ на специальных языках с определенной логической и информационной структурой, ориентированной на исполнение на ЭВМ.

А.П.Ершов рассматривал эту теорию как часть математики. Он считал, что перед вычислительным делом стоит задача, аналогичная той, которая стояла перед математикой, естествознанием в XVIII—XIXвв. и которая привела к созданию математического анализа и целого ряда инженерных наук - техническая физика, строительная механика, электротехника и др. «Нам необходимо построить анализ и исчисление программ и на их основе создавать инженерные программные дисциплины для применения на ЭВМ для самых разных областей науки и техники (управления, экономики, производства программ и др.).

А.А.Ляпунов определил базовые понятия программ:

Схема программы - это конечный ориентированный граф, показывающий структуру программы с применением сигнатур операций и формальных символов.

Интерпретация - это реализация функций и предикатов проверки их значений в рамках задачи конкретной предметной области.

Семантика - это сущность программы, задающая результат ее выполнения во множестве вычислимых функций над некоторым множеством базовых операций.

В.М.Глушков сформулировал сущность алгебраического программирования, положенного в основу языка АНАЛИТИК для семейства машин МИР-1, 2, 3 и определил модель дискретного преобразователя.

1.2. Теория технологии программирования синтез, сборка, доказательство

Технология программирования (ТП) - это совокупность методов, средств и инструментов для описания свойств и функций разных задач предметной области в виде программ с целью получения решения задачи. В нее вошли на тот период следующие понятия:

Программирующая программа (ПП). ПП (1953) для языка операторных схем А.А.Ляпунова была представлена операциями, содержащими команды управления по графу программы операторов программы, которые зафиксированы в специальных таблицах для реализации программы. ПП-1 (1954) сделана в МГУ Э. З. Любимским, А. П. Ершовым под руководством Ляпунова А.А.

Библиотечный метод. В.М.Глушков в статье «Об одном методе автоматизации программирования», ж. Проблемы кибернетики, № 2, 1957) определил это метод для решения системы дифференциальных уравнений для машин (МЭСМ, Урал 1, УМШН, Днепр и др.) с использованием адресного языка (Е. Л. Ющенко, В.С.Королюк).

Библиотеки численных методов (Е. А. Жоголев) и метод интерпретации программ ИС-2 (М. Р. Шура-Бура) на машинах М-20, БЭСМ и др.

1). Трансляторы с языка Алгол (ТА):

ТА1– С. С. Лавров (ЛГУ, 1962);

ТА2 – М. Р. Шура-Бура и Э. З.Любимский (ИПМ, 1963);

ТА-3 (Альфа-система) русской версии язык Алгол-60 (СО АН СССР, 1964);

ТА-4 – Е. Л. Ющенко, Е. М. Лаврищева – для УВК «Днепр-2» (ИК АН УССР).

2). Операционные системы

Отечественная ОС БЭСМ-6 выполняла общие функции управления памятью, задачами и данными. (В.П.Иванников, 1985–1991). В ядро вошли средства взаимодействия с IBM и Эльбрус, набор протоколов обслуживания программ, трансляторы ЯП (Фортран, С, Паскаль и др.) и средства поддержки абстрактных типов данных, кластерного сборочного конвейера для загрузки модулей в ОС и в библиотеку программ трансляторов, а также средства отладки программ на ЯП.

Системы синтеза, композиции и сборки программ

Синтез программ в языке *Утопист* и ЯП на основе семантической модели предметной области и программ разработан под руководством Э. Х. Тыугу в системе ПРИЗ. Метод синтеза программ в PL/1, Fortran, Assembler реализован путем подстановки семантики их реализации для каждого ЯП (1982).

Композиция программ (Редько В.Н.) – это операции объединения функций и данных типа: «данные–функция–имя» и «функции-композиция-дескрипция» на некотором множестве именованных данных, дескрипций и денотатов (значений). Операции композиции – это подкласс стандартных композиций и композиционных функций (1981).

Сборка программ (Лаврищева Е.М.). Сборочный конвейер (по идеи В. М. Глушкова) собирает из модулей в разных ЯП (Алгол, Кобол, Фортран, ПЛ-1, Модула и др.) системы. Связь модулей проводится с помощью посредников интерфейсного типа (позднее stub и skeleton в IDL (Interface Definition language). Сборка выполнялась специальным сборщиком метода сборки в системе АПРОП, переданной в 52 организации СССР (1980-1985).

Доказательство программных систем (А.П. Ершов): синтезирующее, сборочное и конкретизирующее (1985)

Синтезирующее программирование — это процесс получения программы, исходя из условия задачи и метода ее решения. Условие задачи записывается в виде совокупности логических формул, в которые входят символы известных и неизвестных величин, операций и функций. Спецификация программы определяет формулировку теоремы, утверждающую существование решения задачи. Синтез обеспечивает доказательство теоремы существования в некотором конструктивном логическом исчислении (В.Турский, Э.Х.Тыугу).

Сборочное программирование решает задачу многократного и быстрого процесса создания программ из заранее изготовленных «деталей», роль которых играют модули, обладающие определенной структурной и функциональной целостностью и приспособленностью к взаимодействию с другими модулями через интерфейс.

Конкретизирующее программирование – это конкретизация из универсальной программы некоторых программ данного класса, для которого требуется теория определения и интерпретации данных в структуре предметной области. Полнота этой теории базируется на определении метода решения любой задачи из заданного класса.

1.2.1. Фундаментальные и общие типы данных в программных ресурсах

Тип — совокупность элементов, выделенных на всём множестве предметной области (R. Hindley, 1960). Полиморфный тип — представление набора типов как единственного типа (R. Miller, 1960). Математический тип - это:

– множество всех значений, принадлежащих типу.

– предикат, определяющий принадлежность объекта к данному типу.

Тип данных (ТД) используется в описании программ на ЯП (Algol, Пролог, PL/1, Fortran, Pascal и др.). Аксиоматика ТД разработана С.Дейкстрой, Н. Виртом, В. Турским, П. Науром, А.Замулиным, Н. Агафоновым и др. в 1970-х.

Любые данные, с которыми оперируют программы в ЯП, относятся к следующим стандартным ТД.

– FDT – Fundamental Data Type. Фундаментальные типы данных.

– GPD – General Purpose Datetypes. Общие типы данных (ISO/IEC 11404 GDT).

– Big Data – Большие данные.

Система типов данных реализована в .Net. В ней представлены *типы-значения (*value type) *и типы-ссылки (*reference type). *Типы-значения* – это статические типы в CTS, их значения могут занимать память от 8 до 128 байтов. Они копируются при присвоении им значений. *Типы-ссылки* используют указатели на объекты, которые они типизируют, а также механизмы хранения и освобождения памяти. Ссылочные типы включают: объектные типы (object type), интерфейсные типы (interface type), типы-указатели (pointer type) и др.

Реализована теория сборки и преобразования ТД в отечественных работах:

1. Лаврищева Е.М., Грищенко В.Н. Связь разноязыковых модулей в ОС ЕС. М.: Фин. и стат. 1982, 136 с.

2. Лаврищева Е.М., Грищенко В.Н.. Сборочное программирование.-К.: Наук.Думка.-1991.-256с.

3. Липаев В.В., Позин Б.А., Штрик А.А. Технология сборочного программирования.-М.-1992.-280с.

1.3. Теория объектно-компонентного моделирования (ОКМ) изменяемых систем

ОКМ - метод проектирования графа и объектной модели (ОМ) проектируемой системы ПС из объектов и интерфейсов включает:

– математические операции (*объединения* ∪, *пересечения* ∩, *вычитания* ⊕, *симметричного вычитания* и др.). С их помощью проводится сборка объектов в систему и отображение ОМ к компонентной модели (СМ);

– объектную и компонентную алгебры для изменения ГОР и Reuses компонентов (CRU, assets, servises, artifacts и др.);

– операции сборки (конфигурирования) ГОР, CRU и преобразования данных к разным форматам современных сред (VS.Net, IBM, Corba, Eclipse и др.) с помощью примитивных функций GDT ISO/IEC 11404–2007 к FDT ЯП (и обратно);

– линии (типа ProductLine, ProductFamily) для создания отдельных систем и семейств ПС с помощью готовых ГОР, CRU и трансформации передаваемых через интерфейс данных GDT↔FDT (Лаврищева Е.М. Теория объектно-компонентного моделирования программных систем.-www.ispras.ru/preprint/prep_29_2016)**.**

Описание ОКМ объектного подхода к моделированию ПС и СПС представлено в разделе 1 (стр.18-23). Здесь рассматривается компонентный подход ОКМ к моделированию ПС, СПС и ПТС.

Компонентное моделирование ПС, СПС, ПТС.

Компонентная модель (СМ / КМ) имеет вид:

$CM = \langle RC, In, ImC, Fim \rangle$, где

RC – базовые компоненты множества С, полученные после трансформации объектов графовой модели ОМ;

In – множество интерфейсов для reuse в точках вариантности;

ImC – множество реализованных базовых компонентов в среде;

m (·) – функции преобразования входных и выходных параметров интерфейсов во множестве данных ПС, СПС.

Операции над компонентами:

Операция добавления, удаления, замены компонента, которая задается знаком "–" и имеет вид: $CE.NameSpace(C1) - C2 = (CE \lozenge C\ 1) \oplus C\ 2$.

Операция объединения ($\cup$) и др.

Компонентная алгебра – это набор операций для представления переменных объектов и их эволюции: $\sum = \{\varphi 1, \varphi 2, \varphi 3\}$, где

$\varphi 1 = \{CSet, CESet, \Omega 1\}$ – внешняя алгебра,

$\varphi 2 = \{CSet, CESet, \Omega 2\}$ – внутренняя алгебра,

$\varphi 3 = \{Set, CESet, \Omega 3\}$ – алгебра эволюции компонентов.

С помощью операций алгебры проводиться замена, удаление и добавление нового объекта.

Теория взаимодействия программ и систем

Взаимодействие – это взаимосвязь двух и больше объектов или систем через интерфейс.

Модель взаимодействия *Мвз* имеет вид: $Мвз = \{Мпр, Мсис, Мсред\}$, где

$Мпр = \{Com, Int, Pro\}$ – модель программы,

$Мсис = \{FPC, Int, Pro\}$ – модель системы,

$Мсред = \{Envir, Int, Pro\}$ – модель среды,

Int, Pro - совокупность интерфейсов и протоколов, которые передают данные между программами разных сред *Envir* сети.

Модель Мвз по отношению к стандартной модели открытых систем OSI является моделью верхнего уровня, которая реализуется сервисом Eclipse в Интернет.

Разработаны конкретные модели для современных сред. (Островский А.В. Подход к обеспечению взаимодействия сред Java, MS.Net // Проблемы программирования. – 2011. – № 2. – С. 37–44):

Модель Visual Studio $\leftrightarrow$ Eclipse.

Модель CORBA $\leftrightarrow$ Visual Studio $\leftrightarrow$ Eclipse.

Модель JAVA $\leftrightarrow$ Visual Studio $\leftrightarrow$ Eclipse.

Эти модели отработаны на примерах при создании ряда прикладной систем.

2. Методы верификации, конфигурирования архитектуры их модулей и тестирования копонентов и систем из них

2.1. Верификация моделей MF и систем

Истоки формальной верификации и доказательства правильности программ лежат в Венском методе (VDM). В нем дана система доказательство программ с помощью формальных методов проверки правильности или неправильности объекта в соответствии со спецификацией свойств программ. Объектами верификации являются модель характеристик MF для разных предметных областей и требования к разработке новой ПС, а также методы проверки, основанные на model checking.

В них свойства компонентовв, которые подлежат верификации в модели MF описываются средствами линейной темпоральной логики (Linear Temporal Logic (LTL)) или логики деревьев вычислений CTL (Computational Tree Logic).

Формальная верификация объектов – это *дедуктивный анализ* и проверка модели (model checking) заданным формальными спецификациями или темпоральной логикой.

Верификация model checking применяется к моделям ПС с конечным числом состояний. Основу метода верификации составляет математическая формулировка требований к создаваемым программам и алгоритмы формальной проверки требований: Модель Крипке, которая задается в виде: $M = (S, S0, R, L)$, где S — множество состояний, $S0$ — множество начальных состояний — отношений переходов $L: S \to 2^{AP}$x0 — функция разметки.

На основе спецификации формулируются утверждения, истинность которых необходимо проверить. По описанию модели осуществляется автоматическая верификация, в результате которой либо устанавливается, что модель удовлетворяет спецификации, либо делается ее опровержение в виде перечня действий над моделью, которые привели к нарушению спецификации.

Разработан ряд инструментов поддержки верификации моделей и систем:

FeatureModelPlugin FMP) – реализован как плагин Eclipse. **FMP** поддерживает моделирование, построение модели характеристик и конфигурирования СПС (http://gsd.uwaterloo.ca/featureModelingAndModelTemplates).

CaptainFeature основан на нотации FODA для построения MF и содержит конфигуратор для специализации созданных моделей (https://sourceforge.net/projects/captainfeature/).

Requiline – это инструмент инженерии требований для эффективного управления в SPLE. Из описаний характеристик модели и требований «выводится» конфигурация продукта. Включает функцию контроля (checker) согласованности и не выполняет другие операции анализа. Проводится в таких средах функционирования как Microsoft.NET, Oracle DBMS.

Pure::Variants – инструмент поддержки моделирования характеристик, конфигурирования с помощью средств Prolog, решателя ограничений (constraintsolver) (https://www.pure-systems.com/products/pure-variants-9.html).

AHEAD ToolSuite (ATS) – это набор инструментов SPLE для разработки и поддержки характеристик и их компоновки. Можно выполнять определенные операции анализа модели с помощью SAT-решателей и сохранять ее в виде правил грамматики (http://www.cs.utexas.edu/~schwartz/ATS.html).

2.2. Управление вариабельностью моделей и систем

Вариабельность – это свойство системы к расширению, изменению, приспособлению или конфигурированию с целью использования в определенном контексте и обеспечении последующей его эволюции (ISO/IEC FDIS 24745-2009). Модель MF формируется в процессе разработки продукта (Software Product Line Engineering, SPLE) и включает общие функциональные и нефункциональные характеристики входящих элементов в систему, которые могут использоваться членами семейства СПП при создании разных вариантов ПС, сконфигурированных из ГОР по точкам вариантности.

Точка вариантности – это место в системе, по которому осуществляется выбор варианта ПС. Эта точка транслируется в коллекцию вариантов, присоединяемых к ядру изготавливаемой системы. В SPLE определен процесс создания ПС из ГОР (ранее они назывались компонентами повторного использования – КПИ). При производстве ПС из ГОР создается ПС и семейство СПС. MF в линии SPLE формировалась на двух процессах: инженерия предметной области и инженерия приложений.

Инженерия предметной области (**domain engineering)** состоит в определении и реализации общих функций с обеспечением вариабельности ПС при изготовлении нового *варианта* продукта.

Инженерия приложений **(application engineering)** состоит в определении специфики приложений и задания артефактов, необходимых пользователю.

Внесение изменений в СПС может быть выполнено на уровне предметной области и приложений.

Управление вариабельностью СПС проводится по точкам вариантности, вариантным артефактам ПС, ограничениям и зависимостями через предикаты, определенные на множестве точек вариантов ПС. Для управления вариабельностью используется метод Е. Деминга, основанный на функциях *F1- F4* (рис.1), которые задают действия по планированию и контролю проведения изменений в СПС.

Рис. 1. Функции действий в цикле Деминга

Каждая функция *F1- F4* выполняет следующее:

F1 – подготовку артефактов СПС (**Act**);

F2 – планирование системы СПС из артефактов (**Plan**);;

F3 – системный контроль и проверка состояния изменений СПС (**Check**);

F4 – выполнение актуализации систем СПС (**Do**).

Управление вариабельностью СПС с учетом требований *R* (Requriment) состоит в:

1) обосновании решений для функции *F1* (*R1*);

2) согласовании способа реализации артефактов на процессах СПС (*R2*);

3) реализации проверки правильности создания СПС (*R3*);

4) отслеживании связей между характеристиками ПС и СПС (*R4*).

В соответствии с требованиями *R1 – R4* и функциями *F1 – F4* осуществляется проверка модельной среды и модели вариабельности СПС

2.3. Конфигурационная сборка ГОР, КПИ в ПС и СПС

К операциям конфигурационной сборки отнесены операции выполнения функций управления вариабельностью F1–F4 с использованием ГОР, которые размещаются в репозитории с учетом

– требований и предикатов выбора необходимых элементов ГОР;

– операций управления конфигурацией ПС в СПС;

– идентификация конфигурации, ее элементов и данных;

– управление процессом изменений ГОР;

– изменение модели вариабельности под новые требования;

– оценка уровня вариабельности ПС и СПС.

Для управления артефактами ПС и СПС и их вариабельностью создается, так называемая *модельная среда* конфигуратора, в которую входят:

– процесс сборки ГОР, КПИ и артефактов системы;

– схемы формального описания артефактов;

– определение модели МХ с отмеченными вариантными точками;

– БД, включающая описание требований, архитектуру, набор базовых ГОР, тесты;

– конфигуратор – инструмент для объединения артефактов в ПС и СПС.

При управлении конфигурацией проводится сборка данных для проведения таких стандартных операций, как *отчетность и аудит конфигурации* на определение запланированной функциональности ПС или СПС. Конфигуратор собирает требуемые артефакты и ГОР в структуру СПС.

Одним из шагов конфигураторного сборщика - компиляция исходного кода, где файлы превращаются в промежуточный код или в код выполнения. Процесс связи представляет собой замену адресов функций внешними характеристиками библиотек и реальными адресами, используемыми в выполняемой программе. В готовых элементах – это компонент конфигуратора, который объединяет функциональные элементы и их интерфейсы в MF.

Разработана схема конфигуратора ст. инженером Колесником А. при написании кандидатской диссертации (2013), принцип работы которого приведен на рис.2. В нем стрелка 1 задает задание на доработку или создание нового ГОР и его отправку в репозиторий. Каждый ГОР состоит из двух файлов: *.cs, которые задают бизнес логику процесса и *.xoml представляют собой атомарные абстрактные объекты домена и алгоритм выполнения логики. Интерфейс конфигуратора устанавливает связь с репозиторием (стрелка 3-5) и получает список всех доступных ГОР. Они отображаются в правом верхнем углу схемы конфигуратора.

После построения новой модели, она размещается на сервере (стрелка 4) или в репозитории (стрелка 2, 4) ГОР для последующего использования. Каждый элемент конфигуратора является отдельным самостоятельным ГОР (рис2).

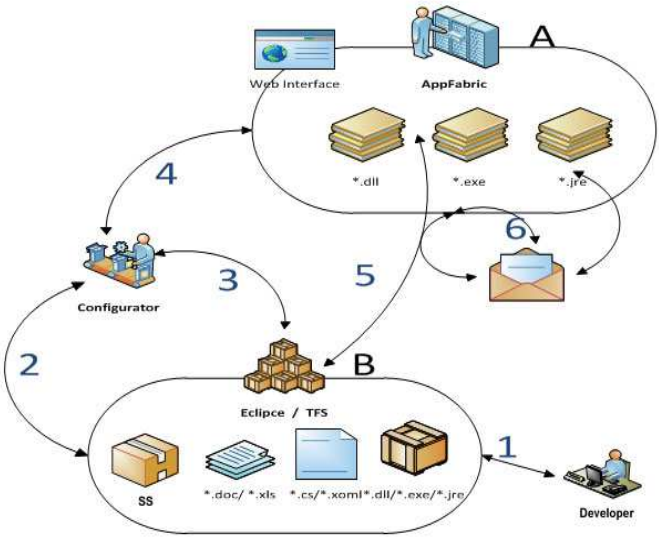


Рис. 2. Общая модель работы конфигуратора в среде .Net

2.4. Тестирование ПС семейства

В состав СПС входят ГОР и рабочие продукты для тестирования (планы, набор тестов, тестовые данные и др.). Тест-код создается для тестирования отдельных элементов ПС и образует набор тестов СПС. Метод тестирования СПС базируется на методе Дж. МакГрегора (McGregor) «от требований» (requirements-based testing). В нем задаются действия по управлению тестированием «от требований» с помощью тестов, проверяющих функциональные и интерфейсные объекты. Контроль степени тестирования объектов и интерфейсов обеспечивает оценку критерия качества СПС для вариантов ПС. В качестве инструмента тестирования используется TestManager фреймворка VisualStudio, содержащий средства проверки правильности тестирования разных ГОР и планирования процесса тестирования при выполнении тестовых сценариев, приведенных в таблице 1

Таблица 1. Основная схема тестирования СПС

Виды тестирования	Объекты тестирования
Тестирование программной архитектуры системы	Все ПС семейства, отдельные ПС и функциональные элементы.
Набор тестов СПС, адаптированных к конкретной ПС	Все ПС семейства, отдельные ПС семейства
Тестирование требований (ScenTED)	Отдельные ПС семейства
Самотестирование	Отдельные объекты и интерфейсы
Применение метаданных	Отдельные объекты и интерфейсы
Оценка тестопригодности компонентов (для повышения производительности)	Отдельные объекты и интерфейсы
Ориентация на сервисное обеспечение и надежность	Отдельные объекты и интерфейсы
Автоматическая генерация тестов по спецификациям в булевой форме	Все ПС семейства, отдельные ПС семейства
FCTA (Fault Contribution Tree Analysis)	ПС семейства

Таким образом, по данному методу проводится:

1. Тестирование артефактов, приложений, отдельных ПС и ГОР.

2. Тестирование отдельных характеристик объектов СПС с помощью тестов (автономных и общих).

3. Проверка степени тестирования функциональных и интерфейсных объектов СПС в среде Microsoft VisualStudio на инструментах TeamFoundationBuild и Microsoft Test Manager.

В завершении тестирования определяется количественная метрика *KT* степени тестирования объектов ПС:

KT = 1, если операции тестирования объекта независимы одна от другой;

KT = 0, если операции зависят от пути выполнения и взаимосвязей.

KT принадлежит отрезку [0; 1] и вычисляется по следующей формуле:

$$KT = \frac{1}{n}\sum_{i=1}^{n} KTi$$

Конечное значение *KT* СПС задает степень проверки:

KT = 1, если все объекты проконтролированы;;

KT = 0, если не все объекты проконтролированы;

0<*KT*<1 означает, что объекты СПС частино проконтролированы.

При тестировании интерфейсов взаимодействующих объектов проводится оценка метрики контроля интерфейса C*I* по формуле:

$$CI = 1/n\sum_{i=1}^{n} CI_i$$

где CI_i – степень корректности преобразования типов данных в *i*-ом интерфейсном объекте. Если $CI = 1$, интерфейс полностью проконтролирован, $CI = 0$ интерфейс не полностью проконтролирован; $CI \in (0;1)$ – интерфейс частино проконтролирован.

Количественное значение метрики *CI* означает*:*

1 – контроль CI_i интерфейса завершен;

0 – в противном случае - нет.

Этим завершается оценка тестирования функциональных объектов и их интерфейсов с помощью тестов СПС.

Последние годы в технологии и инженерии сформировались новые методы моделирования ПС и их семейств. Они ориентированы на изменение (вариабельность) готовых (legacy Systems) и вновь создаваемых систем (см. раздел 1 стр.24…). Одна из первых моделей характеристик МХ, разработана в SEI (sei.cmu.edu) для продуктовой линии Product Line/Product Family (2002) изготовления программных продуктов (ПП) и их семейств (СПП) из готовых ресурсов (artifacts, reuses, assetsи др.). Дано определение СПП стандарта ISO/IEC FDIS 24765:2009 «СПП – это группа продуктов или услуг, которые имеют общее управляемое множество свойств, удовлетворяющих потребностям определенного сегмента рынка или вида деятельности». Сформирована концепция конвейерной сборки систем и семейств ПС, основанная на модели МХ К. Чарнецки (K. Czarnecki) из готовых reuses (Reusability, 1987).

Под ***системой*** понималась совокупность программных артефактов и reuses, из которых осуществляется конфигурационная сборка. ***Семейство ПС*** – это совокупность ПС, которые имеют множество внешних характеристик, свойственных каждому отдельному члену ПС. Разработан логико-математический ОКМ моделирования функциональных и интерфейсных элементов данной предметной области. Эти элементы являются общими для всех ПС семейства и используются при генерации выходных вариантов ПС или СПС.

Концепция вариабельности для ПС и их семейств начала использоваться и в действующих Legacy-системах (ОС IBM, Intel, Linux и др.). Она обсуждается на международных конференциях (ICTERI 2006–2015, Reusability 1994–2015, Vamos 2005–2016 и др.) и реализуется индустриально (www.sei.cmu.edu/productlines, www.7dragons.ru/ru и др.). Общее, что их объединяет – это линии производства продуктов из ГОР, модели вариабельности и их верификация, управление конфигурационной сборкой систем из ГОР для получения вариантов систем.

В работе предлагаются новые оригинальные модели ПС и МХ, ориентированные на обеспечение изменяемости программных и операционных систем. Рассматривается метод ОКМ проектирования ПС и их семейств с помощью объектов, компонентов и сервисов. Описывается метод управления конфигурацией вариантов ПС из ГОР с использованием вариабельных моделей МХ, ПС и ОС.

Модель вариабельности систем

Модель характеристик для производства ПС впервые предложил К. Похл (K.Pohl) с учетом требований и особенностей новых языков и инструментов их реализации (*VSL*, *ConIPF*, *CBFM*, *Koalish, COVAMOF* и др.), с помощью которых задаются понятийные и количественные зависимости между характеристиками МХ и в конфигурационном файле ПС.

Дается описание основ моделирования вариабельных продуктов и систем, предложенных в работах К. Похла, К. Чарнецки, а также логико-математического четырехуровневого объектно-компонентного метода (ОКМ) к проектированию ПС и их семейств. Предложен формальный аппарат определения объектной модели и МХ в ОКМ для проектирования семейств СПС с помощью функциональных и интерфейсных объектов. Определена модель конфигурационной сборки и управляемое изготовление вариантов систем и их семейств на сайте www.7dragons.ru/ru.

2.5. Вариабельность в пространстве проблем и решений для ПС и ПТС

К. Чарнецки моделирует архитектуру ПС и СПС в пространстве проблем (problem space) и решений (problem solution), аналогично Feature-oriented подхода SPLE. Он проводит анализ характеристик ГОР, которые входят в модель ПС и СПС, и внешние из них отображаются в FM, если они реализуют требования к ПС. Между характеристиками и требованиями устанавливается ($n \times m$)связей. Для каждой ПС определяются характеристик базовых функций, которые необходимы пользователю и размещаются в областях проблем и решений. Характеристики, которые соответствуют требованиям, описываются в языке DSL (Domain Specific Language) и присоединяется к ядру СПС в точках вариантности. Характеристики отражают интерфейсные, производственные и реализационные свойства ГОР в ПС и СПС.

Теорема. Взаимодействие двух функциональных объектов является корректным, если первый объект полностью обеспечивает функции и передачу данных, необходимых другому объекту: $In(fo_k) \subseteq Out(fo_l)$.

Далее приводятся описание моделей систем и их вариабельности с использованием рассмотренных объектов.

Определение модели ПС и ее вариабельности

Модель ПС – это $M_{ПС} = (CL, M_f, M_s, M_i, M_d)$, (1)

где CL – языки в $L = L_1, L_2, ..., L_N$;

$M_f = (O_1, O_2, ..., O_r)$ – множество функциональных элементов;

$M_s = (Ms_{\text{in}}, Ms_{\text{out}}, Ms_{\text{inout}})$ – множество интерфейсных сервисов (s) – входного Ms_{in}, выходного Ms_{out} и серверного Ms_{inout};

M_i – множество интерфейсов в языке IDL;

M_d – множество данных и метаданных ПС.

Вариабельность ПС, СПС реализуется путем конфигурирования ГОР или взятых из библиотек reuses.

Модель вариабельности ПС – $MF_{var} = (SV, AV)$, где

SV – подмодель вариабельности артефактов в структуре ПС;

AV – подмодель вариабельности готовых артефактов продукта ПС.

Модель MF_{var} ориентирована на обеспечение изменений артефактов ПС, снижение затрат и уменьшение стоимости разработки системы.

Подмодель $SV = ((G_t, TR_t), Con, Dep)$,

где $G_t = (F_t, LF_t)$ – граф артефактов F_t и языка описания LF_t на уровне t; TR_t – связи артефактов на уровне t;

Con и Dep – предикаты на декартовом произведении множества артефактов, которые задают ограничения и зависимости между функциональными элементами F_t и их показателями качества.

Подмодель AV определяет ГОР, которые хранятся в репозитории. Эта подмодель отображает характеристики ГОР и системы, а также аспекты интерфейсов между ними на разных уровнях модели. Точки вариантности обрабатываются конфигуратором и заменяют некоторые ГОР другими, более корректными или новыми.

Семейство СПС из программных элементов

СПС – это совокупность систем (программных или операционных) с общим множеством понятий, специфических данных, функциональных и интерфейсных характеристик, которые присущи каждому члену семейства.

Модель СПС имеет вид:

$M_{СПС}$ = (($M_{ПС1}$, $M_{ПС2}$, …, $M_{ПСk}$), (M_{gf}, M_{sf}=(M_{sf1}, M_{sf2}, …, M_{sfk})), M_i, M_d))),

где $M_{ПС1}$, $M_{ПС2}$, …, $M_{ПСk}$ – множество членов семейства ПС;

M_{gf} – множество внешних функций ПрО, представленных, объектом, компонентом;

M_{sf} = ⟨M_{sf1}, M_{sf2}, …, M_{sfk}⟩ – множество внутренних функций;

M_i – множество интерфейсов объектов;

M_d – множество данных и метаданных ПрО.

Модель СПС включает совокупность функциональных элементов и их интерфейсов (или контрактов) отдельных ПС, взаимодействующих между собой. Эти интерфейсы могут группироваться в разные сочетания с учетом семантики внешних и внутренних характеристик функций, входящих в модель СПС.

Модель вариабельности СПС – это кортеж:

SVспс = ((CF; (DR,TC); (CM,FR,TS,TA); (ER,TF)); Con; Dep), (2)

где CF – характеристики функций системы для заказчика продукта, Con – ограничения и Dep зависимости между ними;

DR – характеристики функций, связанные с требованиями к ПС;

TC – связи между требованиями к ПС и их свойствами для потребителя;

FR – множество функций;

CM – множество формально описанных элементов функций FR:

TS – множество формально описанных тестов для элементов системы;

TA – интерфейс между программными элементами СПС;

ER и TF – данные БД для обработки ПС на множестве элементов CM.

Модель вариабельности в артефактах СПС – это тройка

FA = (CAS; REP; DBF), $CAS = \cup_{t=1,\dots,4} Aft$, (3)

REP = (RPR∪RPA∪RPE); RPE = CM∪TS, (4)

где Aft = (f(aft)) – формальное представление артефактов СПС на уровнях t=1 – задания *aft*;

RPR, RPA, RPE – репозитории ресурсов СПС уровня t: документов требований к ПС (t=1*),* типичных архитектур ПС *(*t=2) и готовых ресурсов (*t=3*);

DBF – БД для обработки членов ПС и СПС;

CM, TS – множества функциональных элементов и тестов для них.

К оценке вариабельности СПС сложилось два подхода. В первом подходе необходимо интегрировать вариабельность модели, а в другом – строить специальные модели вариабельности МХ. Дополненная ортогональная модель вариабельности согласовывает состав и взаимосвязи элементов СПС, артефактов процессов сборки ПС и СПС. Эта модель входит в состав интегрированной модели вариабельности OVM, которая позволяет оценить уровень вариабельности продукта с учетом требований к артефактам.

Модель OVM имеет вид:

OVM= *(EVM*; *VP*; *VAR)*; *EVM* = (*VL*, *VR)*, (5)

где *VP = (VpR*, *VpA*, *VpE*, *VpB)* – множества точек вариантности в структуре СПС, которые задают характеристики ПС, включая поля таблиц БД (*VpB*), ограничения *Con* и зависимости *Dep*;

VL – модель оценки уровня вариабельности с учетом требований *vrl к* компонентам архитектуры *val*, артефактам *vel* и данных *vbl*;

VR – модель оценки уровня вариабельности СПС с учетом требований *vrr*, архитектуры *var*, артефактов *ver* и данных *vbr*.

Модель OVM определяет два вида оценок вариабельности СПС – уровень и соответствие потребностям. Оценка уровня вариабельности и степени ее соответствия потребностям выполняется с помощью дерева ценности и параметров *VL u VR,* которые учитывают трудозатраты и частоту передачи вариантов ПС заказчику. Прогнозирование вариабельности можно выполнить также с помощью Байесовской сети.

Действующие модели вариабельности

Объектами верификации является модель МХ и требования к разработке ПС. Эти элементы задаются в качестве параметров при обращении к инструментам верификации. К ним относятся:

FeatureModelPlugin или FeaturePlugin (FMP) – реализован как плагин Eclipse. **FMP** поддерживает моделирование, построение модели характеристик и конфигурирования СПС (http://gsd.uwaterloo.ca/featureModelingAndModelTemplates).

CaptainFeature – использует нотацию FODA для построения МХ и содержит конфигуратор для специализации созданных моделей (https://sourceforge.net/projects/captainfeature/).

Requiline – это инструмент инженерии требований для эффективного управления в SPLE. Из описаний характеристик модели и требований «выводится» конфигурация продукта. Включает функцию контроля (checker) согласованности и не выполняет другие операции анализа. Проводится в таких средах функционирования как Microsoft.NET, Oracle DBMS.

Pure::Variants – инструмент поддержки моделирования характеристик, конфигурирования с помощью средств Prolog, решателя ограничений (constraintsolver) (https://www.pure-systems.com/products/pure-variants-9.html).

AHEAD ToolSuite (ATS) – это набор инструментов SPLE для разработки и поддержки характеристик и их компоновки. Можно выполнять определенные операции анализа модели с помощью SAT-решателей и сохранять ее в виде правил грамматики (http://www.cs.utexas.edu/~schwartz/ATS.html).

3. Подходы к созданию вариантов ядра OS LINUX для предметных областей знаний

Операционная система Linux – это совокупность отдельных программных фрагментов функции и специальных функций ядра ОС Linux, управляющих задачами, процессами и прикладными системами. Такие функции могут формировать самостоятельные компоненты с заданным поведением и способствовать общему функционированию варианта ядра ОС. ОС Linux можно условно разделить на два уровня (рис. 1).

На верхнем уровне находится пользовательское пространство, где исполняются приложения пользователей. Под этим пространством располагается пространство ядра. Имеется библиотека GNU C (glibc), содержащая интерфейс системных вызовов для связи с ядром. Ядро и пользовательское приложение располагаются в разных защищенных адресных пространствах. Каждый процесс в пространстве пользователя имеет свое собственное виртуальное адресное *пространство*.

Рис.1. Уровни ядра ОС linux

Ядро Linux можно, в свою очередь, разделить на три больших уровня. Наверху располагается интерфейс системных вызовов, который реализует базовые функции (чтение и запись). Ниже интерфейса располагается архитектурно-независимый код ядра. Этот код является общим для всех процессорных архитектур, поддерживаемых Linux. Еще ниже располагается архитектурно-зависимый код, образующий. BSP (Board Support Package), который зависит от процессора и платформы для конкретной архитектуры.

Linux можно откомпилировать для огромного количества разных процессоров и платформ, имеющих разные архитектурные ограничения и потребности. Например, Linux может работать на процессоре как с блоком управления памятью (MMU), так и без MMU. Поддержка процессоров без MMU реализована в версии ядра ОС linux.К основным компонентам ядра Linux относятся (рис. 2):

Рис.2. Компоненты ядра ОС linux

– интерфейс системных вызовов - тонкий уровень для вызова функций ядра;

– управление процессами или потоками с помощью интерфейса программирования приложений (API) через SCI для создания нового процесса и алгоритма планировщика, время работы которого не зависит от числа потоков;

– управление памятью с участием ааппаратных средств, устанавливающих соответствие между физической и виртуальной памятью;

– виртуальная файловая система (VFS), которая предоставляет общую абстракцию интерфейса к файловым системам и служит для коммутации между SCI и файловыми системами ядра.

– сетевой стек имеет многоуровневую структуру самих протоколов. Internet Protocol (IP) - это базовый протокол сетевого уровня, располагающийся ниже транспортного протокола Transmission Control Protocol, TCP). Выше TCP находится уровень сокетов, вызываемый через SCI.

Уровень сокетов представляет собой стандартный API к сетевой подсистеме. Он предоставляет пользовательский интерфейс к различным сетевым протоколам.

– драйверы устройств обеспечивают возможность работы с конкретными аппаратными устройствами.

В Linux разработан гипервизор, с помощью которого можно строить ОС для других систем. Был реализован новый интерфейс, позволяющий исполнять поверх его ядра другие ОС.

Подход к моделированию операционных систем (ОС)

Проектирование любой системы - это процесс определения архитектуры, компонентов, интерфейсов, других характеристик системы и конечного состава программного продукта.

Базовая концепция программирования ПО - это методология создания архитектуры с помощью разных методов (объектного, компонентного и др.), процессы ЖЦ (стандарт ISO/IEC 12207) и техники - декомпозиция, абстракция, инкапсуляция и др.

Ключевыми вопросами моделирования ПС является: декомпозиция программ на функциональные компоненты для независимого и параллельного их выполнения, принципы распределения компонентов в среде выполнения и их взаимодействия между собой, механизмы обеспечения качества и живучести системы и др.

При разработке операционной системы на базе Linux основным параметром, определяющим необходимый функционал, является область применения этой ОС.

Функциональные элементы ОС Linux отвечают за определенную сторону, а именно:

– видеопамять и работа с ней;

– HAL (Hardware Abstraction layer) - поддержка нескольких аппаратных архитектур, включая процессы, семафоры и.др.:

– управление памятью;

– управление исполнением управление устройствами;

– виртуальные файловые системы;

– API (Application Programming Interface) IPC (Interprocess Communication) и т.д.

В маленьких встроенных системах, не будут использоваться функциональные модули, отвечающие за видеопамять, API или управление исполнением.

Выбор элементов ядра для сборки элементов варианта ОС Linux

Сборкой (англ. config, assembly, build) называется процесс получения продукта из исходного кода. Чаще всего используется компиляция, компоновка и выполнения инструментальными средствами автоматизации.

В ходе сборки ядра Linux применяется операция сборки **make** и специальные make-файлы, в которых указаны зависимости файлов друг от друга и правила для их удовлетворения. На основе информации о времени последнего изменения каждого файла make определяет и запускает необходимые программы. В данном случае make выступает в роли инструмента управления сборкой, относящегося к инструментам конфигурационного управления.

Существует ряд стандартных операций, таких как:

– config - традиционный способ конфигурирования. Программа выводит параметры конфигурации по одному, предлагая установить для каждого из них свое значение.

–oldconfig - файл конфигурации создаётся автоматически, основываясь на текущей конфигурации ядра.

– defconfig - файл конфигурации создаётся автоматически, основываясь на значениях по умолчанию.

– menuconfig - псевдографический интерфейс ручной конфигурации, не требует последовательного ввода значений параметров. Рекомендуется для использования в терминале.

– xconfig - графический (X, QT) интерфейс ручной конфигурации, не требует последовательного ввода значений параметров.

– gconfig - графический (GTK+) интерфейс ручной конфигурации, не требует последовательного ввода значений параметров. Рекомендуется для использования в среде GNOME и др.

Стандартный путь такого конфигурирования – использование операции make с параметром, зависящим от выбранного инструмента конфигурации. Например, если выбрана конфигурация через config, то осуществляется команда make config. После выполнения этой команды запускается сборка с помощью make. В случае установки ОС на многоядерную машину можно задать количество потоков с тем, чтобы ускорить процесс сборки.

С общей точки зрения модель ОС Linux – это множество разных операционных артефактов (функций) и интерфейсов между ними:

$$M_{oc} = (C_k, M_f, M_s, M_o, M_i), \quad (1)$$

где C_k – множество отдельных фрагментов, артефактов ОС;

M_s – множество характеристик ядра Linux (более 10000 для версии 4.11);

M_o – множество ограничений характеристик функций на глубине дерева зависимостей (глубина 8 для 22 ограничений);

M_i – множество интерфейсных характеристик (menuconfig) в ядре Linux.

Модель вариабельности ОС согласно проведенных исследований и обсуждений на конференциях VAMOS имеет следующую модель ОС:

$MFvar = (SV_{inoc}; AV_{inoc})$, где

SV_{inoc} – подмодель вариабельности артефактов структуры ядра ОС Linux;

AV_{inoc} – подмодель вариабельности продукта ОС Linux.

Модель MF_{var} задает изменяемость отмеченных артефактов, определяет затраты и уменьшает стоимость варианта системы.

Подмодель $SV_{inoc} = ((G_{inoc}, TR_{inoc}), Con_{inoc}, Dep_{inoc})$,

где $G_{inoc} = (F_{inoc}, LF_{inoc})$ – граф артефактов ядра ОС Linux;

TR_{inoc} – набор связей между зафиксированными артефактами ОС Linux;

Con_{inoc} и Dep_{inoc} – предикаты ограничений и зависимостей между отдельными функциями на множестве артефактов ОС Linux.

AV_{inoc} определяет изменяемость структуры ОС из готовых ресурсов, которые хранятся в БД системы. Эта подмодель отображает характеристики артефактов и отношений между ними на дереве зависимостей. Точки вариантности, отмечающие изменяемые артефакты, обрабатываются конфигуратором и способствуют получению варианта ОС для конкретного применения.

Практическими экспериментами в системе ОС Linux определен набор функций и их характеристик. Для генерации из них некоторого варианта ОС требуется извлечь из ядра системы необходимые готовые артефакты или функциональные элементы с их интерфейсами. Для извлечения изменчивости (Mining variability) используется инструмент LEADT. В нем содержатся средства поиска и анализа готовых ГОР в унаследованном коде ОС. Данный инструмент извлекает отдельные характеристики функций в Legacy code системы, восстанавливает архитектуру для практического использования и оформляет их в формате для представления в репозитории системы. Из выявленных функций формируется новый вариант системы, который удовлетворяет требованиям системы.

3.1. Моделирование вариабельности с помощью диаграмм характеристик

Данное моделирование осуществлено в системе *FODA*, которая и базируется на подходах *featureRSEB*, Рибича, *Forfamel*, *RequiLine*, *CBFM* и представлены в магистерских работах Газазулиной Р., Гарипова И, Губайдулина, Козина Е. и др.

FODA (Feature-oriented domain analysis) является основоположником в сфере создания вариабельных систем и является простой моделью с признаками, которые организованы с использованием “состоит из” и “обобщение/специализация” отношений, используя И/ИЛИ граф. Модель разделяется на пространство задач и решений, которые в свою очередь используют цель, контекст использования и атрибуты качества; способности, рабочую среду и конструктивные особенности (рис.3).

Рис.3. Схема моделирования ПС в системе FODA

Моделирование вариабельности средствами UML

UML (Unified Modeling Language) содержит ***сценарии использования*** (use cases). Этот язык был первоначально реализован в Rational Rose. С Rose, а затем оформлен как стандарт IBM Rational и консорциума OMG (Object Management Group) в виде объектной модели. UML Его начали применять многие фирмы-производители программного обеспечения. UML – это диаграммный язык документирования и моделирования систем. В нем графовая модель системы задается набором Use Case диаграмм вида: варианты использования (use case diagrams); классы (class diagrams); поведение (behavior diagrams); взаимодействие (interaction diagrams); состояние (statechart diagrams); деятельности (activity diagrams), реализации (implementation diagrams) и др. Модель системы – это архитектура системы, которая трансформируется к программным текстам отдельных элементов в языках C++, Pascal, Basic и др. Фактически промышленных продуктов средствами UML было создано мало и в экспериментальном виде.

3.2. Вариабельные зарубежные и отечественные системы

Проблеме вариабельности посвящены работы**:** *ConIPF*, *CONSUL/Pure::Variants*, *GEARS*, включая *Koalish*, подход Ван драл Хока, которые интегрируют вариабельность в существующий язык моделирования архитектуры (языки *Koala* и *xADL*) и *OVM*, а также *VSL* для моделирования точек вариантов с целью обеспечения изменяемости отдельных артефактов ПС и др.)

Среди них сложились подходы, которые хорошо формализованы и реализованы, а также доступны для оценивания, сопоставления и моделирования вариабельности в пространстве готовых производственных решений: *VSL*, *ConIPF*, *CBFM*, *Koalish* и *Pure::Variants*. Сущность вариабельности и ее моделирование с помощью языков и инструментов представлена в работах:

– VSL (Variability Specification Language);

– ConIPF. Язык моделирования AMPL (Asset Model for Product Lines). Web-инструменты для моделирования (KBuild) и конфигурирования (KConfig);

– CBFM (Cardinality-Based Feature Modeling) и инструмент FeaturePlugin;

– Koalish. Расширяет концепты используемого языка Koala средствами моделирования вариабельности в рамках архитектуры и реализации систем и семейств СПС;

– Pure::Variants (Pure Systems Website), представленный набором инструментов на основе подхода CONSUL, использующего при моделировании вариабельности, объектную модель с отношениями специализации (is-a) и агрегации (part–of), а также Prolog для описания ограничений и проверки согласованности моделей характеристик и конфигураций продукта;

Названные подходы не ориентированы на преодоление проблем, связанных с неопределенностью, неполнотой и неточностью связей в модели вариабельности, вызванных наличием зависимостей между вариантами решений. Для этого потребовалась организация много итерационного процесса «вывода» программных систем из артефактов семейств. Подходы не содержат формальной поддержки процесса построения программных систем в семействе, так как основываются на знаниях экспертов.

Для преодоления указанных проблем исследованы средства фрейворка *COVAMOF* (ConIPF Variability Modeling Framework) для моделирования вариабельности. В нем вариабельность и зависимости между характеристиками моделируются унифицировано на разных уровнях абстракции (рис. 4). Инструмент *COVAMOF* может погружаться в среду MS Visual Studio.

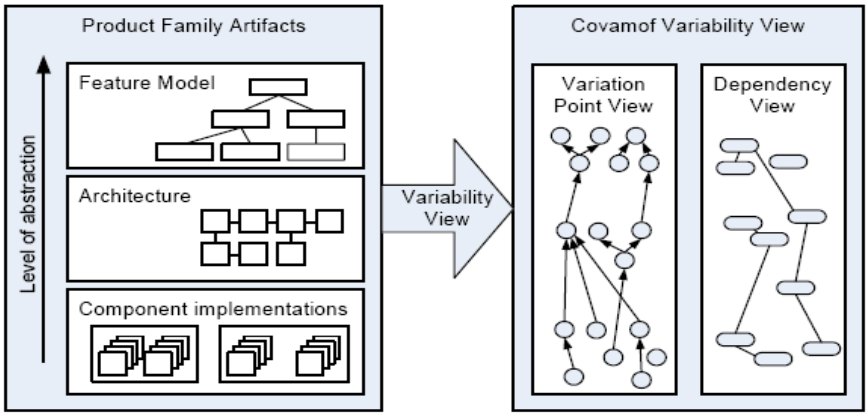


Рис. 4. Ракурсы представления вариабельности в COVAMOF

Инструмент *Kumbang* с помощью компонента *Kumbang Modeler,* моделирует вариабельность, используя онтологию домену, реализованную как plug-in в Eclipse.

Клаус Похл ввел понятие вариабельности в виде модели характеристик (Feature Model) для элементов системы, помечаемых вариантными точками для изготовления ПП.

Модель FM формируется в процессе разработки ПП аналитиками и разработчиками и включает общие функциональные и нефункциональные характеристики элементов системы (рис.5), которые используются членами семейства при решении задач. Она имеет несколько видов представления – в семействе и в отдельных приложениях этого семейства.

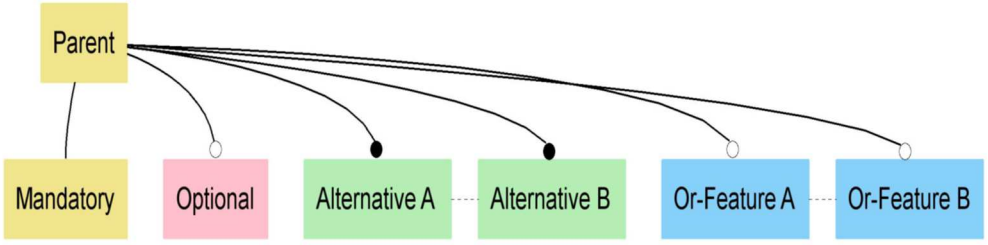


Рис. 5. Structure and notation of Feature models

Задача обеспечения вариабельности ПС базируется на методах разработки и конфигурирования продукта (product configuration) из готовых reuses и процессов управления конфигурацией (configuration management), направленных на определение и внедрение эффективных процедур разработки и эволюции систем, а также их адаптации к новым условиям функционирования в современных гетерогенных средах.

К главным аспектам обеспечения вариабельности ПП и систем отнесены:

– моделирование вариабельности на наивысшем уровне характеристик объектов, которые имеют вариантные характеристики. Данное моделирования задает вариабельность таким способом, который улучшает понимание задач и функций системы и возможность управлять ими;

– реализация вариабельности архитектуры системы посредством механизмов вариабельности на уровнях ее реализации (или она будет реализована в момент определения архитектуры СПС или во время процессов разработки продукта СПС). Соответствующее решение принимаются аналитиками и разработчиками СПС с учетом их собственного опыта, задач домена и механизмов теории обеспечения вариабельности (рис.3);

– управление вариабельностью, связано с планированием, контролем и регуляцией в ходе жизненного цикла (ЖЦ) объектов и системы, которые ориентированы на изменяемость.

Технология SPLE (Software Product Line Engineering) обеспечивает производство вариабельных ПП и семейств СПП. Она формировалась более 8 лет и экспериментально отработана на ряде коммерческих продуктов. Технология SPLE представлена в базовой работе K.Pohl и 12 соавторов в институте SEI USA. Отдельные аспекты индустрии систем, касающиеся производства вариабельных продуктов, нашли отражение в статьях и докладах специалистов на научно-технических конференциях SPLE (2006–2015), ICTERI (2007–2015), Reusebility (1998–2015) и др. На конференции ICTERI (2011-2013) автором представлена технология конвейерной сборки разнородных объектов (модуль, объект, компонент и др.), разрабатываемых средствами соответствующих парадигм программирования в стандартной форме и конфигурируемых методом сборки в новые вариантные структуры ПС и СПС в среде Grid. **SPLE** подход обеспечивает процесс создания ПП из базовых элементов, которые называют, компонентами повторного использования (КПИ). К ним относятся – reuses, assets, application и др. Данный SPLE подход направлен на производство ПП из отдельных элементов СПС и КПИ с вариантами, на снижение стоимости и увеличение производительности труда разработчиков ПС за счет КПИ и внесения изменений, как в отдельные элементы, так и в саму систему. При производстве ПП из готовых элементов, КПИ, ПС в СПС используется метод систематического изменения КПИ по модели Feature для варьирования ими в структуре ПП и стал самым эффективным средством для поддержки адаптивности ПС и СПС.

Система генерирующего программирования K.Chetrnetski осуществляет моделирование архитектуры системы в пространстве проблем и решений (рис. 6) средствами feature-oriented подхода.

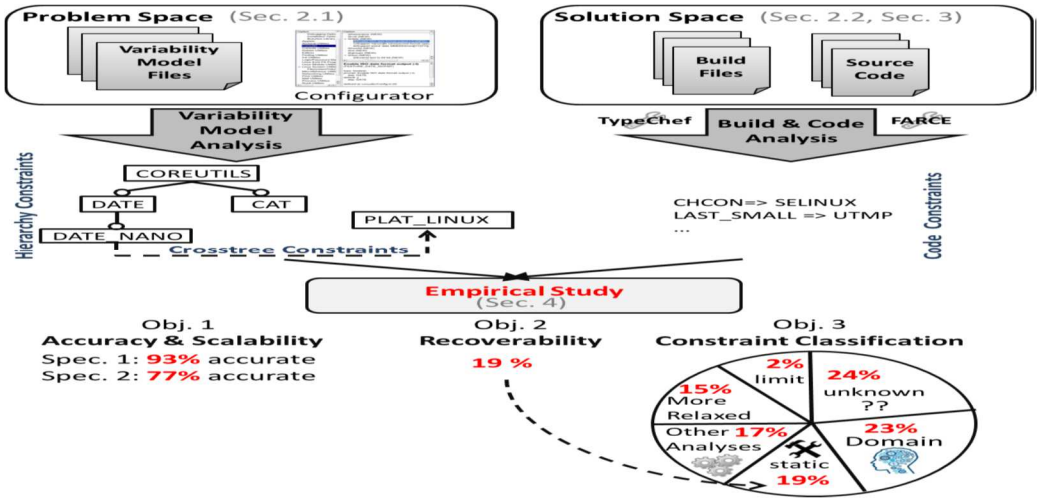


Рис. 6 Общий подход к представлению объектов по Chernetski

Его основу составляют характеристики семейства, которые отображаются в модели характеристик его членов (компоненты повторного использования, программные системы, reuses, assets и др.) и семейства в целом. В данной схеме модель характеристик (рис.7) отражает функции члена семейства, которая доступна пользователю системы и является обязательной, необязательной или альтернативной. В этой системе проводится построение модели вариабельности согласно приведенной схемы.

Рис. 7. Схема построения модели характеристик MF

Ими могут быть интерфейсные, проектные, производительные и реализационные свойства. При этом доменная модель может рассматриваться как метамодель предметно-ориентированного описания моделей семейства DSL с механизмами повышения уровня абстракции семейства ПС для ввода этих механизмов в процессе реализации отдельных членов семейства.

Моделирование архитектуры семейства проводится с помощью нового предметно-ориентированного языка DSL (Domain Specific Language), который определяет архитектурный стиль программной системы семейства из инвариантов системы,

механизмами изменчивости и эволюционного ее развития с помощью комбинаций соответствующих шаблонов проектирования этой архитектуры. При этом доменная модель программной системы семейства расширяется путем привлечения к ней инвариантных понятий и модели характеристик членов системы соответственно определенным их критериям и предметно-ориентированному описанию этой модели средствами соответствующего DSL. Этот язык разработан для описания специфики предметной области, спецификации отдельных членов СПС и генерации этих членов семейства систем. в рамках генерирующего программирования К. Чернецки. Методология разработки предметной области средствами DSL заключается в разработке ЯП для предметной области и реализации следующих этапов жизненного цикла: *анализа, реализации, использования.*

Этап анализа – это: определение предметной области; сбор всех релевантных знаний о предметной области; построение системы знаний в виде семантических нотаций и операций над ними; определение языка DSL для описания модулей и компонентов для задач предметной области; определение правил последовательного преобразования моделей MF and Msys и определения предикатов проверки семантики моделей.

Этап реализации – это разработка специального редактора семантики языка DSL; создание библиотеки примитивов верификации семантики; реализация генератора для выполнения последовательных трансформаций моделей по библиотечным вызовам.

Этап использования – это реализация модели с помощью языка DSL; преобразование моделей и их интеграция на каждом этапе; генерация кода системы к выходному коду.

Использования языка DSL позволяет: улучшить тестирование ПС и обеспечить полную автоматизацию ПО; трансформировать модели из одного DSL в другой DSL; давать описание предметной области на одном уровне абстракции с дополнительной детализацией на более низких уровнях; автоматизированную реструктуризацию системы и выполнение в среде WWW3C.

Имеются некоторые недостатки DSL, состоящие в повышении стоимости проектирования, реализации и сопровождения, а также в отсутствии доступных к реализации разных вариантов DSL. Необходимость сохранения равновесия между конструкциями, специфическими для ПО в DSL снижает эффективность по сравнению с ПО, описанным в общих ЯП.

Система Grid Европейского проекта Колайдер построена по принципу моделирования систем с применением модели MF и операции интеграции, сборки - buiding. Основным компонентом системы Grid (рис.8), выполняюшим задачи моделирования вариабельных систем, является подсистема Etics (*EInfrastructure for Testing, Integration and Configuration Software).*

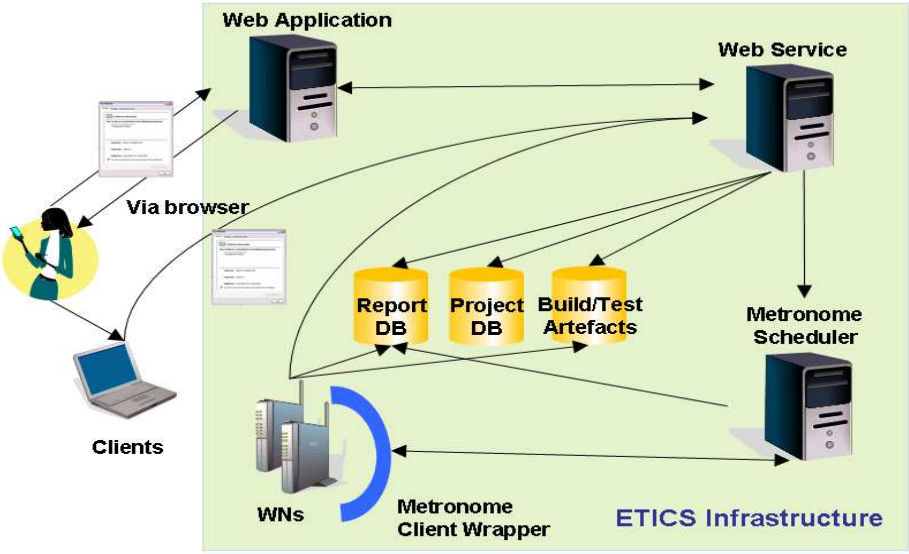


Рис.8. Инфраструктура Grid

Эта система обеспечивает управление малыми и большими программными проектами с применением стандартной конфигурационной сборки, стандарта качества, интеграция пакетов из модулей и их тестирование в среде этой системы. Система используется для запуска удаленной сборки buid и тестирования на компьютерах пользователей, работающих на разных платформах в своих разных организациях.

Процесс сборки и тестирования ПО в ETICS проходит три фазы:

1. Конфигурирование – это процесс регистрации программных компонентов для сборки и тестирования; создания конфигураций и присо+единения команд контроля версий, команд сборки, команд тестирования, свойств, переменных и зависимостей. Функциональные возможности задачи предоставляются Веб-приложением Сборкой и Тестированием (Build and Test Web Application) и Клиентом. Командная строка (Command-Line Client) обеспечивает доступ к Сервису Сборки и Тестирования (Build and Test Service).

2. Локальная сборка/тестирование базируется на собственных компонентах и информации конфигурации пользователя, зарегистрированного в системе Grid клиентом с помощью командной строки, позволяющей выполнять локальные сборки и тесты на машине пользователя.

3. Удаленная сборка/тестирование. Этот шаг позволяет запускать сборку и тесты на разных платформах Grid для подтверждения мобильности кода и генерации ряда отчетов о статическом и динамическом тестировании при разных условиях. Программные объекты могут быть следующих типов: “проект” (P), “подсистема” (S) и “компонент” (C). Их взаимосвязь показана примере связи модулей, компонентов в подсистеме, системе и проекте Системы.

Сборка в ETICS – это множество информационных структур, которое представляют определенную версию модуля. Сборка системы присоединяется к одному и только одному модулю (проекта, подсистемы, компонента) и каждый модуль может иметь одну или несколько конфигураций. *Конфигурация* – это иерархия дерева для отображения иерархии соответствующих модулей.

Таким образом, если проект системы ПС содержит определенные подсистемы, и каждая из них содержит компоненты, то конфигурация ПСа может содержать конфигурации

подсистем, а каждая конфигурация подсистемы может содержать конфигурации компонентов.

Дерево конфигурации содержит конфигурации для всех компонентов в соответствующем дереве модулей. Дерево модулей задает структуру проекта, а дерево конфигурации ее отдельные версии проекта и может содержать подмножество конфигураций. Деревья конфигураций для проектов и подсистем могут содержать произвольное сочетание имеющихся конфигураций, но не более одной конфигурации одного и того же модуля. Имеется три разных набора команд и два типа свойств (то есть свойства ETICS и переменные среды).

Команды VCS. Эти команды используется для управления кодом, сберегаесым в системе контроля версий.

Команды сборки. Эти команды для сборки кода, генерирации документации и пакетов с публикацией артефактов сборки со стандартным расположением и форматом.

Команды тестирования. Эти команды для выполнения динамических (интеграционных, системных) тестов программных пакетов и генерации отчетов по тестированию.

Конвейерная сборка разнородных объектов (модуль, объект, компонент и др.) предложена в ряде работ и сделан доклад на конференции ICTERI Лаврищева Е.М. В ней DSL представлена технология **сборки разных типов программ**, разрабатываемых средствами соответствующих парадигм программирования в стандартной форме и конфигурируемых в новые вариантные структуры ПС или СПС. Базовую основу составляет метод объектно-компонентного (ОКМ) моделирования ПС с применением логико-математического аппарата на четырех уровнях моделирования ПС по графу предметнной области или домена ОКМ метода.

Объектно-компонентный метод (ОКМ). В ОКМ реализована методология формального определения понятий предметной области, модель программной системы ПС и семейств программных продуктов СПП из объектов и компонентов. В нем объекты рассматриваются на логическом уровне моделирования ПС., как функции и методы предметной области с компонентов после физических реализаций объектов. Каждый компонент может быть реализацией нескольких объектов или некоторой части объектной системы, полученной на уровнях моделирования ПС. Компонент получает многократное использование в разработках новых ПС, обеспечивая объектную парадигму с использованием интерфейсов взаимосвязи объектов. В этом методе ПС определяется, как совокупность отдельных объектов и компонентов для реализации взаимосвязанных функций некоторой предметной области.

Клаус Похл ввел понятие вариабельности в виде модели характеристик (Feature Model) для элементов системы, помечаемых вариантными точками для изготовления программного продукта. *Вариабельность* – это свойство продукта (системы) к расширению, изменению, приспособлению или конфигурированию с целью использования в определенном контексте и обеспечения последующей его эволюции. *Точка вариантности* – это место в разработанной ПС, по которой осуществляется выбор варианта элемента системы.

Модель FM формируется в процессе разработки ПП аналитиками и разработчиками и включает общие функциональные и нефункциональные характеристики элементов системы (рис. 2), которые используются членами семейства при решении соответствующих задач. При этом для представления используются признаки характеристик: обязательные, альтернативные или факультативные.

Система FODA (Feature-oriented domain analysis) - основоположник в сфере моделирования вариабельных систем и является простой моделью с признаками - “состоит из” и “обобщение/специализация”, используя И/ИЛИ граф. Модель характеристик MF имеет простую структуру отношений - альтернативность, факультативность и взаимозависимость. Модель характеристик MF описывает продуктовую линию, например, для мобильного

телефона. Задача обеспечения вариабельности ПС базируется на методах разработки и конфигурации продукта (product configuration) из готовых reuses и процессов управления конфигурацией (configuration management), направленных на определение и внедрение эффективных процедур разработки и эволюции систем, а также их адаптации к новым условиям функционирования в современных гетерогенных средах. К главным аспектам вариабельности ПП и систем отнесены:

– моделирование вариабельности на наивысшем уровне характеристик объектов, которые имеют вариантные характеристики. Целью подходов моделирования является представление вариабельности таким способом, чтобы улучшить понимание задач и функций системы и возможность управлять ими;

– организация вариабельности структуры системы посредством механизмов вариабельности на уровнях ее реализации (или она будет реализована в момент определения архитектуры, или во время процессов разработки продукта семейства). Соответствующие решения принимаются аналитиками и разработчиками семейства ПП с учетом их опыта, задач домена и механизмов теории обеспечения вариабельности;

– управление вариабельностью связано с ее планированием, контролем и регуляцией в ходе жизненного цикла объектов и системы в целом, которые ориентированы на изменяемость.

Фундаментальную основу концепции моделирования ПС из объектов составляет теории **ООП Буча, Фреге и Геделя-Бернайса** и понятия: классы объектов, инкапсуляция, наследование свойств и операций над элементами класса (экземпляризация, сериализация, агрегация, классификация и др.). Дается описание построения концептуальных моделей ПС и СПП с использованием функций и операций объектного анализа предметной области. Объекты группируются в классы.

Класс - это определенное множество объектов, имеющих общие переменные, структуру и поведение. Объект является экземпляром класса.

Одним из основных свойств объектов является **инкапсуляция**. Она реализуется с помощью интерфейсов, которые состоят из методов и атрибутов. Атрибут может определять внешние переменные экземпляра класса. Для каждой внешней переменной существуют методы выборки значения переменной (get-метод) и присвоение ему новое значение (set-метод). С общей методологической точки зрения, интерфейс экземпляра объекта состоит из переменных методов. Каждый объект может иметь несколько интерфейсов, которые определяют его функциональные свойства. Кроме того, объект может иметь специальный интерфейс, методы которого работают с экземплярами (например, Home-интерфейс в EJB модели EJB языка Java). На абстрактном уровне интерфейс может рассматриваться как частичный вид абстрактного класса, которая сводится к операции приведения классов. Если выполняется операция приведения экземпляра класса в соответствии с определенным интерфейсом, то обязательным условием является реализация этим экземпляром всех методов соответствующего интерфейса.

Теория Г.Буча предлагает стратегию проектирования ПС, исходя из утверждения, что весь материальный мир состоит из объектов. Любая ПрО – это совокупность объектов, связанных между собой некоторым множеством отношений и поведения в течение некоторого времени жизни системы. То есть, <объектная ориентация> = <обьекты> + <наследование>.

Каждое понятие предметной области, вместе с его свойствами и поведением является отдельным объектом, а вся ПС - это совокупность объектов со связями, которые устанавливаются на базе отношений между этими объектами. В качестве объекта выступают как абстрактные образы, так и конкретные физические предметы или группы предметов с указанными общими характеристиками и функциями.

3.3. Электронная наука для представления знаний о задачах систем

Электронная наука ориентирована на проведение современных научных исследований, включая подготовку и проведение экспериментов, сбор данных, распространение результатов, а также их хранение и доступ ко всем полученным материалам. Исследования проводятся методом моделирования, анализа данных и информации для представления накопленных знаний в разных областях. Главными организациями развития Науки (e-science) в Интернет являются:

– Европейский центр по ядерным исследованиям ЦЕРН (European Organization for Nuclear Research, CERN), объединяющий более 120 стран-участниц, более 20 стран-наблюдателей, включая Россию, а также несколько международных организаций (ЮНЕСКО и др.);

– UK e-Science Programme (Великобритания) (http://www.escience-grid.org.uk/) и Core Programme (eSCP), объединяющие более 20 научных центров во главе с Национальным центром электронной науки в Эдинбурге (National e-Science Centre, NSC);

– общественные организации Глобальный форум Grid (GGF, Global Grid Forum), DataGrid и Globus Toolkit (http://www.globus.org);

– Консорциум Европейского союза DEISA (Distributed European Infrastructure for Supercomputing Applications);

– Проекты Tera-Grid, Strategic Grid Computing Initiative и Grid-сети комитета Национального управления по воздухоплаванию и исследованию космического пространства (National Aeronautics and Space Administration, NASA) и министерства обороны США;

– Китайский проект China-Grid;

– Индийский национальный grid-проект GARUDA;

– Проект создания единой компьютерной сети GLORIAD, включая США, Канаду, Европу, Россию, Китай, Южную Корею и др.

На конференции «Е-наука 2014» комитет IEEE дал новый вариант ее определения: «Электронная наука включает в себя то, что называют большие данные, большой адронный коллайдер ЦЕРН, который в год производит около 780 терабайт научных данных для интенсивной обработки в вычислительной биологии, биоинформатике, геномики и в социальных науках…». Главным аспектом электронной науки является научный эксперимент.

Научные эксперименты в электронной науке

В электронной науке для проведения научных экспериментов разработана общая схема (рис.1).

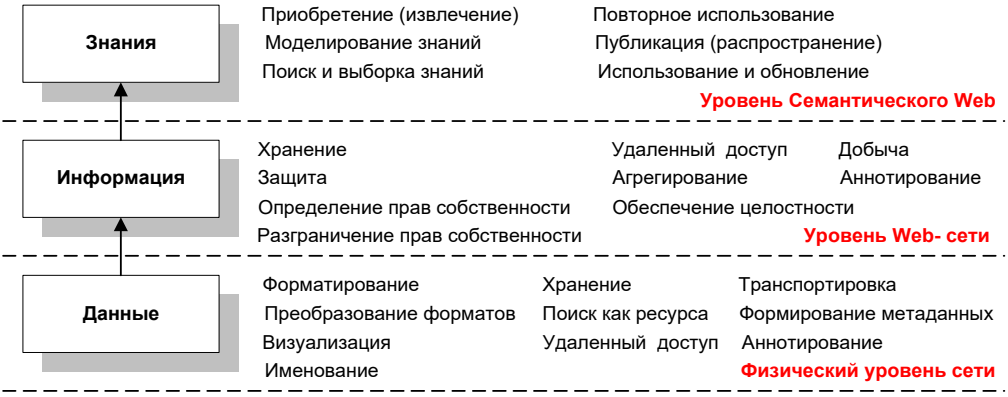


Рис. 1. Три процесса проведения научных экспериментов

Основные элементы этой схемы это:

Знания, как прикладная информация, которая необходима для достижения цели решаемой научной проблемы или определения путей ее решения;

Информация, как данные, которые несут в себе смысл (например, содержимое (контент) документа, метаописание данных, их значения и др.);

данные, как совокупность битов и байтов (например, запись базы данных, документ в файле, memo-поле базы данных, файл изображения и др.), которые не интерпретируются.

Для знания (knowledge) предложен жизненный цикл (рис.2), включающий процессы:

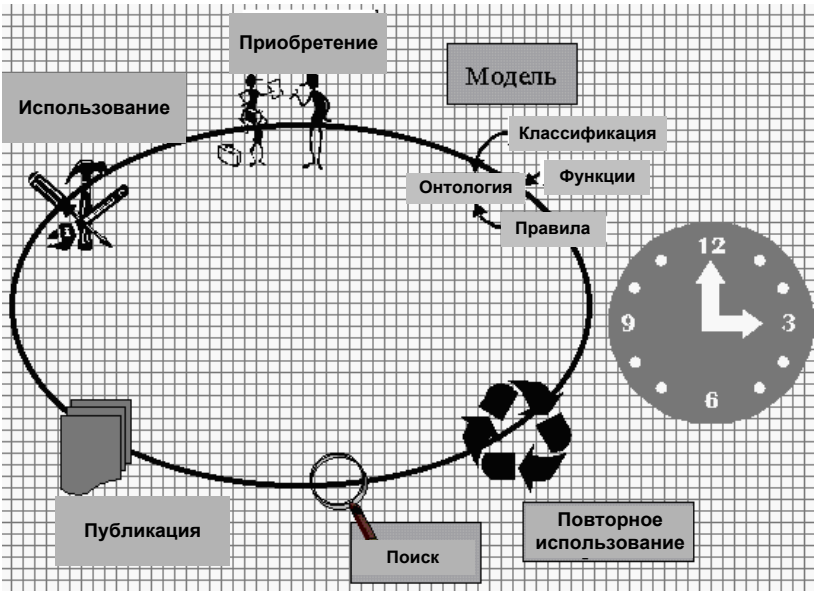


Рис. 2. Модель жизненного цикла знаний

1. ***Приобретение (извлечение) знаний*** (knowledge acquisition). Этот процесс превращает имеющуюся информацию в конкретные знания, которые можно использовать для решения отдельных задач. При этом проводится переработка огромного объема данных и разной избыточной информации. Приобретение знаний заключается в: преобразовании неявных знаний в явные; поиске пробелов в знаниях; извлечения и интеграции знаний из различных источников; изучении распределенных Web-систем; анализе неструктурированной информации – текстов, изображений, экранов и др.

2. ***Моделирование знаний*** (knowledge modelling) состоит в рассмотрении извлеченных знаний и их использования. При моделировании знаний в виде модели, она должна быть пригодной для хранения, накопления и применения знаний при решении конкретных задач.

3. ***Поиск и выборка знаний*** (knowledge retrieval). Накопленные знания используются при эффективном поиске ранее сохраненных знаний и подмножеств контентов, релевантных решаемой задачи.

4. ***Повторное использование знаний*** (knowledge reuse). Выявляемые знания сохраняются в определенном формате в базах знаний для многократного применения в разнообразных контекстах разных предметных областей.

5. ***Публикация (распространение) знаний*** (knowledge publishing). Цель публикации – предоставление знаний в удобной стандартной форме, понятной пользователям.

6. ***Использование и обновление знаний*** (knowledge maintenance). Использование знаний может повлечь за собой глубокий анализ содержания (контента) знаний и требовать его обновления. Контент знаний должен подвергаться верификации и валидации, а также сертификации для сохранения и использования другими.

3.4. Управление знаниями

Для определения целей и задач любой предметной области используется подход, именуемый управлением научными знаниями. Объектами исследования предметной области знаний являются готовые ресурсы сети, описанные в машинной семантике. С помощью машинно-читаемых языков (machine-readable languages) задается взаимодействие и интеграция гетерогенной информации в глобальной информационной сети. Семантика ресурсов глобальной сети, включая сервисы, задается на уровнях исследования:

1. *Верхний уровень* обеспечивает уникальную глобальную *идентификацию* ресурсов, описание *метаданных* о ресурсах средствами *языков описания метаданных* и знаний, которые являются базовыми понятиями *онтологий,* обеспечивают взаимопонимание и задание общепринятого словаря для метаданных и механизмов вывода новых знаний.

2. *Базовый уровень сервиса знаний* включают стандартные умозаключения, целенаправленный просмотр метаданных онтологий, а также дает объяснение выполненным умозаключениям. К этому уровню относятся сервисы:

- поддержки методов извлечения знаний по прецедентам (Case-based reasoning, CBR);
- кластеризации, индексации, уточнения и визуализации больших объемов информации;
- связи одних наборов онтологий с другими и с заданной концептуальной схемой онтологии;
- обеспечения задач умозаключения, касающихся мониторинга, диагностики причин появления исключительных ситуаций и оценивания их завершения.

При обработке знаний сервис формирует *управляемые словари* (controlled vocabularies), в которых отражены области знаний о предметной области и возможности машинных сервисов. Обработка знаний основана на ***логическом выводе*** (inference engines) или на *дескриптивной логике* (Description Logics, DL). Представление знаний о прикладной предметной области и формулировка высказываний осуществляется с помощью логики предикатов первого порядка и средств языков описания онтологии OWL – OWL-DL и OWL-Lite.

Онтология для представления знаний

Понятие онтологии, заимствованное из философии, сейчас используется в искусственном интеллекте и информатике. В искусственном интеллекте онтологии определяются в контексте концептуализации знаний и систем, основанных на знаниях. В некоторых работах онтологии описываются на основе понятий и конструкций логики и математики. К настоящему времени построено много различных онтологий и область их применения увеличивается.

Основу онтологии составляют понятия и термины, которые включаются в словарь терминов. На формальном уровне онтология - это система, состоящая из набора понятий и набора утверждений об этих понятиях, на основе которых можно строить классы, объекты, отношения, функции и теории. Онтология в Web задает *семантику* на уровнях знаний, с помощью которой уточняются, обогащаются и аннотируются контенты глобальной сети. Использование средств онтологии способствует повышению качества коммуникаций между системами, машинами, пользователями и организациями. Эти средства используются для:

- аннотирования прикладных (предметных, проблемных) областей (ПрО), например, моделей ПрО, оформления терминологии в ПрО и др.;
- описания взаимозависимости задач и процессов научных исследований в ПрО;
- аннотирования характеристик качества результатов исследований (качества знаний);
- персонификации (создание моделей для потребителей знаний);
- аргументации поиска знаний и моделирования контента.

Контент предметной области в онтологии, может использоваться в *технологиях знаний*, которые обеспечивают фильтрацию информации и повышение уровня интеллектуальности технологий. Технология знаний поддерживается множеством инструментов, включающих языки и средства представления семантики ПО, язык управления контентами и обмена информацией и знаниями (рис.3).

Для построения онтологий используется инструмент ***Protégé*** (http://protege.stanford.edu/), который поддерживает два основных способа моделирования онтологий – с помощью редакторов Protégé-Frames и Protégé-OWL. Онтологии могут экспортироваться в разные форматы, включая RDF(S), OWL и XML Schema. В основу подмножества языка описания онтологий OWL (Ontology Web Language) и языков OWL-DL и OWL-Lite положена дескриптивная логика.

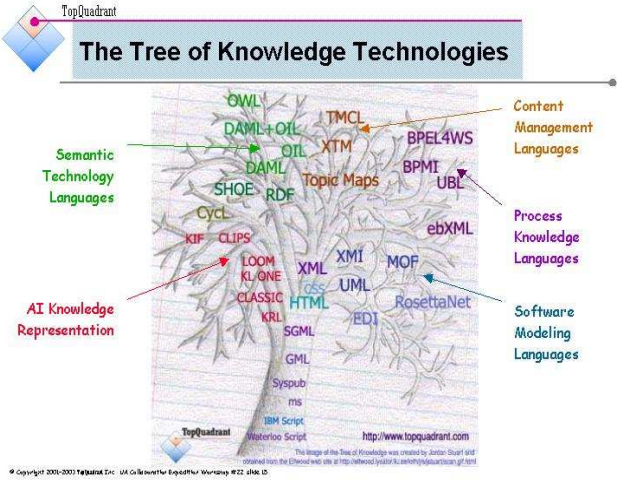


Рис. 3. Дерево технологии знаний

К современным инструментам представления знаний и рассуждений в терминах дескриптивной логики относятся: ***FaCT*** (на LISP), ***FaCT++*** (на C++), ***CEL*** (2005), ***KAON2*** **и** ***Pellet*** (на Java), ***MSPASS*** (на C) и др. Необходимой составляющей инфраструктуры уровня знаний являются ***прикладные сервисы***, ориентированные на коллективное решение задач (problem-solving environments, PSE) научным сообществом.

3.5. Средства создания моделей знаний

Процесс создания и документирования *моделей знаний* задается методологиями, основанными на знаниях (Knowledge-based systems, KBS), в том числе методологией CommonKADS, которая базируется на идее построения библиотек, содержащих элементы решения задач в проблемных областях, являющиеся повторно использованными (reuses) в других областях. Данные инструменты моделирования расширены агентной методологией для проведения мультиагентного анализа и проектирования систем, основанных на знаниях. В этих инструментах определяется множество моделей, охватывающих аспекты процесса разработки *систем управления знаниями*.

К моделям знаний относятся следующие:

– *модель агентов* для спецификации характеристик, определяющих способность агента к рассуждению, сенсорным /эффекторам (sensor/effectors) и сервисам;

– *модель задач*, которые могут выполняться агентами для достижения поставленных целей;

– *организационная модель*, которая задает социальную структуру сообщества агентов;

– *координационная модель,* задающая переговоры (общение) между агентами;

– *коммуникационная модель* для взаимодействия агентов, людей и программных агентов;

– *модель проекта* системы включает операции проектирования, связанные с определением сети агентов, выбора подходящей архитектуры агентов и платформы для разработки новых агентов.

Единую *платформу знаний* на домене (Knowledge Domain) формирует инструмент ***GRACE.*** (http://www.grace-ist.org/). С его помощью задаются ресурсы информации, онтологии и сообщества. Домен знаний представляется в виде концептуального фрейма, содержащего документы описаний, образующих единую структуру представления домена.

GRACE обеспечивает предварительную индексацию документов из многих источников контента с помощью *инженерии онтологии*, описываемой ниже.

Инженерия онтологии

Онтология, как инструмент и средства представления знаний о предметной области, задает онтологическое описание предметной области, путем:

1. Описания основной онтологии в виде запросов, которые систематически передаются множеству источников контента для выбора подходящих документов (эти запросы повторяются периодически, что позволяет обеспечить обновление документов, созданных средствами GRACE);

2. Отбора документов для анализа содержания и связи с различными понятиями онтологии;

3. Просмотра и выполнение запросов с индексами, которые отмечают документы источника контента, из которого они были выбраны.

Для выборки информации из онтологии используется система Grid (в частности, Data Grid). Публикация и распространение знаний поддерживается рядом *систем управления документами* (document management systems), которые используют языки разметки контента для индексирования, поиска и предоставления контента. Некоторые из них используют персонификацию (customization or personalization) контента в системе разработки AKT (Advanced Knowledge Technologies, www.aktors.org). Они позволяют персонифицировать контент и представлять его в соответствии с интересами пользователей онтологии (http://eldora.open.ac.uk/my-planet/).

Для публикации знаний используется ***ePrints*** (www.eprints.org), который предоставляет набор сервисов для архивирования публикации и сопровождения их расширенными метаданными. Это способствует созданию разнообразных сервисных знаний, основанных на методах контентного поиска (content-based retrieval).

В развитие инструментов, поддерживающих создание электронных библиотек, и Grid-технологий среды e-science, в проекте *TextGrid* разработаны ***текстовые решетки Grid***. С их помощью предоставляются средств обработки, анализа, индексирования, аннотирования, редактирования и опубликования текстовых данных для академических исследований.

Для решения проблемы упорядочения текстовой информации используется *система управления терминологией* (System Quirk – SQ, http://www.computing.surrey.ac.uk/ SystemQ). Эта система поддерживает создание и ведение терминов в терминологической базе данных, а также организацию коллекций текстов на компьютере. Система включает в себя целый набор разнообразных инструментов: (*Virtual Corpus*, *KonText*, *Ferret и др.).*

В функции инфраструктуры уровня знаний входит также *поддержка коммуникаций* при совместной (коллективной) работе сообщества ученых. Эти функции реализуются набором средств организации *Web-конференций* и *виртуальных распределенных рабочих пространств* (virtual shared workspace), которые в значительной мере облегчают диалог и коммуникации между отдельными лицами и группами, как, например, среда CAVE.

Один из базовых компонентов, которыми должны оснащаться региональные центры UK (e-Science Regional Centres), – ***сети доступа*** (access grids). Эти сети поддерживают

распределенные заседания, сессии коллективного творчества, семинары, лектории и тренинги. Сеть доступа – это коллекция ресурсов и технологий, которые обеспечивают аудио- и видео-сотрудничество в территориально распределенных сообществах. Архитектура сети доступа включает широкоформатные мультимедийные дисплеи, презентации и интерактивные среды, а также интерфейсы с программным обеспечением промежуточного слоя сети Grid и сред визуализации.

Инструмент поддержки функционирования сети доступа - ***Access Grid®***, текущая версия которого – *Access Grid* 3.1 *beta*1 базируется на стандартизованных Интернет-технологиях и протоколах. При этом все сетевые соединения защищены и сертифицированы.

Жизненный цикл в онтологии

Жизненный цикл онтологии включает следующие действия:

– *Поддержка жизненного цикла знаний*. Базовые концепции и языки проекта Semantic Web, в целом пригодные для того, чтобы специфицировать и поставлять сервисы на информационном уровне. Они являются основой для моделирования *знаний* и выполнения рассуждений с использованием знаний.

– *Достижение доверия к добываемым знаниям*. Проблема доверия к результатам логического вывода связана с обеспечением качества информации и логического вывода.

– *Обеспечение качества информации,* используемой машинами вывода. Информацию необходимо проверять на *корректность* и *полноту* фактов, содержащихся в аннотациях.

– *Обеспечение качества машин логического вывода*. Машины логического вывода (для сервисов базового уровня) зависят от количества и качества аннотаций, найденных в обогащенном контенте. Требуется определить, насколько надежными являются методы логического вывода.

Модель жизненного цикла знаний включает основные, вспомогательные и организационные процессы (аналогично стандарта ISO/IEC Lyfe Cycle 12207). К вспомогательным процессам можно отнести извлечение знаний для построения онтологии, оценивание онтологий, верификацию, интеграцию и документирование онтологии. К организационным процессам относятся процессы планирования построения онтологии и обеспечения гарантии качества онтологий (согласованность, полнота, лаконичность и др.).

3.5.1. Виды онтологий

К видам онтологий для проблемных областей относятся:

– *Онтология предметной области* – объекты, свойства и их взаимосвязи в домене. Было разработано множества аннотаций для медицинских образов, аннотаций для информации о климате, словарях для описания важных характеристик в инженерном проектировании и др.;

– *онтология задач предметной области* - это концептуализация задач и процессов, их взаимозависимости и свойства;

– *онтология программного обеспечения* (готового ПО систем) для повторного использования отдельных элементов ПО, ОМО;

– *онтология сообщества специалистов* и др.

К настоящему времени разработаны стандартные методы и средства *верификации* онтологий, *измерения* (метрик), тестирования и оценивания качества онтологий, а также затрат на построение онтологий.

Задачи построения онтологий

К основным задачам онтологии относятся следующие:

– *Широкомасштабное аннотирование* и обогащение онтологии *существующими* научными данными, информацией и знаниями для различных предметных областей.

– *Разработка сервисов онтологий*. Сервисы предоставляют *согласованные словари* и *концептуальные* научные области (домены) для поддержки аннотирования и извлечения знаний. Концептуальные описания включают понятия и отношения, возникающие между объектами и процессами, представляющими интерес для исследователей в соответствующих областях знаний. Сервисы онтологий управляют *отображением* онтологии, которое используется агентами с различными интересами, а также сервисы онтологий для построения онтологий по XML-схемам.

– *Разработка сервисов персонификации*. Эти сервисы обеспечивают взаимодействие с сервисами аннотирования и сервисами онтологий.

– *Разработка агентов*. При разработке агентов проводится учет (accounting), мобильности агентов и обеспечение безопасности (security). Предотвращение неавторизованного манипулирования на удаленных Grid-узлах, и наоборот, защита Grid-узлов от неавторизованного манипулирования чужими агентами (не ограничивая гибкости Grid).

3.5.2. Web-средства, Grid в E-Sciences для представления областей знаний

Web-средства накладывают определенные ограничения на инфраструктуру и возможности систем электронной науки. Типичная распределенная транзакция предусмотрена для большинства сетевых приложений. Поток информации всегда инициируется клиентом. Если приложение получает информацию спонтанно от сервера HTTP, необходимо включить функциональные возможности сервера HTTP. Типичные сетевые транзакции охватывают намного меньшее число машин, чем того требуют Grid-вычисления. Из-за технических ограничений инфраструктуры глобальной информационной сети семантический Web не может обеспечить пользователей электронной науки (е-ученых) правильной информацией в нужный момент времени. Эта проблема усугубляется все возрастающими объемами информации, которые будут генерироваться в этих системах. В связи с этим возрастает потребность в вовлечении в e-science *Grid-технологий* и развитии семантического Web в направлении *Семантического Grid*. Это даст возможность облегчить гибкое сотрудничество и проводить вычисления в глобальном масштабе с помощью системы Grid.

Сервисы для описания Web-систем

Сервисы Интернет обеспечивают решение разных прикладных и бизнес-задач.

К сервисам относятся:

– общие сервисы системных сред для поддержки процессов и функций обработки программ и данных (например, службы именования, каталогизации и др.);

– сетевые сервисы стандартной модели OSI, моделей SOA (Service-oriented Architecture), SCA (Service-Component Architecture), как инструменты представления и обработки сервисов в сети Интернет, которые реализуют деловые, финансовые, экономические и другие услуги при решении соответствующих задач;

– готовые программные и информационные ресурсы (services, artifacts, reuses, assets и др.), используемые как многоразовые, повторно используемые сервисы для решения разного рода задач в e-science и других прикладных областях.

Некоторые из сервисов стали обязательной частью общесистемных средств (VS.Net, IBM, Intel, Linux и др.), другие используются в специальных областях (например, медицина, биология) в плане предоставления сервисов при работе с современными данными (FDT, GDT, Big Data).

Инструменты поддержки сервисов

Для проведения композиции сервисов предлагается инструмент – ***Jopera for Eclipse*** (http://www.jopera.ethz.ch/). Этот инструмент предоставляет пользователю визуальный язык и гибкую исполнительную платформу для построения распределенных приложений на базе

повторно используемых сервисов, в том числе Web-сервисов. В сети Интернет имеются мощные интегрированные инструменты для:

– визуального определения процесса (потока управления и потока данных);

– облегченной (agile) композиции сервисов с немедленной обратной связью, рефакторингом и регрессионным тестированием процессов;

– рекурсивной композиции сервисов путем автоматического опубликования процессов как Web-сервисов;

– эффективного исполнения процесса с помощью визуальных моделей, которые компилируются в исполнительный код Java;

– визуального мониторинга, интерактивного управления и бесшовной (seamless) отладки композиций сервисов, интегрированных в инструменты проектирования;

– управления изменением интерфейсов сервиса при условии, что JOpera сообщает, каких процессов будут касаться изменения в сервисе;

– трассирования «родословной», т.е. метаданных, содержащих сведения о происхождении данных, которые автоматически собираются движком WF;

– масштабируемого и автономного исполнения процесса с помощью версии JOpera, обеспечивающей запуск кластеризованных компьютеров.

Opera обеспечивает набор Eclipse-плагинов для связи различных программных компонентов, допускает итеративную композицию сервисов, гетерогенность сервисов (включая маршрутизаторы SOAP и RESTful Web-сервисы, Grid-сервисы, Java snipets и др.), а также моделирование и исполнение процесса. Версия *JOpera* 2.1.2 находится по адресу http://www.jopera.ethz.ch/ download/index.

Поиск сервисов

Для онтологического поиска сервисов по их семантическим описаниям используется компонент *Feta*, состоящий из *Feta Client* и *Feta Engine*. *Feta Client* – это GUI-плагин к инструменту Taverna, используемый для описания сервиса, а *Feta Engine* – Web-сервис.

Подключаясь к *Feta Engine*, плагин *Taverna Feta* позволяет:

– конструировать ориентированные на ПрО семантические запросы к нужным сервисам, которые должны отсылаться на *Feta Engine*;

– отображать информацию о результатах выполнения запроса на поиск сервисов;

– интегрировать результирующие сервисы в WF.

Критериями поиска сервисов являются:

– сервис, на входе которого приемлем элемент семантического или общего типа X;

– сервис, который производит на выходе элемент семантического типа Y;

– сервис, который решает задачу X или еще более конкретную;

– сервис, который использует метод X или еще более конкретный;

– сервис определенного типа, например, процессор WSDL и др.

Фрагмент формы для составления запросов на поиск сервисов в среде Taverna Workbench представлен на рис. 4. Пример отображения результатов поиска задан на рис. 5.

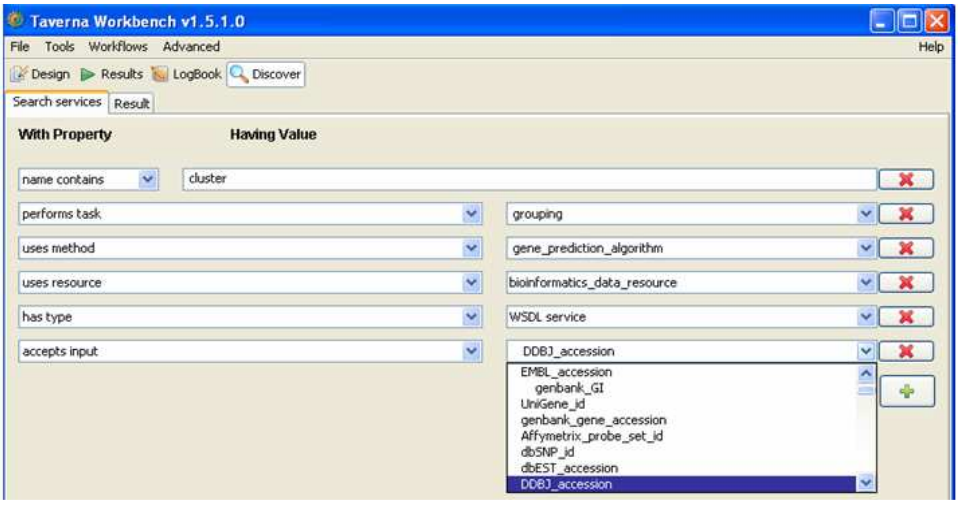


Рис. 4. Форма для задания критериев онтологического поиска сервисов

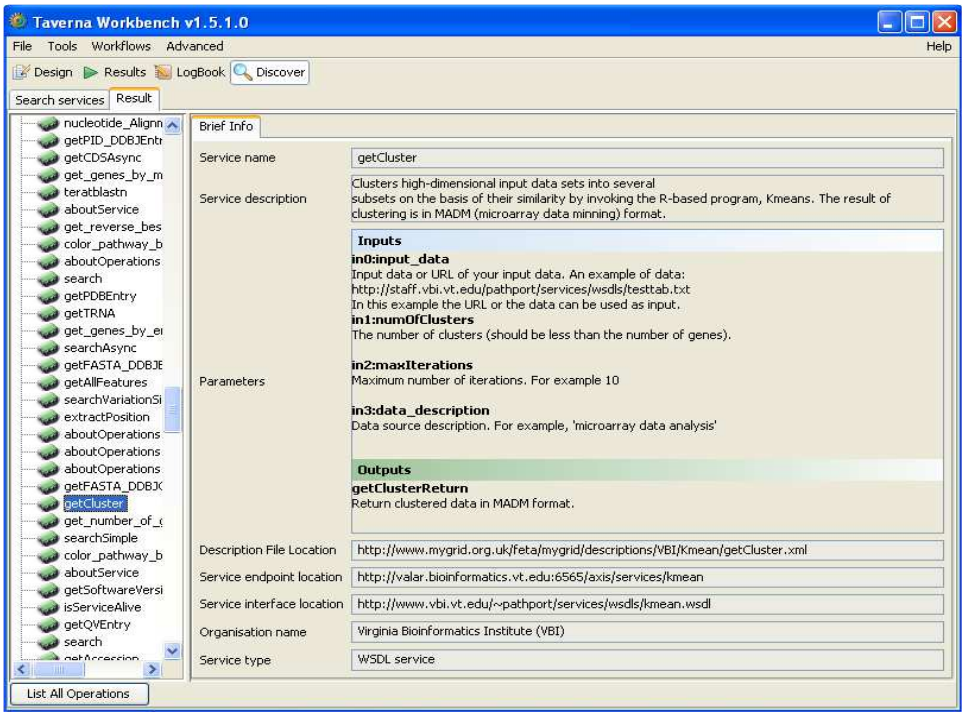


Рис. 5. Результат онтологического поиска сервиса по критерию

3.5.3. Web-сервисы OGSA - Grid и Etics

Система Grid обеспечивает распределенные вычисления, в которой «виртуальный суперкомпьютер» представлен в виде кластеров, соединённых с помощью сети слабосвязанных гетерогенных компьютеров, работающих вместе для выполнения огромного количества заданий (операций). Для сетевых и распределенных вычислений используется система BOINC (Berkeley Open Infrastructure for Network Computing), разработанная в университете Беркли. Данный инструмент используется:

– в астрофизике, гравитационной физике, физике высоких энергий, физике нейтрино и ядерной физике;

– молекулярной динамике, информатике и вычислительной технике, нанотехнологии;

– структурной биологии, вычислительной биологии, геномике, протеомике и медицине.

OGSA (Basic Execution Service) определяет интерфейс к сервисам, которые инициируют вычислительные процессы в Grid, отслеживают и управляет вычислительной активностью и моделями жизненного цикла (состоянию) процессов, а также информационными модели вычислительного процесса. Ws-naming определяет схему для ссылки на ресурсы в Grid на абстрактном уровне, независимо от их физического расположения. Ws-dissemination определяет сервисы для распространения сообщений в Grid (рис.6) среде.

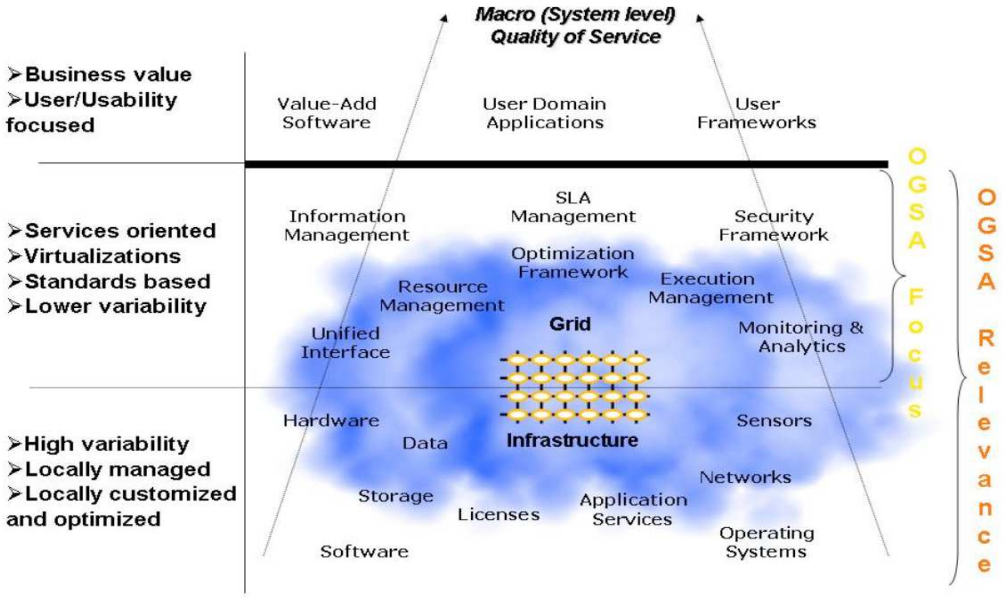


Рис.6. Схема Grid-системы

Web-сервисы *не имеют состояния*, и срок их жизни определяется поставщиком услуги, то есть они существуют независимо от потребителя.

Grid-сервисы имеют состояние и срок жизни, ограниченный *потребностями потребителя*. Они «рождаются» на серверах в Grid-сети для решения конкретной задачи и «исчезают» по ее завершении. Представляют «свой» сервер не как что-то, реально оказывающее услуги, а как *фабрику*, на которой по заказу клиента создается нужный сервис, используемый в заданное время. Этот подход получил название «фабрики образцов».

Если возникает необходимость решить какую-то математическую задачу в Grid-сети, под нее создается виртуальный компьютер, предоставляющий нужный *физический сервер*. Этот сервер может поддерживать множество подобных *виртуальных вычислительных машин*, создаваемых под требования клиента и уничтожаемых, когда в них отпадает необходимость.

Для представления спецификаций Grid-сервисов и других ресурсов используются ***WSRF*** (*Web Services Resource Framework*) (www.globus.org/wsrf). С помощью средств спецификации проводится моделирование и управление состоянием контекста Web-

сервисов. Это дает возможность контролировать и изменять состояние доступных сервисных ресурсов (*WS-Resource*).

Описание Grid-архитектура для моделирования прикладных систем

Grid-архитектура представляет собой архитектуру взаимодействующих *протоколов*, *сервисов* и *интерфейсов*. Эти элементы сохраняются в репозитории. Ими управляет брокер сервисов. С помощью сервисов и ресурсов устанавливается соединение с Grid-системой для вычисления различного рода задач. Для обработки ресурсов выполняются протоколы следующих уровней (рис. 7):

Уровень ресурсов Grid содержит физические ресурсы, к которым нужно предоставлять совместный доступ (ресурсы вычислений (процессоры), ресурсы памяти, сетевые ресурсы). Физические ресурсы обладают всеми характеристиками оборудования Grid – многочисленность, гетерогенность конфигурации и разнообразие политик использования. На каждом месте размещения ресурсов ими управляет *программное обеспечение управления ресурсом* (resource management software (RMS)), которое решает задачи управления – планирование, распределение, контроль *состояния ресурсов* и доступ пользователям.

Уровень услуг (сервисов) ресурсов Grid предоставляет сервис планирования ресурса и вычислений. В первом случае ресурсы предоставляются на определенный период времени. Во втором случае сервисы выполняют интенсивные вычисления и возвращают результаты. Для получения ресурсов Grid работает брокер ресурсов.

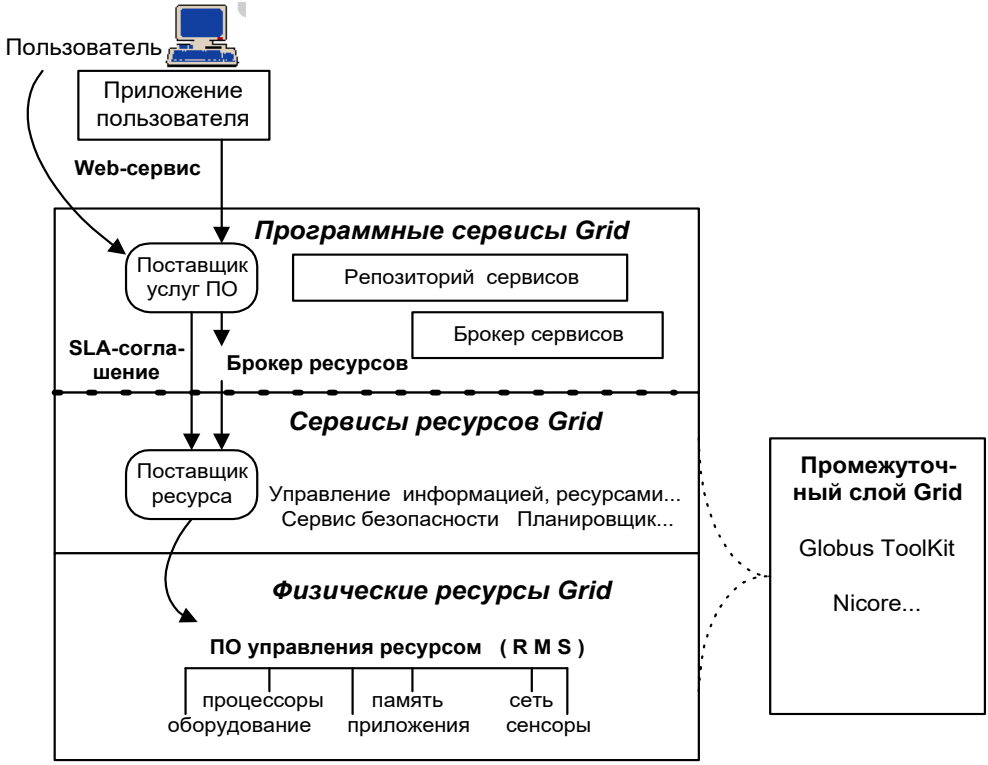


Рис. 7. Архитектура Grid

Grid обеспечивает взаимодействие протоколов, сервисов и интерфейс*ов*, определяющие базовые механизмы и устанавливают соединения с Grid-системой для совместного использования ресурсов. *Архитектура протоколов Grid* включает уровни (рис. 8), смысл которых дан ниже.

Базовый уровень (Fabric Layer) описывает сервисы, непосредственно работающие с ресурсами, ресурсами памяти и информационными ресурсами.

Сетевой ресурс – связующее звено между распределенными ресурсами Grid. Его основная характеристика - скорость передачи данных.

Связывающий уровень (Connectivity Layer) определяет коммуникационные протоколы и протоколы аутентификации. *Коммуникационные протоколы* обеспечивают обмен данными между компонентами базового уровня (ресурсами). *Протоколы аутентификации* основываются на коммуникационных протоколах и предоставляют криптографические механизмы для идентификации и проверки подлинности пользователей и ресурсов.

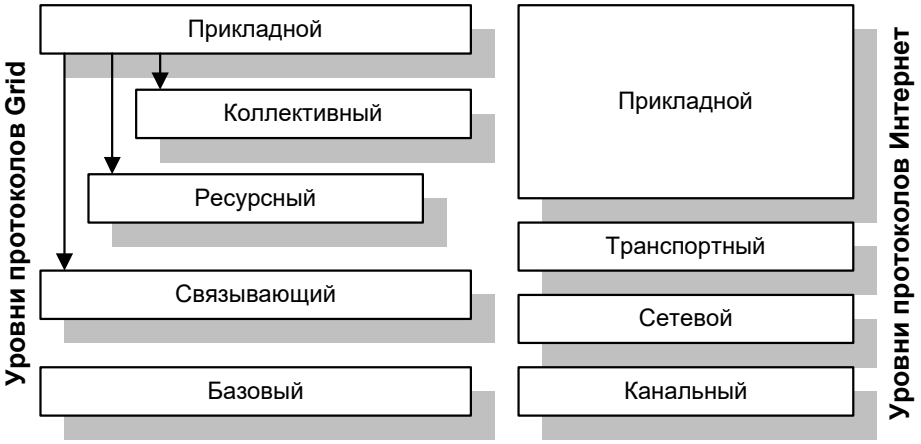


Рис. 8. Уровни архитектуры протоколов Grid и Интернет

Ресурсный уровень (Resource Layer) работает над протоколами коммуникации и аутентификации уровня в архитектуре Grid. Этот уровень реализует инициацию, мониторинг, контроль ресурсов и взаимодействие ресурсов через *унифицированный интерфейс.*

Коллективный уровень (Collective Layer) отвечает за *глобальную интеграцию различных наборов ресурсов.*

3.5.4. Инструменты разработки Веб- систем средствами Grid

Для разработки Grid-систем используются следующие инструменты:

Legion (legion.virginia.edu) обеспечивает объектно-ориентированную модель инфраструктуры, где все ресурсы описаны в виде объектов, и предоставляет систему единой целостной виртуальной машины;

Globus Toolkit (GGF) обеспечивает сервисно-ориентированную слоистую архитектуру, в которой глобальные сервисы строятся на основе базовых сервисов нижележащих уровней;

Condor (www.cs.wisc.edu/condor) – открытый проект, разработанный для выполнения заданий с интенсивными вычислениями, не требующими вмешательства человека. Задание специфицируется в файле конфигурации и его выполнение распределяется по множеству узлов Grid-сети. Данный инструмент использует *Globus Toolkit* для управления заданиями;

WebFlow (http://www.npac.syr.edu/users/haupt/WebFlow/) обеспечивает создание целостной оболочки с возможностью публикации и многократно использования вычислительных модулей, чтобы пользователи могли работать с Web-браузером в конструирования распределенных приложений и с редактором в качестве визуального инструментального средств. Нижний (серверный) слой системы реализован с помощью *Globus Toolkit*;

Gridbus Data Grid Service Broker развивает модель брокера ресурсов вычислительного Grid, в плане использования распределенных сетей, ориентированных на данные; динамическую параметризацию для доступа к репозиториям и оптимизации передачи данных;

GRACE (Grid Architecture for Computational Economy) – развитие *Nimrod/G*. Инструмент обеспечивает динамическое сотрудничество с владельцами ресурса для выбора тех ресурсов, которые предлагают *оптимальную стоимость* использования или критериям оптимального времени;

Grid-порталы – позволяют обращаться к ресурсам, определенным для конкретной ПрО, через Web-интерфейс, и обеспечивают доступ к Grid-ресурсам. Доступ к Grid-порталу может также индивидуализироваться при помощи профилей, которые создаются и сохраняются для каждого пользователя (например, портал *HotPage*);

Grid Port Toolkit – портальный набор инструментов многократного использования, основанный на инфраструктуре *HotPage*. В нем два компонента: *GridPort* – сервисы Grid-портала и прикладной программный интерфейс (API) приложений. Модуль Web-портала выполняется на сервере сети и обеспечивает безопасное подключение к Grid. API приложений обеспечивает Web-интерфейс, который помогает конечным пользователям разрабатывать специализированные порталы (не требуя знаний об основной портальной инфраструктуре). *GridPort* спроектирован так, чтобы позволить выполнение портальных сервисов и приложений клиентов на отдельных Web-серверах. Портальная архитектура основана на трехслойный модели, где браузер клиента надежно подключается к Web-серверу по безопасному соединению через двунаправленный канал (по протоколу HTTP). Web-сервер может получить доступ к различным Grid-сервисам, используя инфраструктуру *Globus.*

Заключение

В отчетном материале по проекту РФФИ 352 -2017 приведены определение, цели и задачи электронной науки, предназначенной для проведения научных экспериментов. Основу этой науки составляют знания, информация и онтология. Приведена схема процесса представления знаний для любой предметной области. Описана структура моделей знаний, онтологии, инженерии концептуализации знаний и задач исследования научных знаний в разных научных областях. Рассмотрены Web-средства и инструменты для создания из готовых ресурсов и сервисов Grid-систем. Приведена архитектура Grid–системы (репозиторий ресурсов, брокер и др.), стандарт протоколов обработки сообщений при работе и выполнении инструментов в Web-системе.

Литература к 3.1-3.5 раздела 2 РФФИ-352- 2017

1. Bohle, S. What is E-science and How Should it Be Managed? http://www.scilogs.com/scientific_and_medical_libraries/what-is-e-science-and-how-should-it-be-managed/
2. IEEE International Conference on e-Science, homepage, accessed December 18, 2014, https://escience-conference.org/
3. Science 2.0: новая научная парадигма или развитие инструментария исследовательской работы .- А.С.Милославов //Сборник научных статей XVIII Объединенной конференции «Интернет и современное общество» IMS-2015, Санкт-Петербург, 23-25 июня 2015.
4. European organization for nuclear research CERN. http://public.web.cern.ch/public/en/research/DataAnalysis-en.html
5. OWL Web Ontology Language. W3C Working Draft 29 July 2002. Latest version is available at http://www.w3.org/
6. OWL-язык Веб-онтологий. http://sherdim.ru/pts/semantic_web/REC-owl-features-20040210_ru.html

7. De Roure D., Jennings N. R., Shadbolt N. R. The Semantic Grid: Past, Present and Future e-Science Infrastructure. http://www.semanticgrid.org/documents/semgrid-journal/semgrid-journal.pdf.

8. Semantic Web, Representation of data on the World Wide Web, based on the RDF standards, http://www.w3.org/2001/sw/

9. TopQuadrant Technology Briefing. Semantic Technology. Version 1.2 (march 2004). http://www.topquadrant.com/documents/TQ04_Semantic_Technology_Briefing.PDF

10. Semantic Web programming//S. Hebeler, M. Fisher, R.Blace, A/Peter-Lopes, Willey Publiching Inc., 2008. http://semwebprogramming.org.

11. Goble C., Bechhofer S. OntoGrid: A Semantic Grid Reference Architecture. http://www.ctwatch.org/quarterly/articles/2005/11/ontogrid-a-semantic-grid-reference-architecture

12. W3C OWL Working Group OWL2 Web Ontology Language, W3C.2009 http://www.w3.org/TR/owl2-overview/

13. Демичев А. П., Ильин В. А., Крюков А. П., Введение в грид-технологии. Препринт НИИЯФ МГУ, 2007, 11/832 2007, http://www.sinp.msu.ru.

14. Лаврищева Е. М., Карпов Л. Е., Томилин А. Н., Семантические ресурсы для разработки онтологии научной и инженерной предметных областей, Доклад на конференции "Научный сервис в сети Интернет" 19-24 сентября 2016, Абрау.

15. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition) W3C Recommendation 27 April 2007. http://www.w3.org/TR/SOAP12-part1/

16. Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001 http://www.w3.org/TR/wsdl

17. W3C. RDF Current Status. http://www.w3.org/standards/techs/rdf

18. SPARQL Query Language for RDF W3C. http://www.w3.org/TR/rdf-sparql-query/

19. W3C Annotea Project. http://www.w3.org/2001/Annotea/

20. Reference Model for Service Oriented Architecture 1.0. http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html

21. Web Services Resource 1.2 (WS-Resource). http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf

22. Web Services Resource Properties 1.2 (WS-ResourceProperties)
http://docs.oasis- open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf,
http://docs.oasis-open.org/dita/v1.2/spec/DITA1.2-pec.html

Научные работы по реализации задач проекта РФФИ 352- 2017г.

1. **Лаврищева Е.М.** Научные основы программ и технологии программирования систем.-15.03.2017.Российская Академия наук, Центральный дом учёных. Презентация - http://www.ispras.ru/publications/2017/the_scientific_foundations_of_programs_and_technology_programming_systems/

2. **Лаврищева Е.М.** Развитие теории программ и систем в СССР. История и современные теории программ и систем.- Сборник SORUCOM-2017.- Развитие ВТ в России и в странах бывшего СССР. История и перспективы.- 3-5 октября 2017.-с.162-176.

Аннотация. Приведен анализ теории программ советских ученых (Ляпунова, Ершова, Янова, Глушкова, Ющенко, Липаева и др.). Определена сущность теории программ, технологии программирования, синтеза, сборки и доказательства программ и систем (1963-1990). Представлены элементы теории инженерии программных продуктов - Software Engineering (1980-2016) и новые перспективные теории и методы моделирования изменяемых систем из готовых программных ресурсов (объектов, компонентов, сервисов и др.), Определена конфигурационная сборка ресурсов по моделям в варианты выходного кода продуктов и систем и процесс верификации моделей систем и модели характеристик MF, а также теория сборки готовых ресурсов в систему по верифицированным моделям и аппарата

алгебраических систем с механизмами трансформации передаваемых данных между объединяемыми ресурсами через интерфейс. Предложен подход к тестированию отдельных ресурсов систем и создания конфигурационного выходного файла.

Ключевые слова: Теория, методы, научные основы, программные системы, построение, верификация, доказательство, моделирование, интероперабельность.

3. **Лаврищева Е.М**. Научные основы программной инженерии (Понятия, Парадигмы, Технологии, CASE-средства. Theoretical and Applied Aspects of Program Systems Development» (TAAPSD'2017), 4-8 дек. 2017. - с. 201-214.

Аннотация. Определяются научные понятия и фундаментальные основы Software Engineering (SE). Базовые понятия - это объекты, модули, программы, системы; процессы разработки объектов. Фундаментальную основу SE составляют: метод сборки модулей; дисциплины SE (научная, инженерная, экономическая, управленческая и др.); парадигмы программирования модулей, объектов, компонентов и др.; жизненный цикл (стандарт ISO/IEC Life Cycle 12207); линии изготовления систем, фабрики программ и AppFabric; новая логико-математическая теория объектно-компонентного моделирования (ОКМ) систем; верификация и тестирование изготовленных систем. Инструменты поддержки научных и фундаментальных основ на сайте http://7dragons.ru/ru.

Ключевые слова: наука, концепция, формализмы модулей, метод сборки, теория, ОКМ, ОСМ, Product Line, Life Cycle, model FM, configuration, verification/ realization.

4. Новые аспекты моделирования программных и операционных систем, включая конфигурационную сборку, тестирование и оценку качественных показателей продуктов, созданных из программных ресурсов**:**

Чарнецки К., Азенакер У. Порождающее программирование. Методы, инструменты, применение. – Изд. Дом №Питер» - М.: СПб. Харьков-Минск. 2005. 730 с.

Лаврищева Е.М., Грищенко В.Н. Методы и средства объектно-компонентного программирования. Кибернетика и системный анализ. 2003. №1, с. 39-55.

5. **Е.М. Лаврищева, В.С. Мутили, А.Г. Рыжов.** Аспекты моделирования вариабельных программных и операционных систем. Сб. XIX Всероссийский научной конференции «Научный сервис в сети Интернет», Новороссийск, 18-23 сентября 2017.-327-341.

Аннотация. Рассматриваются подходы к заданию вариабельности программных и операционных систем (ПС), и их семейств (СПС). Предложена модель характеристик (Feature Model), которая включает базовые элементы для ПС и структура ядра OS Linux, на основе которой формируется вариант модели ОС из системных и функциональных элементов ядра ОС. Предложена модель конфигурационной сборки элементов ПС и OS Linux. Определена концепция задания модели вариабельности для OS Linux, основанные на графовой структуре системы из готовых ресурсов (ГОР) – функциональных и интерфейсных элементов для ПС, а также системных и функциональных элементов ядра OS Linux, с помощью которых генерируются варианты ОС. Определен подход к управлению вариабельностью и конфигурационной сборкой ОС с помощью объектно-компонентного метода (ОКМ) для получения вариантов выходного продукта. Предложен подход к тестированию вариантов ОС.

Ключевые слова: программная система, операционная система, семейство программных систем, вариабельность, функциональный и интерфейсный элемент, модель характеристик, управление конфигурацией, верификация, тестирование.

6. **E.M. Lavrischeva, V.S. Mutilin, A.G. Ryzhov**. Designing variability models for software, operating systems and their families, 2017, Сборник ИСП РАН, 2017, issue 5, 2017, pp…Doi: 10.15514/ISPRAS-2017(5).

Abstract. The complexity of existing Legacy systems and the difficulty of amending it led to the formation of the new concept of variability of systems specified by a model of the characteristics of MF (Feature Model). The approaches to formal definition of a model MF and

creating on its basis modified systems program (PS), operating systems (OS) and families (SFS) for PS, OS are discussed. Approaches are methods of manufacture of PS in the Product Family/Product Lines, the Czarnecki conveyor Assembly of finished artifacts in the space of problems and solutions, logical-mathematical modelling PS from the functional and interface objects by OCM, extraction of the functional elements from core OS Linux to MF for the generation of new variants of the OS are given. It was discussed approaches for formalization of variability of Legacy systems, new PS and their SFS. The new concept of management of variability systems with help OCM are defined. The approach to verify models of the MF, PS, SFS and OS and to configurate of functional and interface objects for obtain the variants of the output product are proposed. The characteristic of the process testing of variants of the PS, OS and SFS are done. OCM is realized in ITK of site http://7draguns/ru/ru.

Keywords: variability model, software systems, family of systems, configuration, variant, functional, interface element, requirement, management.

7. **Лаврищева Е.М.** Фундаментальные основы программной инженерии. Сборник научных трудов 5-международной конференции «Актуальные проблемы системной и программной инженерии». - Сборник трудов АПСПИ-2017, 14-16 октября 2017.-с.163-177.

Аннотация. Определены базовые понятия и фундаментальные основы Software Engineering (SE). Базовые понятия – это объекты, модули, программы, системы; процессы разработки объектов. Фундаментальную основу SE составляют: метод сборки (конфигурирования) модулей; дисциплины SE (научная, инженерная, экономическая, управленческая и др.); парадигмы программирования модулей, объектов, компонентов и др.; жизненный цикл (стандарт ISO/IEC Life Cycle 12207); линии изготовления систем; фабрики программ и AppFab; логико-математическая теория объектно-компонентного моделирования (ОКМ) изменяемых систем; технология тестирования систем и оценивания качественных показателей.

8. Подход к созданию Веб-приложений и систем с помощью Веб-сервисов

Е. М. Лаврищева, Л. Е. Карпов, А. Н. Томилин. Подходы к представлению научных знаний в Интернет науке. Сб. XIX Всероссийский научной конференции «Научный сервис в сети Интернет», Новороссийск, 18-23 сентября 2017.-с.310-326.

Аннотация. Рассматриваются подходы к формированию вариабельности действующих и создаваемых программных систем (ПС), и их семейств (СПС). Предложенная модель характеристик (Feature Model) K. Pohl в проекте SEI Product Line и Product Family, а также модель конвейерной сборки готовых артефактов в пространстве задач и решений K. Czarnecki послужили исходной концепцией для создания модели вариабельности для программных и операционных систем (ОС). Определены новые модели ПС и ОС, включающие готовые ресурсы (ГОР) – функциональные и интерфейсные элементы, а также модель характеристик (МХ) элементов, по которым генерируются варианты ПС из их семейств. Определена концепция управления вариабельностью и конфигурационной сборкой ПС из ГОР в объектно-компонентном методе (ОКМ) для получения вариантов выходного продукта. Предложен подход к тестированию вариантов ПС.

9. **Лаврищева Е.М., Рыжов А.Г.** Применение теории общих типов данных стандарта ISO/IEC 12207 GDT к Big Data", «Актуальные проблемы в современной науке и пути их решения», 27 декабря 2016, http://euroasia-science.ru

Аннотация. Рассматриваются вопросы применения теории общих типов данных (GDT) к Большим Данным (Big Data). Подан базовый аппарат представления типов данных (ТД) GDT (стандарта ISO/IEC 11404-2007) и теория генерации нестандартных ТД GDT. Определены операции и набор функций трансформации данных с одной платформы на другую. Предложен подход к анализу неструктурированных данных и генерации с помощью функций, аналогичных функциям библиотеки CTS (Common Type System) VS.Net.

Ключевые слова: общие типы данных; простые, сложные, генерируемые, неструктурные данные; трансформация; генерация; большие данные; библиотеки функций.

Основные результаты по проекту РФФИ-352 в 2017 году

Рассмотрены базовые концепции моделирования вариабельных ПС и СПС. Приведены оригинальные решения К. Похла по модели вариабельности МХ в SPLE, К. Чарнецки в пространстве проблем и решений в плане генерации ГОР и в методе ОКМ для проектирования графовой модели СПС из функциональных и интерфейсных элементов и проведения сборки ресурсов. Разработан подход к теории определения модели СПС из ГОР и ОС Linux с использованием модели вариабельности МХ. Предложена модельная среда и управление вариабельностью СПС с учетом заданных требований. Дано расширение модели вариабельности в интегрированной модели для оценки вариантов систем с учетом требований к ПС, планирования средств и сроков выполнения оценочной модели вариабельности. Определены методы верификации, тестирования и выполнения конфигураций ПС и СПС.

Раздел 3. Форма 503. Проект РФФИ 16-01-00352 (2018). Теория и методы моделирования сложных прикладных систем и обработки данных

Карпов Л.Е. Лаврищева Е.М., Томилин А.Н., Рыжов А.Г.

Анотация. Излагается теория моделировании сложных программных, операционных и Веб-систем из готовых ресурсов - ГОР (reuses, модулей, объектов, компонентов, сервисов и др.). Определены модели таких систем Msys и показана реализация метода конфигурационной сборки программных, ОС и Веб-систем (services-component architecture) из ГОР, основанных на моделях SOA (services-oriented architecture), SCA и используемых при создании веб-систем в среде Семантик Веб Интернет и IBMSphere. Определены общие и внутренние характеристик ГОР, задаваемые в ЯП, их интерфейсы в IDL/WSDL, а также проведены процессы верификации моделей для класса Веб-систем и OS Linux. Проанализированы операции сборки link/make в системах BSD, IBMSphere, .Net и config стандарта ISO/IEC 826-92-2012 (configuration) из ГОР для систем. Реализован метод моделирования ЖЦ стандарта ISO/IEC 12207-12207 средствами онтологии Semantic Web и получен патент (2018). Отработан процесс тестирование ГОР, систем из них и сервисных компонентов. Проведена оценка качества систем из системных сервисов и сервисных компонентов. Многие результаты исследований и реализаций данного проекта обсуждались в докладах на конференциях Всероссийских (Научный сервис - 16, 17, 18), (ISPRAS OPEN-16, 17, 18) и др., и Международных (Science and Information -2015 – Онтология доменов), (Future Technology - 2016, 2018 – отечественные технологии программирования). Доклады опубликованы в соответствующих публикациях этих конференций и в журналах Scopus/ Web-sciences и др.

THEORY AND METHODS FOR DEVELOPING VARIABLE SOFTWARE SYSTEMS

Annotation. The theory of object and component modeling of complex variability software and Web systems using ready-made resources - RMR (reuses, modules, objects, components, services, etc.) is worked out. The models of MF variability, systems Msys, a component model (CM) and shows an implementation of the method of Assembly/Building of various software and Web systems of RMR, including the service component elements for Web systems environments IBMSphere SOA, SCA and Semantic Web. The general and internal characteristics of RMR, their representation in PL, interfaces in IDL/WSDL are defined, and also verification of these models is given. The operations of link/make programs Assembly in BSD, IBMSphere, .Net and config of ISO/IEC 826-92-2012 (configuration) from RMR for web systems are analyzed. Implemented method of modeling a domain of life cycle ISO/IEC 12207-12207 by means of ontology and Semantic Web by patent. The process of testing according to the patent for systems of RMR and service components has been worked out. The assessment of quality of systems from RMR and service components is carried out.Many of the results of research and implementation of this project were discussed in reports at conferences of all - Russian (Scientific service - 16, 17, 18), (ISPRAS OPEN - 16, 17, 18) and International (Science and Information – 2015, 2016), (Future Technology - 2016, 2018). The reports are published in the relevant publications of these conferences and in the journals Scopus/Web-sciences, etc.

В данном разделе определяется теория и методы моделирования сложных систем (программных, информационных и операционных) из накопленных готовых системных ресурсов - ГОР (модулей, объектов, компонентов, сервисов и др. артефактов). Теория основывается на объектном моделировании прикладных систем и средствах обеспечения вариабельности (изменяемости) систем с учетом принятой в международном сообществе модели MF (Model Feature) и моделей систем (Msys) создаваемых прикладных и действующих Legacy-systems (таких как OS Linux, IBM, MS, Intel и др.). Методы должны обеспечить: верификацию моделей, используемых готовых компонентов повторного использования (КПИ, типа reuses) или ГОР; тестирование КПИ, ГОР и их сборку (конфигурационную сборку) по моделям MF и Msys; обработку данных (в том числе и Big Data), используемых в системах с учетом стандарта ISO/IEC 11404 – General Data Types, 2007. Отработанные теории, методы и средства применены при создании технологии программирования веб-систем и варианта ядра OS Linux в среде Semantic Web Internet.

1.1. Основные задачи проекта 3

1. Анализ отечественных теоретических основ программ и систем с целью поднятия теоретического уровня подходов к разработке и моделированию новых сложных систем в нашей стране. Анализ начинается с теории программ и технологий, разработанных первыми учеными СССР (А.А. Ляпунова, А.П. Ершова, Ю.И. Янова, С..Лавров, В.М. Глушкова и др.) в начальный период появления первых отечественных ЭВМ (МЭСМ, БЭСМ, и др.) и послуживших формированию начальной теории программ, методам синтеза и сборки программ из разноязычных модулей в языках программирования - ЯП (ALGOL, COBOL, FORTPAN, PL-1 MODULA-2 и др.), которые использовались вплоть до 1991г., Дальнейшим развитием теории программ и систем программирования является теория ООП Г. Буча (1987), внесшая математические понятия (множество, классы, наследование, полиморфизм и др.) программирования и вошедшая во все ЯП, как основа теории моделирования систем из объектов (UML) и формирования логико-математического аппарата графового моделирования систем и вариабельных моделей MF (Feature Model), а также обеспечения трансформации созданных ГОР и создаваемых программных, операционных систем (ОС) и работ по обеспечению взаимодействия разных систем в современных средах [1-7].

2. Апробация подходов к верификации моделей создаваемых систем, конфигурированию используемых ГОР, компонентов и интерфейсов ГОР, к тестированию отдельных ресурсов и систем из ГОР и модификации систем по MF (добавление, удаление, создание новых ГОР);

3. Разработка варианта ОС семейства Linux путем извлечения (Variability Mining) системных и функциональных модулей с целью определения их как базовых элементов для построения моделей MF, OS Linux и генерации по ним вариантов ядра ОС для применения в экономике, промышленности, медицины и др. Эти модели и элементы ГОР верифицированы, а интерфейсы протестированы. Описана конфигурационная сборка варианта OS Linux.

4. Разработан метод преобразования типов данных компонентов, задаваемых в их интерфейсах, с целью обеспечения взаимодействия отдельных ГОР, систем и семейств систем в современных средах.

5. Применен компонентный подход к созданию Веб-систем из сервисов и сервисных компонентов SOA, SCA, SCM Интернет на основе новых языков W3C.

6. Описана конфигурационная сборка вариантов систем, базисом которой является технология сборочного программирования разнородных модулей с интерфейсами в сложные комплексы программ специального и общего назначения создана в рамках ВПК под руководством Глушкова В,М. и Липаева В.В. в 70-80 годах прошлого столетия. Эта технология создана впервые в истории программирования СССР. Метод сборки модулей реализован в ОС ЕС (IBM-360) и использован в Sun Microsystems, .Net, MS.VS и др. Модули описывались в языках программирования (ЯП) (Algol-60, Fortran, Cobol, Modula и др.), их

интерфейсы описывались в языке MIL (Module Interface Language), а сборка модулей задавалась оператором link (M_1, M_2). Метод сборки с интерфейсными примитивами преобразования разнородных данных ЯП опубликован в книге «Связь разноязыковых модулей ОС ЕС (м: Финансы и статистика, 1982, 137с.). Она использовалась многими пользователями и внедрена в 52 организации страны. По словам А.П.Ершова «сборочное программирование обеспечивает построение уже существующих (проверенных на правильность) готовых отдельных фрагментов программ (типа reuses) в сложную структуру». См. Работы:

1. Ершов А.П. Научные основы доказательного программирования.– Научное сообщение в президиуме АН СССР, Наука, 1984.–с.1–11. а звание академика СССР.

2. Лаврищева Е.М. Грищенко В.Н. Сборочное программирование.1991.- 219 с.

3. Липаев В.В. Позин Б.А., Штрик А.А.Технология сборочного программирования.- Москва, 1992.-281с.

После развала СССР в 1992 был создан проект информатизации России с участием 30 ученых страны. В этом проекте разработаны экономические, технические и научные концепции информатизации России. К научным направлениям программирования отнесены алгебраические и логические теории формализации параллельных и распределенных систем; формальные спецификации и верификация сложных систем; парадигмы программирования; формализмы представления знаний и модели интерфейсов и визуализации. К основным направлениям инженерии ПО отнесены: методы разработки, безопасности, надежности защиты, качества и стандартизация ЖЦ систем.

В рамках программы информатизации РФ появились новые ЯП (С, С++, Basic, Ruby, Python и др.) для программирования в них программ и систем, а также стандартные языки описания интерфейсов IDL, API, WSDL и др. Разработанный ранее метод сборки продолжал использовался для новых ЯП с помощью операций сборки link в системах IBM, make в BSD (1996) и config, building в Grid, SPAROL (2000) и др. Оператор make как и link ранее собирал исполняемые модули из библиотек Filemake и задавал им порядок связей модулей в новых средах ОС Linux, Unix и др.. Оператор config стандарта (IEEE 828-96-2012 Configuration, 2009) стал главным в международном Европейском проекте Grid (http://www.semanticgrid.org/documents/semgrid2004/) с системой ETICS, основанной на методах управления процессами разработки информационных и программных систем из КПИ из системных и сервисных ресурсов (см. статьи Лаврищева Е.М. Теория объектно-компонентного моделирования программных систем. Препринт ИСП РАН, № 29, 2016 г. - М.: 48 с. ISBN 078-5-91474-025-9.13;ьЛаврищева Е.М. Программная инженерия и технология разработки сложных систем, Юрайт. www. biblio-0nline.ru, 2017).

В период информатизации быстрыми темпами начала развиваться *индустрия ПО.* Появились продуктовые линии - ProductLine/ProductFamily (K.Pochl), 2004, технология мультисборки К.Чернетски и У.Эзенакера (2005); новые модели систем *Msys* и базовых характеристик вариабельности MF. Эти модели и метод сборки стали основой реализации изменяемых систем и индустрии множества вариантных продуктов. Появились фабрики программ с методом сборки (Д.Гринфилда и К.Ленца – потоковая сборка, 2007; И.Бей – взаимодействие разноязыковых программ в разных ОС операционных сред, 2005; К.Ленц – Use Case прецеденты (2007); M. Fauler K., Martin P. - re-assemble (2008); КНУ - assembler factory (http://programsfactory.univ.kiev.ua, 2011) и др. В результате создания этой фабрики определена теория фабрик программ из КПИ, объектов, компонентов и сервисов (Theory and practice of software factories K. M. Lavrischeva, 2011, Volume 4, Number 6, Pages 961-972). В ее основе положен метод конвейерной сборки из ГОР и процессы планирования, верификации, тестирования и оценки качества выходного конфигурационного файла системы. Данная отечественная концепция фабрик программ и систем находит постоянно отклики и запросы от специалистов Китая, Таиланда, Южной Кореи, Европы, Канады, США и др.

В русле названных направлений моделирования проводилась за рубежом разработка метода моделирования систем по моделям систем, MF и проводились на Международных конференциях (VAMOS, SPRL, Reusebility, ETAPS и др.). Обзор упомянутых моделей и оригинальный объектно-компонентный графовый метод моделирования основан на функциональных, системных и сервисных элементов и интерфейсных элементах приведен в статье Лаврищевой Е.М. и Петренко А.К. Моделирование систем и семейств систем, Труды ИСП РАН, 2016, том.28, issue 6, с.46-65.

Большое внимание уделяется вопросам обеспечения качества выпускаемых программ и продуктов. В ИСП РАН проведена конференция OS Day. – Надежность, 17-18 мая 2018.

Пакулин Н.В., Лаврищева Е.М. Анализ методов оценки надежности оборудования и систем. Практика применения методов. 5 Научно-практическая конференция - OS DAY, Москва, 17-18мая 2018. Опубликована статья с участием Рыжова А.Г., Зеленова С. на анг. языке.-Труды ИСП РАН, том5 DOI: 10.15514 ISPRAS-2018-30(3), 2018.-р. 99-120.

1.2. Новые направление информатизации - интеллектуализация знаний

Подходы к интеллектуализации описаны в разделах отчета за 2016-2017, обсуждались в докладах конференций «XIII Российский Фестиваль науки “NAUKA 0+” 2018 и на Конференции ISPRAS OPEN -18 и отображена в статьях. См. доклады: Лаврищева Е.М. Современные системы искусственного интеллект», 13.10.2018. в Финансовом университете при Президенте XIII Всероссийского фестиваля науки “NAUKA 0+”, симпозиума «Искусственный интеллект: различные подходы к его воплощению»;

Аветисян А.И., Лаврищева Е.М. «Информатика и ЭВМ. Анализ и аспекты развития» Открытая конференция ИСП РАН им. В.П.Иванникова, 22-23.11.2018. Потом сделана статья - В направлении интеллектуализации жизненного цикла ПО (ISO/IEC Cycle Life 12207-2007) сделан доклад Лаврищевой “Ontological Approach to the Formal Specification of the Standard Life Cycle, "Science and Information Conference-2015", Jule 28-30, London, UK, www.conference.thesai.org.- p.965-972 и Lavrischeva E.M. Ontology of Domains. Ontological Description Sofware Engineering Domain—The Standard Life Cycle, Journal of Software Engineering and Applications, July 24, 2016). На эту статью много запросов в среде Интернет из Китая, Тайланда, Индии и др. Сегодня интеллектуализация касается Больших данных (Big Data). Это направление особенно стало актуальным в связи с роботизацией разных сфер человеческой деятельности, в том числе работы с огромными объемами данных, получаемых с космоса, недр земли, океана и др. (См. статья Лаврищева Е.М. Рыжов А.Г. Применение теория общих типов данных стандарта ISO/IEC 12207 GDT применительно к Big Data.- The cconference “Actual problems in science and ways their development”, 27 desember 2016, http://euroasia-science.ru/ p.99-110.

Опубликована статья, посвященная информатике и программированию информационных и программных систем ИСП РАН с учетом проекта Информатизации России.

Lavrischeva E.M.. Petrenko A.K. Informatics. Formation of computer software and technologies of software systems. Trudy ISP RAN/Proc. ISP RAS, 2018.

Далее описываются теория и технология программирования с начала появления ЭВМ и метод компонентного моделирования ОКМ задачи проекта и пути их решения. Рассматриваются подходы к созданию вариантов ОС и приводится результаты создания экспериментального варианта ОС Linux и веб-систем из готовых сервисных ресурсов Интернет для класса прикладных, информационных и программных систем (см. Список литературы и зарубежных статей).

1.3. Разработаны средства обработки фундаментальных и общих типов данных ресурсов

Тип данных — совокупность элементов, выделенных на всём множестве предметной области (R. Hindley, 1960). Полиморфный тип — представление набора типов как единственного типа (R. Miller, 1960). Математический тип - это:

– множество всех значений, принадлежащих типу.

– предикат, определяющий принадлежность объекта к данному типу.

Тип данных (ТД) используется в описании программных модулей на ЯП (Algol, Пролог, PL/1, Fortran, Pascal и др.). Аксиоматика ТД разработана С.Дейкстрой, Н. Виртом, В. Турским, П. Науром, А.Замулиным, Н. Агафоновым и др. в 1970-х.

Любые данные, с которыми оперируют программы в ЯП, относятся к следующим стандартным ТД.

– FDT – Fundamental Data Type. Фундаментальные типы данных.

– GDT – General Data Types. Общие типы данных (ISO/IEC 11404 GDT).

– Big Data – Большие данные.

Система типов данных реализована в .Net. В ней представлены *типы-значения (*value type) *и типы-ссылки (*reference type). *Типы-значения* – это статические типы в CTS, их значения могут занимать память от 8 до 128 байтов. Они копируются при присвоении им значений. *Типы-ссылки* используют указатели на объекты, которые они типизируют, а также механизмы хранения и освобождения памяти. Ссылочные типы включают: объектные типы (object type), интерфейсные типы (interface type), типы-указатели (pointer type) и др.
(Лаврищева Е.М. Рыжов А.Г. Применение теория общих типов данных стандарта ISO/IEC 12207 GDT применительно к Big Data.- The cconference “Actual problems in science and ways their development”, 27 december 2016, http://euroasia-science.ru/ p.99-110.

Реализована теория сборки и преобразования ТД в отечественных работах и сделаны доклады на конференции в Зеленограде 2017. Развитие ВТ в России и в странах бывшего СССР. История и перспективы.- 3-5 октября 2017. В том числе доклад: Лаврищева Е.М. Развитие теории программ и систем в СССР. История и современные теории.- Сборник SORUCOM-17, 2017. -с.162-176. (Lavrischeva E.M.. Development of the theory programs and systems in the USSR. History and modern theory .- Sorucom-2017, IEEE Springer-2017. P.31-47).

1.3.1. Отечественный метод преобразования ТД разнородных ресурсов на основе интерфейсов

В мировых библиотеках Интернет существует большое количество разнородных программ для вычисления физических, биологических и других задач. Данные могут быть представлены в виде пространственных зрительных образов, отчетов и наборов данных, генерируемых с различных датчиков или специализированной аппаратуры. Такие данные относятся к классу больших данных и при вычислениях требуют нестандартных методов и приемов для их анализа, обработки и организации вычислений.

Типы данных стандарта GDT ISO/IEC 11404-1996, 2007

К общим типам данных (GDT - General Data Types) стандарта ISO / IEC 11404-2007 относятся:

– независимые от языка типы данных, которые используются для формального описания концептуальных данных, их элементов и значений;

– полуструктурированные и неструктурированные совокупности данных, в которых ТД являются неизвестной или неопределенной заранее структурой данных;

– расширяемые общие типы данных.

Стандарт GDT устанавливает номенклатуру и семантику наборов типов данных, которые используются в языках программирования и в интерфейсах программных систем. В этом стандарте специфицированы базовые типы данных и сложные, которые полностью или

частично определяются с помощью простых типов данных. Термин «независимые от языка типы данных» означает, что специфицированные типы данных образуют классы, представители которых в языках программирования (ЯП) соответствуют общей концепции типов данных стандарта GDT ISO/IEC 11404 и частично совпадают с фундаментальными типами данных (ФДТ) ЯП.

Примитивные типы данных стандарта GTD

Рациональный (rational) – математический тип данных, который соответствует действительным числам.

Масштабированный (scaled) – это семейство типов данных, пространством значений которого является подмножество рациональных чисел, а каждый отдельный тип данных имеет фиксированный знаменатель и предполагает аппроксимацию его значений.

Комплексный (complex) – это семейство типов данных, каждый из которых задает числовой математический тип данных для комплексных чисел.

Пустой (void) – это тип данных, который задает объект с необходимыми синтаксическими и семантическими описаниями и не несет никакой информации.

Значение типа данных (ТД) определяется путем:

1) перечисления;

2) аксиоматического определения;

3) подмножество пространства значений;

4) комбинация любых значений путем конструирования новых значений.

Сгенерированные типы данных стандарта GDT

Сгенерированный ТД – это ТД, полученные в результате генерации типов данных.

Генерация ТД семантически зависит от параметрических типов и собственных характеристических операций. Важной характеристикой всех генераторов ТД является то, что генератор может применяться к разным параметрическим ТД. Генераторы указателя и процедуры дают типы данных, значения которых атомарные, тогда как генератор выбора и агрегатных типов данных выдает типы данных, значения которых позволяют производить их декомпозицию. ***Генератор ТД*** – это концептуальная операция над одним или несколькими типами данных, которая создает новый тип данных. Генератор оперирует с типами данных, а не с его значениями и представляет собой:

1) набор критериев для характеристик типов данных, над которыми будут выполнены операции;

2) процедуры конструирования, которые допускают набор типов данных с данным критерием для создания нового пространства значений из пространств значений этих типов данных;

3) набор характеристических операций, которые применяются в конечном пространстве значений для завершения определения нового ТД.

Агрегатный тип данных - это сгенерированный ТД, каждое значение которого получено из значений параметрических ТД. Параметрические ТД агрегатного ТД или его генератор включают в себя имена компонентов ТД. Генератор агрегатного ТД выдает ТД с помощью алгоритмической процедуры в пространстве его значений. Агрегатный ТД обеспечивает доступ к компонентам значений через характеристические операции. Агрегатные значения разных типов различаются между собой свойствами, которые задают отношение между компонентами ТД и между каждым компонентом и агрегатным значением.

Сложные типы данных стандарта GDT и генераторы ТД

Множество (set) задает тип данных, пространство значений которого составляет набор всех поднаборов пространства значений. Операции соответствуют математическому множеству set. Стандарт GDT включает генераторы ТД сложных типов данных: выбор (choice), указатель (pointer), процедура (procedure), запись (record), набор (set), портфель (bag), последовательность (sequence), массив (array), таблица (table) и т.п.

Характеристические операции создают значение любого типа с помощью генератора ТД в пространстве значений параметрических ТД. Практически не существует

уникальной коллекции характеристических операций для заданных ТД. Одна коллекция операций ТД достаточна для выделения этого ТД среди других из пространства значений той же мощности.

Преобразование типов данных ISO/IEC 11404-96

Данный Стандарт определяет LI–язык, который преобразует ТД независимо от ЯП, и включает следующие виды преобразований:

– внешнее преобразование внутренних ТД ЯП в LI–типы данных;

– внутреннее преобразование LI–типа данных в ТД ЯП;

– обратное внутреннее преобразование к внешнему.

Внешнее преобразования ТД состоит в следующем:

а) установки связи с LI–типом данных;

в) задание связи между допустимым значением и его эквивалентным значением.

с) LI–типа данных определяется значением любого внутреннего типа данных.

Внутреннее преобразование задает связь примитивного ТД или сгенерированного в LI–тип данных с внутренним ТД ЯП.

Обратное внутреннее преобразование LI–типа данных состоит в преобразовании значений внутреннего ТД в соответствующее значение LI–типа при наличии соответствия и отсутствия двусмысленности. Это преобразование для ЯП является коллекцией обратных внутренних преобразований LI–типа данных.

Генерация ТД стандарта GDT

Разработана схема генерации ТД GDT в структуры фундаментальных ТД ЯП. Предлагается разработать библиотеку функций генерации ТД стандарта GDT, элементы которой для всех новых ТД этого стандарта выполняют следующие виды операций:

– преобразование ТД, содержащихся в ЯП ($ЯП_1$, …, $ЯП_n$) и которые входят в состав ФДТ ТД;

– функции трансформации сложных ТД стандарта GDT, включенных в стандартную библиотеку CTS VS.Net и используются трансляторами с ЯП этой системы для преобразования сложных ТД к более простым;

– операции взаимодействия компонентов повторного использования, записанных в разных ЯП, интерфейс которых задается в языке IDL.

Все ТД стандарта GDT представлены в виде следующих классов алгебраических систем:

$\sum 1 = \{G\alpha^b , G\alpha^c , G\alpha^u , G\alpha^r\}$,

$\sum 2 = \{G\alpha^a , G\alpha^z , G\alpha^u , G\alpha^e\}$,

$\sum 3 = \{G^s , G^{ns}\}$, где $G\alpha^t = <X\alpha^t, \Omega\alpha^t>$, t – ТД языков L,

$X\alpha^t$ – множество значений ТД;

$\Omega\alpha^t$ – множество операций над ТД;

$\sum 1$ - простые ТД: $t = b$ (bool), c (char), i (int), r (real)),

$\sum 2$ – сложные (структурные) типы данных: t = a (array), z (record), u (union), e (enum).), как комбинации простых ТД;

$\sum 3$ - сложные, неструктурированные ТД (портфель, контейнер, протокол и т.п.) и сгенерированные из них более простые данные.

В каждом классе этих систем преобразование $t \rightarrow Tq$ для пары языков l_t и l_q основано на таких свойствах отображений:

1) системы $G\alpha^t$ и $G\beta^q$ – изоморфны, если их q, t определены на том же множестве ТД;

2) между значениями $X\alpha^t$ и $X\beta^q$ типов данных t, q существует изоморфизм, если множество операций $\Omega\alpha^t$ и $\Omega\beta^q$ разные;

3) если множество $\Omega = \Omega\alpha^t \cap \Omega\beta^q$ не пустое, то имеет место изоморфизм двух систем

$G\alpha^{t'} = < X\alpha^{t'}, \Omega >$ и $G\beta^{q'} = < X\beta^{q}, \Omega >$.

4) если типы данных отличаются, например, t - строка, а тип q – вещественное, то между множествами $X\alpha^{t}$ и $X\beta^{q}$ не существует изоморфного соответствия.

Отображения сохраняют линейный порядок элементов к виду линейной упорядоченности элементов алгебраических систем из этих классов.

1.3.2. Неструктурированные большие данные- Big Data

В результате проведенных нами исследований неструктурированных данных из класса больших данных (Big Data), рассмотрена применимость к ним стандартных ТД ЯП и GDT.

Неструктурированные данные GDT – это совокупность данных, которые:

1. Структурированные ТД или метод доступа;

2. Наполовину структурированные данные, которые имеют один ТД или метод доступа;

3. Неструктурированные данные, включающие набор данных неодинаковой природы.

Приложения, основанные на работе со структурными ТД, включающими реляционные и нереляционные данные, которые используют одну из трех архитектур:

– реляционные данные находятся в БД, а большие нереляционные данные двоичных объектов (BLOB), которые находятся в файловых системах или на файловых серверах;

– нереляционные данные из хранилищ, предназначенных для BLOB-данных;

– реляционные и нереляционные данные, которые находятся в БД.

Эти данные используются разными приложениями путем:

– создания, загрузки, обновления и удаления неструктурированных данных и использования транзакционной согласованности между источниками неструктурированных данных;

– индексирования неструктурированных данных и их поиска;

– извлечения метаданных в явной форме и предоставление их пользователям;

– анализа и преобразования содержимого документов к форматам для выполнения поиска и составления запросов (например, преобразование звуковых файлов в текстовые и выполнение поиска по запросу БД).

Хранение неструктурированных данных в хранилищах проводится с помощью BLOB-данных. При хранении BLOB-данных в БД централизованного хранилища снижаются затраты и быстродействие.

Анализ нестандартных данных из наборов больших данных

К методам анализа данных относятся статистические методы (дескриптивный анализ, корреляционный и регрессионный анализ, компонентный анализ и др.), а также методы синтаксического анализа раздела описания сложных данных GDT.

Регрессионный и компонентный метод математически ориентированы на оценку необходимой величины экспертом и сравнения ее с другими величинами.

Описание ТД представляется в виде таблицы СМ метода терминальных символов и семантических программ их реализации. Каждому представлению терминальных символов соответствует операционный знак его обработки (+, - , / и др.).

Функции их реализации могут повторяться и для других ТД.

Для проведения анализа ТД предлагается создать таблицу ТД и набор функций их реализации в языке XML.

Таблица функций включает набор неструктурированных ТД, которым прикреплены функции трансформации таких ТД к имеющимся в первой таблице. Результатом обработки этой таблицы является разложение неструктурных данных в виду простых данных в том порядке, в котором они заданы в исходной таблице.

В проекте на 2017 будет разработан набор функций обработки нестандартных типов данных.

1.3.3. Формальное описание изменяемых прикладных систем объектно-компонентным методом ОКМ

ОКМ метод обеспечивает создание систем по графу и объектной модели (ОМ) системы из объектов и интерфейсов включает:

– математические операции (*объединения* ∪, *пересечения* ∩, *вычитания* ⊕, *симметричного вычитания* и др.) для формирования программных структур и трансформация ОМ к компонентной модели (СМ);

– объектную и компонентную алгебры для изменения КПИ (CRU, reuses, assets, servises, artifacts и др.);

– операции конфигурирования ГОР, CRU IEEE 828-96 и преобразованиz данных к разным форматам современных сред (VS.Net, IBM, Corba, Eclipse и др.) с помощью примитивных функций GDT ISO/IEC 11404–2007 к FDT LP (и обратно);

– линии (типа ProductLine, ProductFamily) для создания отдельных систем и семейств с помощью ГОР и трансформации данных передаваемых через интерфейс GDT↔FDT (Лаврищева Е.М. Теория объектно-компонентного моделирования программных систем.-www.ispras.ru/preprint/prep_29_2017).

Объекты в ОКМ определяются на четырех уровнях проектирования с привлечением логико-математических формализмов описания ОМ и с уточнением функций объектов и их свойств и характеристик.

К формальным уровням представления ОМ из объектов относятся:

1. Обобщающий уровень
2. Структурный уровень
3. Характеристический уровень
4. Поведенческий уровень

(Katerina Lavrischeva, Andrey Stenyashin, Andrii Kolesnyk. Object-Component Development of Application and Systems. Theory and Practice /Journal of Software Engineering and Applications, 2014, 7, August 2014, http://www/.scirp.org/journal/jsea)

На первых двух уровнях формируется граф предметной области из выделенных объектов и их связяях.

На *структурном уровне* для объектов графа формируются концепты с помощью унарных предикатов: $P = (P_1, P_2, \ldots, P_r)$ и функций концептов $Con_i = \{P_{ik}\}$, $P_{ik} = P_k(O_i) = \text{true}$. Алгебраическая система $\Sigma = (O', \Omega)$ этого уровня включает операции $\Omega = (O, P)$ определения внешних и внутренних характеристик объектов.

На *поведенческом уровне* с помощью атрибутов объектов и их значений формируются состояния объектов в виде диаграмм переходов состояний. Параметр времени *t* (*Timer)* вносится в ОМ для рассылки специальных сообщений в текущий момент времени выполнения объекта**.**

Граф G модели ОМ формируется в виде:

G_{t1} – граф объектов на обобщающем уровне (t=1);

G_{t2} – граф характеристического уровня (t=2);

G_{t3} – граф структурного уровня (t=3);

G_{t4} – граф интерфейсов для взаимосвязи объектов на поведенческом уровне (t=4).

Объектам функций G_{t1} и G_{t4} , их характеристикам соответствуют методы и данные (уровня 2, 3)для обеспечения их взаимодействия.

Элементы графа входят в состав *модели ПС* – $M_{nc} = (CL, M_f, M_s, M_i, M_d)$,

где CL – отдельные элементы в языках $L = L_1, L_2, \ldots, L_N$;

$M_f = (O_1, O_2, \ldots, O_r)$ - множество функциональных элементов;

$M_s = (Ms_{\text{in}}, Ms_{\text{out}}, Ms_{\text{inout}})$ – множество интерфейсов;

M_d - множество данных и метаданных ПС.

Модель вариабельности ПС – $MF_{var} = (SV, AV)$,

SV – подмодель вариабельности артефактов в структуре ПС;

AV – подмодель вариабельности готовых ресурсов ПС.

Компонентная модель (CM) имеет вид:

$CM = \langle RC, In, ImC, Fim \rangle$,

где *RC* – базовые компоненты множества С, полученные после трансформации объектов графовой модели ОМ;

In – множество интерфейсов для reuse в точках вариантности;

ImC – множество реализованных базовых компонентов в среде;

m (·) – функции преобразования входных и выходных параметров интерфейсов во множестве данных ПС, СПС.

Операции над компонентами:

Операция добавления, удаления, замены компонента, которая задается знаком "–" и имеет вид: $CE.NameSpace(C1) - C2 = (CE \lozenge C\ 1) \oplus C\ 2$.

Операция объединения ($\cup$) и др.

Компонентная алгебра – это набор операций для представления переменных объектов и их эволюции: $\sum = \{\varphi 1, \varphi 2, \varphi 3\}$, где

$\varphi 1 = \{CSet, CESet, \Omega 1\}$ – внешняя алгебра,

$\varphi 2 = \{CSet, CESet, \Omega 2\}$ – внутренняя алгебра,

$\varphi 3 = \{Set, CESet, \Omega 3\}$ – алгебра эволюции компонентов.

С помощью операций алгебры проводиться замена, удаление и добавление нового объекта.

1.3.4. Взаимодействия и верификации программ и систем

Взаимодействие – это взаимосвязь двух и больше объектов или систем через интерфейс.

Модель взаимодействия *Мвз* имеет вид: $Мвз = \{Мпр, Мсис, Мсред\}$, где

$Мпр = \{Com, Int, Pro\}$ – модель программы,

$Мсис = \{FPC, Int, Pro\}$ – модель системы,

$Мсред = \{Envir, Int, Pro\}$ – модель среды,

Int, Pro - совокупность интерфейсов и протоколов, которые передают данные между программами разных сред *Envir* сети.

Модель Мвз по отношению к стандартной модели открытых систем OSI является моделью верхнего уровня, которая реализуется сервисом Eclipse в Интернет.

Разработаны конкретные модели для современных сред. (студент МФТИ -Островский А.В. Подход к обеспечению взаимодействия сред Java, MS.Net // Проблемы программирования. – 2011. – № 2. – С. 37–44):

Модель Visual Studio ↔ Eclipse.

Модель CORBA ↔ Visual Studio ↔ Eclipse.

Модель JAVA ↔ Visual Studio ↔ Eclipse.

Эти модели отработаны на примерах ряда прикладной систем.

Верификация моделей MF и систем

Истоки формальной верификации и доказательства правильности программ лежат в Венском методе (VDM). В нем дана система доказательство программ с помощью формальных методов проверки правильности или неправильности объекта в соответствии со спецификацией свойств программ. Объектами верификации являются модель характеристик MF для разных предметных областей и требования к разработке новой ПС, а также методы проверки, основанные на model checking.

В них свойства объектов, которые подлежат верификации в модели MF описываются средствами линейной темпоральной логики (Linear Temporal Logic (LTL)) или логики деревьев вычислений CTL (Computational Tree Logic).

Средствами метода model checking устанавливается соответствие спецификации объектов требованиям и свойствам. Если соответствие удовлетворено, то верификатор сообщает о правильности модели или дает пояснение о возникшем несоответствии (рис.1).

Формальная верификация объектов – это *дедуктивный анализ* и проверка модели (model checking) заданным формальными спецификациями или темпоральной логикой.

Верификация model checking применяется к моделям систем с конечным числом состояний. Основу метода верификации составляет математическая формулировка требований к создаваемым программам и алгоритмы формальной проверки требований.

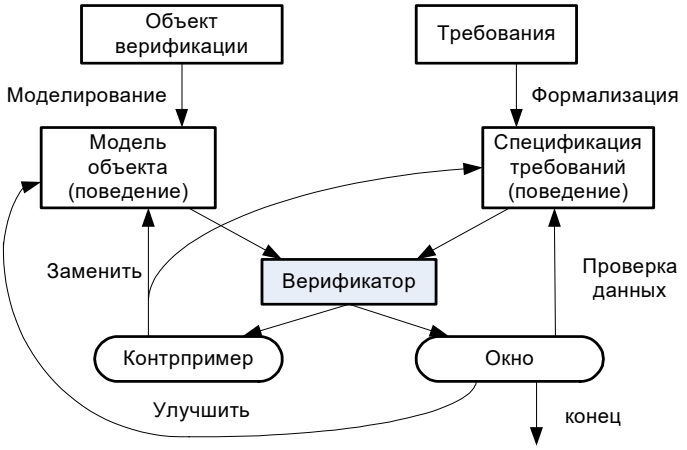


Рис.1. Верификация модели графа ОМ

Модель Крипке задается в виде: $M = (S, S0, R, L)$, где

S — множество состояний,

$S0$ — множество начальных состояний — отношений переходов $L: S \to 2^{AP}$ x0 — функция разметки.

На основе спецификации формулируются утверждения, истинность которых необходимо проверить. По описанию модели осуществляется автоматическая верификация, в результате которой либо устанавливается, что модель удовлетворяет спецификации, либо делается ее опровержение в виде перечня действий над моделью, которые привели к нарушению спецификации.

Разработан ряд инструментов поддержки верификации моделей и систем. Это:

FeatureModelPlugin FMP) – реализован как плагин Eclipse. **FMP** поддерживает моделирование, построение модели характеристик и конфигурирования СПС (http://gsd.uwaterloo.ca/featureModelingAndModelTemplates).

CaptainFeature основан на нотации FODA для построения MF и содержит конфигуратор для специализации созданных моделей (https://sourceforge.net/projects/captainfeature/).

Requiline – это инструмент инженерии требований для эффективного управления в SPLE. Из описаний характеристик модели и требований «выводится» конфигурация продукта. Включает функцию контроля (checker) согласованности и не выполняет другие операции анализа. Проводится в таких средах функционирования как Microsoft.NET, Oracle DBMS.

Pure::Variants – инструмент поддержки моделирования характеристик, конфигурирования с помощью средств Prolog, решателя ограничений (constraintsolver) (https://www.pure-systems.com/products/pure-variants-9.html).

AHEAD ToolSuite (ATS) – это набор инструментов SPLE для разработки и поддержки характеристик и их компоновки. Можно выполнять определенные операции анализа модели с помощью SAT-решателей и сохранять ее в виде правил грамматики (http://www.cs.utexas.edu/~schwartz/ATS.html).

1.3.5. Управление вариабельностью моделей и систем

Управление вариабельностью СПС проводится по точкам вариантности, вариантным артефактам ПС, ограничениям и зависимостями через предикаты, опреде на множестве точек вариантов ПС. Для управления вариабельностью используется метод Е. Деминга, основанный на функциях *F1- F4*, которые задают действия по планированию и контролю проведения изменений в СПС.

Каждая функция *F1- F4* выполняет следующее:

F1 – подготовку артефактов СПС (**Act**);

F2 – планирование системы СПС из артефактов (**Plan**);;

F3 – системный контроль и проверка состояния изменений СПС (**Check**);

F4 – выполнение актуализации систем СПС (**Do**).

Управление вариабельностью СПС с учетом требований *R* (Requriment) состоит в:

1) обосновании решений для функции *F1* (*R1*);

2) согласовании способа реализации артефактов на процессах СПС (*R2*);

3) реализации проверки правильности создания СПС (*R3*);

4) отслеживании связей между характеристиками ПС и СПС (*R4*).

В соответствии с требованиями *R1 – R4* и функциями *F1 – F4* осуществляется проверка модельной среды и модели вариабельности СПС.

Трансформация объектов ОМ к компонентам (СМ)

Для трансформации объектов ОМ в ЯП к программному коду объект графа G преобразуется в компонент графа КМ, вершинам которого соответствуют программные компоненты и интерфейсы (*Comp, C, Io*) вида: $M_{Comp} = (C_{name}, C_{in}, C_{im}, C_{ser})$, где

C_{Name} – имена функциональных компонентов;

$C_{In} = \{C_{Ini}\}$ – набор интерфейсов компонентов;

$C_{Im} = \{C_{Inj}\}$ – набор реализаций функциональных компонентов;

$C_{Ser} = \{C_{Serr}\}$ – набор системных сервисов компонентов.

Формально модель любой подсистемы системы *Msys* задается в виде:

$Msys = <M_f, M_i, M_d>$, где

$M_f = (f_{O1}, f_{O2}, \ldots, f_{Or},)$ – множество функций ПрО;

$M_i = (Io_1, Io_2, \ldots, Io_n)$ – множество интерфейсных элементов для функциональных элементов - f_{On} входные интерфейсы *in,* выходные интерфейсы *out* и входные-выходные - *inout*;

$M_d = (M_{d1}, M_{d2}, \ldots, M_{dn})$ - множество данных и метаданных ПрО, с которыми работает система.

Элементы множеств M_{sys}, M_i и M_d - это компоненты, интерфейсы и общие данные. Они могут иметь пересечение по данным, которые относятся к *in*, *out*, *inout* и входят в состав внешних типов данных, которыми компоненты обмениваются через сервер Application.

Компонент (Comp, C) – это реализация отдельной функции объекта, заданной функции, в ОМ, и включает в себя интерфейсный раздел и код реализации самой функции. Компонент может иметь несколько реализаций в зависимости от операционной среды, модели данных, СУБД и др. Интерфейс задает данные для сборки одного компонента с другими. Компонент может иметь несколько интерфейсов и наследуется в виде классов модели компонента и каркаса на любом ЯП (C, C++, Ruby, Basic, Java, Python и др.).

В общем случае, в КМ могут входить такие типы компонентов: программные, сервисные, системные, серверные, клиентские, веб-серверные и др.

Каждый из этих типов компонентов имеет интерфейс в IDL/WSDL. Компоненты должны удовлетворять требованиям и правилам взаимодействия с другими компонентами.

1.4. Моделирования систем из компонентов

Моделирование систем из компонентов проводится с помощью операций спецификации, поиска компонентов, сборки/конфигурирования, развертывания (deployement) в компонентной среде, тестирования и проверки правильности выполнения функций с учетом требований к системе (См. статья 1 Лаврищева Е.М. Теория объектно-компонентного моделирования программных систем. Препринт ИСП РАН, № 29, 2016 г. - М.: 48 с. ISBN 078-5-91474-025-9.13 и книга «Программная инженерия. Парадигмы, технологии, Case-средства программирования", www.biblio-0nline,ru, 2016). В них описаны операции моделирования теоретически и с формальным доказательством их применения в разработке прикладных систем.

Для компонентов разработаны *базовые операции* сборочного типа, реализованные в разных общесистемных средах (IBM, MS.VS, .NET и др.) и *компонентная алгебра операций.*

Базовые операции моделирования систем из компонентов включает набор операции вида: объектами:

Link C_3 ($C_1 \cup C_2$) – сборка компонентов;
Make C_3 (C_1 , C_2) – связь компонентов;
Config C_4 ($C_1 \cup C_2 \cup C_3$) – конфигурирование компонентов;
Reing (C_n) – реинженерия компонентов;
Revers (C) – реверсная инженерия компонентов;
Prove PS (CPI_i) – доказательство правильности КПИ в системе;
Creat (Cj) – образование класса компонентов;
Verific (C) - верификация компонентов и системы;
Testing (C) - тестирование компонентов и системы;
Funct (Sys A) – выполнение компонентов и системы.

Суть приведенных операций состоит в замене, переименовании компонентов и интерфейсов, добавлении новых компонентов, изменения интерфейсов и реализаций, восстановление исходного представления, преобразования функций компонентов и др.

Используются также ранее приведенные операции компонентной алгебры: make, config реализованные в IBM, Microsoft, BSD, GNU и др.

Реализация оператора link представлена на сайте http://7dragons.ru. Пример link (A, B) промоделирован на сайте в статье студента (Радецкий И.А. Один из подходов к обеспечению взаимодействия сред MS.Net и Eclipse// Проблемы программирования, 2011, №2.-45-52) Фрагмент реализации приведен в Приложении 1.

Оператор сборки make реализован в BSD и GNU (cmake). Они позволяют собирать исполняемые файлы из библиотеки программных ресурсов. Порядок сборки ресурсов задается в файле Makefile, в котором указываются правила (зависимость, tab) и действия над компонентами в разных ЯП в системах GNU и Visual Studio.VS см. рис.2, 3.

Make in GNU

```
edit : main.o kbd.o command.o display.o \
 insert.o search.o files.o utils.o
 cc -o edit main.o kbd.o command.o display.o \
 insert.o search.o files.o utils.o
main.o : main.c defs.h
 cc -c main.c
kbd.o : kbd.c defs.h command.h
```

```
 cc -c kbd.c
command.o : command.c defs.h command.h
 cc -c command.c
display.o : display.c defs.h buffer.h
 cc -c display.c
insert.o : insert.c defs.h buffer.h
  cc -c insert.c
search.o : search.c defs.h buffer.h
 cc -c search.c
files.o : files.c defs.h buffer.h command.h
 cc -c files.c
utils.o : utils.c defs.h
 cc -c utils.c
clean :
 rm edit main.o kbd.o command.o display.o \
 insert.o search.o files.o utils.o
```

Рис.2. Пример Makefile в GNU.

Make в Visual Studio.VS

Генерация CMakeLists.txt к Makefile в Visual Studio.VS включает следующие операции:

add_executable(exec_name source1 source2 …) # создает исполняемый файл exec_name из файлов исходного кода source1, source2, и т.д.

add_library(lib_name sout rce1 source2 …) # создает библиотеку lib_name из файлов исходного кода компонентов source1, source2, и т.д.

target_include_directories(target PUBLIC dir1 dir2 …) # включает директории dir1, dir2, … в список поиска заголовочных файлов при сборке (исполняемого файла или библиотеки) target.

target_link_libraries(target lib1 lib2 …) # выполняет link (включение в сборку) библиотек lib1, lib2.

cmake создает исполняемый файл myExec cmake_minimum_required (VERSION 2.6), project (MyProject)

add_executable(myExec source1.cpp source2.cpp).

Рис.3. Реализация операции cmake в Visual Studio.3

Приведенные операции обеспечивают сборку, конфигурацию компонентов в моделируемой среде и в общесистемных средах компонентов, которые записаны в разных ЯП (C, C++, Basic, Java и др.).

Компоненты модели КМ проверяются на правильность задания их функциональности, после чего проводится их сборка или конфигурационная сборка согласно стандарта IEEE 828-96-2012 (Configuration) файлов версий подсистем выходной системы.

Далее рассматривается моделирование систем из готовых ресурсов ГОР, создаваемых с помощью новых парадигм программирования (сервисной, сервисно-компонентной, агентной и др.), элементы которых реализуются в клиент-серверных средах систем IBMSphere (SOA, SCA, SCM), Semantic Web и др. См. статьи представленные в этой главе проекта РФФИ:

– Лаврищева Е.М., Петренко А.К. Моделирование систем и их семейств. Труды ИСП РАН, 2016, том 28. вып. 6. - с. 180-190. DOI: 10.15514/ISPRAS-2016-28(6)-4.

– Е.М. Лаврищева, В.С. Мутили, А.Г. Рыжов**.** «Аспекты моделирования вариабельных программных и операционных систем. Сб. XIX Всероссийский научной конференции «Научный сервис в сети Интернет», Новороссийск, 18-23 сентября 2017.-327-341;

– Е. М. Лаврищева, Л. Е. Карпов, А. Н. Томилин. Подходы к представлению научных

знаний в интернет науке. Сб. XIX Всероссийский научной конференции «Научный сервис в сети Интернет», Новороссийск, 18-23 сентября 2017.-с.310-326).

1.5. Создание ПС и СПС из системных и прикладных сервисов

Системные компоненты Интернет управляют оборудованием и серверами. К ним относятся: клиент, сервер или Интернет браузер. Они производят обработку запросов от клиента к серверу и обратно в клиент-серверной архитектуре. Первоначально такая архитектура была реализована в системе CORBA и включалет системные компоненты:

– брокер объектных запросов (*Object Request Broker* — ORB) для взаимодействия клиент-объектов с сервер-объектами на ЯП (Smalltalk, Cobol, Ada-95, Lisp, PL/1, C++, Python, Java, IDLScript и др.);

– общие объектные сервисы (*Common Object Services* — COS) для управления изменениями, реализациями, транзакциями, подпроцессами и т.п.;

– общие средства обслуживания (*Common Facilities* — CF) для объединения в различные конфигурации сервисных объектов;

– объектные приложения (*Application Objects* — AO), над которыми могут производиться операции - открыть, инсталлировать, переместить, поместить, выполнить.

Объект-клиент и объект-сервер обмениваются между собой с помощью запросов, каждый из которых обрабатывается брокером ORB на основе интерфейсов объектов для клиента, сервера и ядра ORB.

Интерфейс клиента (*Client Interface*) обеспечивает взаимодействие с объектом-сервером и состоит из трех базовых интерфейсов:

– *stub-интерфейса*, содержащего описание внешних параметров и операций объекта в IDL;

– *интерфейс динамического вызова* (Dynamic Invocation Interface — DII) объекта, определяемого во время выполнения программы клиента при поиске интерфейса в репозитории интерфейсов или в репозитории реализаций;

– *интерфейса сервисов* ORB (ORB Services Interface), содержащего набор сервисных функций, которые клиент запрашивает у сервера через брокер ORB.

Stub-интерфейс обеспечивает взаимосвязь клиента с ORB через *stub* и посылает параметры серверу в запросе. Схема взаимодействия клиента с объектом соответствует схеме вызова RPC удаленных процедур.

Интерфейс DII обеспечивает доступ объектов и их интерфейсов во время выполнения. В каждом вызове указывается тип объекта, тип запроса и параметры. По этой информации извлекается соответствующая программа из репозитория интерфейсов и репозитория реализаций.

Описание интерфейса в IDL начинается с ключевого слова interface, за которым следует: имя интерфейса, описание типов параметров и операций (op_dcl) вызова объектов: **interface** A { ... } и др., а описание типов данных компонентов начинается ключевым словом typedef, за которым следует базовый или конструируемый тип и его идентификатор. В качестве константы может быть некоторое значение типа данного или выражение, составленное из констант. Типы данных и константы описываются как фундаментальные типы данных ЯП: integer, boolean, string, float, char и др. (см.п.1.3.8 и статью -

Лаврищева Е.М. Рыжов А.Г. Применение теория общих типов данных стандарта ISO/IEC 12207 GDT применительно к Big Data.- The cconference “Actual problems in science and ways their development”, 27 december 2016, http://euroasia-science.ru/p.99-110.

Генерация клиента и сервера

Для клиента и сервера генерируется два класса компонентов: клиент и сервер;

В сгенерированный класс – клиент помещаются механизмы связи (коммуникации) между программными компонентами. Пользователь этого класса не должен ничего менять, а использовать в таком виде:

```
…
int param;
…
ClientInterface ci = new ClientInterface();
int result = ci.func(param);
…
```

Здесь ClientInterface – это сгенерированный класс, который содержит механизмы, необходимые для передачи данных на сервер, который предоставляет сервис int func (int param).

Если сгенерированный класс – сервер, то кроме механизмов связи с клиентами в него помещают пустые функции. Они отвечают задекларированным методам, которые описаны в интерфейсе. Для использования сервера клиенту необходимо реализовать конкретную логику данных методов следующим образом:

```
class ServerInterface {
  …
  public int func (int param){
    …//реализация метода
  }
…
void main(){
  …
  ServerInterface si = new ServerInterface();
  si.work();
  …}
```

На этих примерах показаны функции и связывание ь разноязычных модулей через аппарат клиента и сервера.

Тестирование систем и семейств из объектов

В состав систем входят ГОР и *рабочие продукты* для тестирования (планы, набор тестов, тестовые данные и др.). Тест-код создается для тестирования отдельных элементов ПС и образует набор тестов СПС. Метод тестирования СПС базируется на методе Дж. МакГрегора (McGregor) «от требований» (requirements-based testing). В нем задаются действия по управлению тестированием «от требований» с помощью тестов, проверяющих функциональные и интерфейсные объекты. Контроль тестирования объектов и интерфейсов обеспечивает оценку критериев качества систем и их вариантов. В качестве инструмента тестирования используется *TestManager* фреймворка VisualStudio, содержащий средства проверки правильности тестирования разных ГОР и планирования процесса тестирования при выполнении тестовых сценариев, приведенных в таблице 1.

Таблица 1. Основная схема тестирования систем и семейств

Виды тестирования	Объекты тестирования
Тестирование программной архитектуры системы	Все ПС семейства, отдельные ПС и функциональные элементы.
Набор тестов СПС, адаптированных к конкретной ПС	Все ПС семейства, отдельные ПС семейства
Тестирование требований (ScenTED)	Отдельные ПС семейства

Самотестирование	Отдельные объекты и интерфейсы
Применение метаданных	Отдельные объекты и интерфейсы
Оценка тестопригодности компонентов (для повышения производительности)	Отдельные объекты и интерфейсы
Ориентация на сервисное обеспечение и надежность	Отдельные объекты и интерфейсы
Автоматическая генерация тестов по спецификациям в булевой форме	Все ПС семейства, отдельные ПС семейства
FCTA (Fault Contribution Tree Analysis)	ПС семейства

Таким образом, по методу тестирования проводится:

1. Тестирование артефактов, приложений, отдельных ПС и ГОР.

2. Тестирование отдельных характеристик объектов СПС с помощью тестов (автономных и общих).

3. Проверка степени тестирования функциональных и интерфейсных объектов СПС в среде Microsoft VisualStudio на инструментах TeamFoundationBuild и Microsoft Test Manager.

В завершении тестирования определяется количественная метрика *KT* степени тестирования объектов ПС:

KT = 1, если операции тестирования объекта независимы одна от другой;

KT = 0, если операции зависят от пути выполнения и взаимосвязей.

KT принадлежит отрезку [0; 1] и вычисляется по следующей формуле:

$$KT = \frac{1}{n}\sum_{i=1}^{n} KTi$$

Конечное значение *KT* СПС задает степень проверки:

KT = 1, если все объекты проконтролированы;

KT = 0, если не все объекты проконтролированы;

0<*KT*<1 означает, что объекты СПС частино проконтролированы.

При тестировании интерфейсов взаимодействующих объектов проводится оценка метрики контроля интерфейса C*I* по формуле:

$$CI = 1/n\sum_{i=1}^{n} CI_i$$

где CI_i – степень корректности преобразования типов данных в *i*-ом интерфейсном объекте. Если *CI* = 1, интерфейс полностью проконтролирован, *CI* = 0 интерфейс не полностью проконтролирован; $CI \in (0;1)$ – интерфейс частино проконтролирован.

Количественное значение метрики *CI* означает*:*

1 – контроль CI_i интерфейса завершен;

0 – в противном случае - нет.

Этим завершается оценка тестирования функциональных объектов и их интерфейсов с помощью тестов систем.

1.6. Проектирование веб-систем из сервисных компонентов в Интернет

Современный сервис (рис.4) включает: ресурсы (Resource), сервис-профиль (ServiceProfile), сервисную модель (Servicemodel) и сервис основу (ServiceGrounding). Каждый экземпляр сервиса может представляться ServiceProfile и поддерживается

ServiceGrounding. Свойства каждого сервиса описываются describedBy для проведения и обслуживания сервисов.

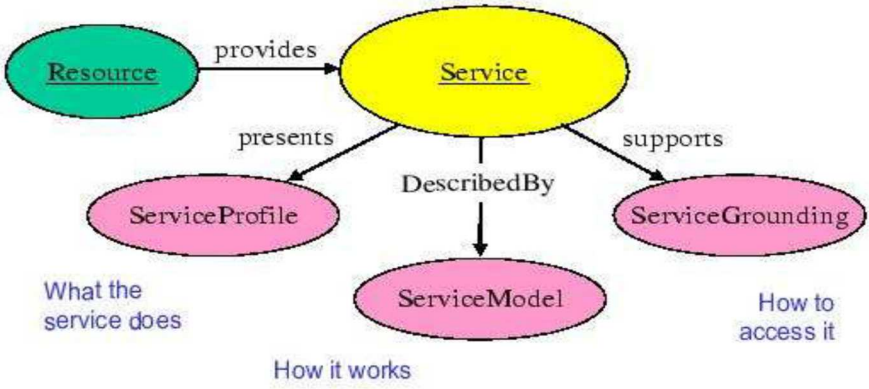


Рис.4. Онтология верхнего уровня сервисов

Модель сервиса задает правила работы сервисной службы. Сервис (ServiceGrounding) задает сведения о том, как агент может получить доступ к сервису с указанием протокола связи, формата сообщений и другие параметры. E-service Интернета поставляется провайдером для компании или учреждение. Сервисы Интернет включают функциональные и системные элементы.

Системные и сервисные сервисы

Системные сервисы обеспечивают решение разного рода деловых и бизнес-задач. К ним относятся: общие сервисы (например, службы именования, каталогизации и др.); объектные сервисы (например, службы диспетчеризации объектными запросами, управления интерфейсом и др.); сетевые сервисы SOA (Service-oriented Architecture), SCA (Service-Component Architecture) для обработки Веб-приложений и систем.

Каждый сервис определяется именем для управления разными ресурсами сети Интернет.

Перечисленные виды сервисов используются при моделировании ПС из готовых ресурсов сети Интернет. Для их использования при создании ПС требуется проводить поиск подходящего сервисного ресурса, его апробацию и встраивание в прикладную программу решения задачи, либо использовать сервис в динамическом режиме.

Таким образом, основным поставщиком сервисов и сервисных компонентов для построения Веб-приложений и систем является Semantic Веб.

Основу Semantic Web составляют модели SOA и: SCA и языки описания программных систем: XML, SOAP, UDDI, WSDL. С их помощью осуществляется реализация базовых свойств веб-сервиса и механизмов взаимодействия между собой веб-сервисов в среде SOA

Языки описания и обработки сервисов WEB

К ним относятся:

– SOAP протокол для определения форматов запросов к сети;

– WSDL - язык сервиса обмена данными среды .NET и развития сетевых служб .Net Remoting;

– UDDI (Universal Description, Discovery and Integration) для регистрации, описания, хранения, поиска в реестре;

– BPMN графический язык для нотации прикладных и бизнес- процессов типа UML; BPEL язык описания бизнес процессов;

– SOA (Service-oriented Architecture) модель сервисно-ориентированной архитектуры программных систем;

– SCA (Service-component Architecture) модель создания сложных систем на основе сервисов и компонентов;

– XML язык разметки, описания структуры и способов взаимодействия компонентов;

– HTML язык разметки документов и гипертекстов для любых Web-браузеров и редакторов.

– MathML (Mathematical Markup Language) — язык математических формул на основе XML.

В настоящее время в Интернет накоплено огромное количество сервисных компонентов, которые представляются средствами моделей SOA (Service Oriented Architecture) и SCA (Service Component Architecture). Интерфейс сервисных компонентов задается в языке WSDL См. статья:

Е. М. Лаврищева, Л. Е. Карпов, А. Н. Томилин. Подходы к представлению научных знаний в Интернет науке. Сб. XIX Всероссийский научной конференции «Научный сервис в сети Интернет», Новороссийск, 18-23 сентября 2017.-с.310-326.

Компонентные модели SOA

Модель SOA задают средства описания некоторой функции (Function) с заданным качеством сервиса (Quality service) в IT-стандартах комитета W3C, а также reuses, сетевые сервисы, объекты планирования, доступа к БД и к системе. Они обрабатываются EJB сервером приложений J2EE и могут собираться в Веб систему путем конфигурационной сборки с использованием моделей:

M_{wsys}, интерфейсов взаимодействия I_{on} клиент-серверной архитектуры с web-сервером и web-клиентом через запросы. Сервисные компоненты ГОР описываются в ЯП (XML, SOAP, UDDI, WSDL, BPREL, BPMN и др.) сред J2EE, .Net, Appach, JAVA и др. Сервисы должны верифицироваться, и тестироваться и и накапливаются в библиотеках для последующего применения. Обмена метаданными (Metadata Exchange Endpoints) осуществляет сервер и клиент в виде прокси-классов. SOA обеспечивает согласованность, языковую независимость взаимодействия компонентов в ЯП (Java, Python, Ruby и др.) с серверной и клиентской частью системы. Сетевые службы проводят прием, поиск, вызов сервисов и выполнению запросов клиентов через реестр (каталог) служб библиотеки сервисов.

Сервисные компоненты модели SCA (IBM Sphere)

Модель SCA предназначена для работы с прикладными компонентами и включает: EJB сервер приложений J2EE компании Sun Microsystems, доступом к БД и к ИС предприятия (Enterprise Information System, EIS) и др. через аппарат ссылок. Компоненты SCA в IBM WebSphere Integration Developer (WebSphereID) упаковываются в модуль для выполнения сервисного модуля эквивалентного EAR-файлу J2EE и др. Подмодули J2EE и артефакты упаковываются с модулем SCA и позволяет запустить сервис, передать данные для обработки. Система Dynamic Profiles WebSphere Portlet Factory обеспечивает динамическую конфигурацию. Сервер каталогов Tivoli (Tivoli Directory Server) дает доступ к каталогам LDAP (Lightweight Directory Access Protocol) и сохраняет сервис в реестр WebSphere Service Registry and Repository для доступа к клиентам. регистрироваться и выбирать сервисы.

Сервисно-компонетная модель SCM представляет собой обобщение объектно-компонентной модели продуктов (СПП). В ней каждый программный элемент содержит компоненты типа reuses - КПИ, которые обмениваются гетерогенными данными при выполнении системы с помощью механизмов данных SDO и сервисов доступа DAS. При описании сетевых сервисов в языке WSDL вводятся следующие типы:

– строка (xsd:string),

– целые числа (xsd:int, xsd:long, xsd:short, xsd:integer, xsd:decimal),

– числа с плавающей запятой (xsd:float, xsd:double),

– логический тип (xsd:boolean),

– последовательность байтов (xsd:base64Binary, xsd:hexBinary),

– дата и время (xsd: time, xsd:date, xsd:g),

– объекты (xsd:anySimpleType).

Здесь протокол Contract WorkFlow обеспечивает связь с клиентом и и сервером через сообщения, в котром задаются следующие данные в языке XML в WCF:

<?xml version="1.0" ?>
<env:Envelope xmlns:env="
http://www.cbsystematics.com">
<!-Конверт протокола SOAP-> <env:Header>
<!-Заглавие протокола SOAP-> </env:Header> <env:Body>
<!-Тело протокола SOAP-> </env:Body> </env:Envelope>

При написании контрактов WCF атрибутами будут Contract типа (Service, Operation, Fault, Message и DataContract).

На этапе выполнения клиента вызывается метод, определенный в интерфейсе сервиса, WCF сериализует типы CLR и вызов метода в формат XML и посылает сообщение в сеть для привязки к схеме кодировки в WSDL. Со стороны XML задается XSD-описание структуры данных и сообщение осуществляется лишь после того, как будет создан экземпляр XML (XML Instance). Со стороны .NET имеется тип CLR, который определяет структуры данных и функциональные возможности после того, как создан объект такого типа.

Инструменты разработки Веб-систем в Семантик Веб

Описание ГОР в этой среде производится с помощью таких средств (см. статья 6):

RDF стандарта W3C (2004) для описания сетевых, семантических ресурсов и метаданных (данные о данных). Служит каркасом для создания отдельных компонентов семантической паутины. RDFS (англ. *RDF Schema*) — это надстройка над RDF, которая позволяет создавать классы и свойства объектов.

OWL (Web Ontology Language) построен на форматах RDF и RDFS, предназначен для описания онтологий, логики и согласуется с современными сетевыми стандартами.

SPARQL (Protocol And RDF Query Language) — язык запросов для быстрого доступа к данным RDF для получения необходимой информации из сети:

RIF — формат обмена правилами (Rule Interchange Format) и др.;

WSDL – язык описания входных и выходных данных для описания запросов Интернет на сервисы и включает подязыки: WSCI, WSCL, BPMN, BPEL и другие.

В качестве адреса объектов в сети используются универсальные идентификаторы ресурсов URI (Uniform Resource Identifier) и интерфейс, задаваемый для управления связями с другими сервисами через XML-документы.

Для сборки сервисов используется инструмент – *Jopera for Eclipse* (http://www.jopera.ethz.ch/), который обеспечивает:

– композицию сервисов (типа Agile) и визуальный мониторинг отладки композиций сервисов;

– управление изменением интерфейсов сервиса с помощью сообщений об изменениях в сервисе Jopera;

– масштабируемость и автономное исполнение процесса запуска систем с помощью сообщений Jopera.

Jopera предоставляет набор Eclipse-плагинов для связи различных программных элементов и допускает итеративную композицию сервисов (через маршрутизаторы SOAP и RESTful Web-сервис, Grid-сервисы, Java snipets и др.), а также путем моделирования и исполнения процессов в сети. Для поиска сервисов по их семантическим описаниям используются *Feta Client* и *Feta Engine*.

Feta Client – это GUI-плагин системы Интернет Taverna, используемый для описания сервиса, а *Feta Engine* для задания Web-сервиса.

Подключаясь к *Feta Engine*, плагин *Taverna Feta* позволяет:

– конструировать ориентированные на заданную предметную область семантические запросы к нужным сервисам, которые затем отсылаются на *Feta Engine*;

– отображать информацию о результатах выполнения запроса на поиск сервисов;

– интегрировать результирующие сервисы в системе WorkFlow.

Критериями поиска сервисов являются:

– сервис, на входе которого находится элемент семантического или общего типа X;

– сервис, который производится на выходе системы и выдает элемент семантического типа Y;

– сервис, который решает задачу X или еще более конкретную;

– сервис процессора WSDL и др.

– сетевые сервисы стандартной модели OSI, SOA, SCA, как инструменты представления и обработки ресурсов в сети Интернет для реализации деловых, финансовых, экономических и других услуг при решении прикладных задач.

1.7. Конфигурирование ресурсов Веб-систем по стандарту IEEE 829-1996-2012

Под конфигурацией системы понимается структура некоторой ее версии, включающая функции, объединенные между собой операциями связи с параметрами, задающими режимы функционирования системы (см. статья E.M. Lavrischeva, V.S. Mutilin, A.G. Ryzhov. Designing variability models for software, operating systems and their families, 2017, Сборник ИСП РАН, 2017, issue 5, 2017, pp.93-109.- DOI: 10.15514/ISPRAS- 2017(5).

Версия или конфигурация системы согласно IEEE Standard 828-96-2012 (Configuration) включает:

– базис конфигурации – BC (Configuration Baseline);

– элементы конфигурации (Configuration Item);

– компоненты, ГОР, входящие в описание моделей M_{sys}, M_{wsys};

Управление конфигурацией (Configuration Management) заключается в наблюдении за модификацией параметров конфигурации и компонентов системы, а также в проведении систематического контроля, учета и аудита внесенных изменений, поддержки целостности и работоспособности системы. Управление конфигурацией согласно стандарта состоит в выполнении следующих задач:

1. Идентификация конфигурации (Configuration Identification).

2. Контроль конфигурации (Configuration Control).

3. Учет статуса конфигурации (Configuration Status Accounting).

4. Аудит конфигурации аудит (Configuration Audit).

5. Трассировка изменений конфигурации на этапах сопровождения и эксплуатации системы;

6. Верификация компонентных сервисов по моделям M_{sys}, M_{wsys};

7. Доказательство изоморфного отображения данных компонентных сервисов с типами данных стандарта ISO/IEC 11404 Object Data Types -2007 (см. статья 4).

При конфигурационной сборке ГОР используется модели систем M_{sys}, M_{wsys} и модель характеристик *MF* (Model Feature). ГОР и КПИ накапливаются в репозиториях или библиотеках системы. Они отбираются, адаптируются и интегрируются в единую систему. Основную роль в этих процессах выполняет конфигуратор ИТК (http://7dragons.ru/ru). Он обеспечивает сборку разнородных ГОР и их интерфейсов с вариантами отдельных готовых продуктов, которые находятся в репозиториях.

В м*одель среды конфигуратора* входит:

– описание графовой схемы системы из ГОР;

– модели вариантов системы;

– конфигурационную модель ГОР и КПИ;

– операцию конфигурационной сборки КПИ и ГОР;

– аудит конфигурации;

– верификатор моделей и ГОР.

– оценку качества ГОР и систем.

Конфигуратор выполняет оператор config из требуемых ГОР и КПИ и их интерфейсов по заданным моделям в веб-систему и формирует конфигурационный файл варианта прикладной системы, который будет выполняться в соответствующей среде.

1.8. Моделирование варианта ядра ОС Linex для прикладных областей знаний

В данном разделе рассматриваются подход к процессам создания варианта ОС Linux, который будет управлять некоторой новой прикладной областью (например, медицина, биология, геология и др.). Такой подход исследовался в рамках проекта РФФИ и представлен в ряде работ по этому проекту и предлагается процесс создания варианта ОС в виде экспериментального варианта ядра ОС членами коллектива проекта. Созданы статьи по этой теме соисполнителей проекта РФФИ:

Козин. С.В. Конфигурационная сборка варианта ядра Linux для прикладных систем.- Труды ИСП РАН, 2018,- Том 29. Вып.3.-с.

Kozin S.V., Mutilin V.S. Static Verification of Linux Kernel Configurations. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 4, 2017, pp. 217-230. DOI: 10.15514/ISPRAS-2017-29(4)-14

Кулямин В.В., Лаврищева Е.М., Мутилин В.С., Петренко А.К. Верификация и анализ вариабельных операционных систем. Труды Института системного программирования РАН, том 28, вып. 3, 2016, стр. 189-208. DOI: 10.15514/ISPRAS-2016-28(3)-12.

Е.М. Лаврищева, Л. Е. Карпов, А. Н. Томилин. Семантические ресурсы для разработки онтологии научной и инженерной предметных областей, Доклад на конференцib "Научный сервис в сети Интернет" 19-24 сентября 2016, Абрау-16.

В защищенной магистерской работе Газизулиной Р.К. («Современные подходы и методы моделирования изменяемых программных систем», МФТИ, 2017) представлен метод искусственного интеллекта Variability Mining для ОС в рамках системы LEADT. Благодаря этой системы найдены и извлечены фрагменты кода функций ядра ОС и определены их характеристики в модели MF и Msys.

В литературе определены основные положения моделирования ОС из системных и сервисных ресурсов Интернет с использованием методов представления и извлечения знаний в Семантик Веб и процесса конфигурационной сборки варианта ОС в классе прикладных систем.

Операционная система OS Linux – это совокупность программных фрагментов функции и специальных функций ядра OS Linux, управляющих задачами, процессами и прикладными системами. Такие функции могут формировать самостоятельные компоненты с заданным поведением и способствовать общему функционированию варианта ядра ОС. OS.C Linux можно условно разделить на два уровня, см. работу Козина В.А. магистранта МФТИ и Рис. 5.

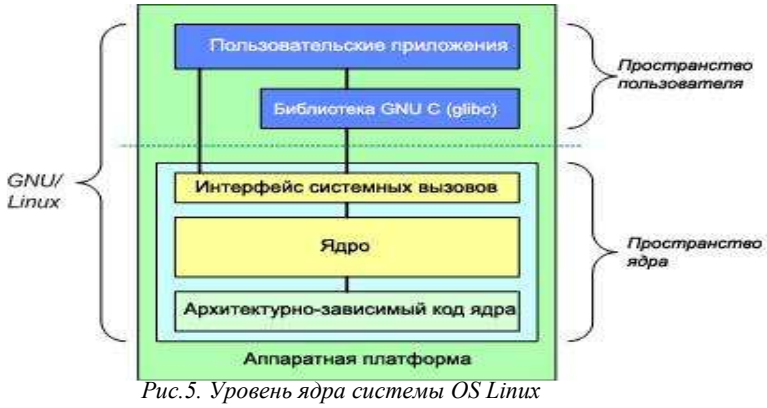


Рис.5. Уровень ядра системы OS Linux

На верхнем уровне находится пользовательское пространство, где исполняются приложения пользователей. Под этим пространством располагается пространство ядра. Имеется библиотека GNU C (glibc), содержащая интерфейс системных вызовов для связи с ядром. Ядро и пользовательское приложение располагаются в разных защищенных адресных пространствах. Каждый процесс в пространстве пользователя имеет свое собственное виртуальное адресное пространство. Ядро OS Linux можно, в свою очередь, разделить на три больших уровня. Наверху располагается интерфейс системных вызовов, который реализует базовые функции (чтение и запись). Ниже интерфейса располагается архитектурно-независимый код ядра. Этот код является общим для всех процессорных архитектур, поддерживаемых Linux. Еще ниже располагается архитектурно-зависимый код, образующий. BSP (Board Support Package), который зависит от процессора и платформы для конкретной архитектуры.

OS Linux можно откомпилировать для огромного количества разных процессоров и платформ, имеющих разные архитектурные ограничения и потребности. Например, Linux может работать на процессоре как с блоком управления памятью (MMU)и без MMU. Поддержка процессоров без MMU реализована в ядре OS Clinux. К основным компонентам ядра Linux относятся (рис. 6):

– интерфейс системных вызовов, как тонкий уровень вызова функций ядра;

– управление процессами или потоками с помощью интерфейса программирования приложений (API) через SCI для создания планировщика и оценки времени от числа потоков;

Рис.6. Компоненттыядра ОС linux

– управление физической и виртуальной памятью и их соответствием;

– определение виртуальной файловой системой (VFS), которая предоставляет общую абстракцию интерфейса к файловым системам и служит для коммутации между SCI и файловыми системами ядра.

– сетевой стек имеет многоуровневую структуру самих протоколов. Internet Protocol (IP) - это базовый протокол сетевого уровня, располагающийся ниже транспортного протокола Transmission Control Protocol, TCP). Выше TCP находится уровень сокетов, вызываемый через SCI. Уровень сокетов представляет собой стандартный API к сетевой подсистеме. Он предоставляет пользовательский интерфейс к различным сетевым протоколам4

– драйверы устройств обеспечивают возможность работы с конкретными аппаратными устройствами.

В OS Linux работает гипервизор, с помощью которого можно строить ОС для других систем. Был реализован интерфейс, позволяющий исполнять поверх его ядра другие ОС.

Моделированию варианта OS Linux

Моделирование любой ОС - это процесс определения архитектуры, компонентов, интерфейсов, других характеристик системы и конечного состава программного продукта. Базовая концепция проектирования ПО - это методология проектирования архитектуры с

помощью разных методов (объектного, компонентного и др.), процессы ЖЦ (стандарт ISO/IEC 12207) и техники - декомпозиция, абстракция, инкапсуляция и др.

Ключевыми вопросами проектирования ПС является: декомпозиция программ на функциональные компоненты для независимого и параллельного их выполнения, принципы распределения компонентов в среде выполнения и их взаимодействия между собой, механизмы обеспечения качества и живучести системы и др.

При разработке ОС на базе Linux основным параметром, определяющим необходимый функционал, является область применения этой ОС. Функциональные элементы ОС Linux отвечают за определенную функциональную деятельность, а именно:

– видеопамять и работа с ней;

– HAL (Hardware Abstraction layer) - поддержка нескольких аппаратных архитектур, включая процессы, семафоры и.др.:

– управление памятью;

– управление исполнением управление устройствами;

– виртуальные файловые системы;

– API (Application Programming Interface) IPC (Interprocess Communication) и т.д.

В маленьких встроенных системах (сайтах), не будут использоваться функциональные модули, отвечающие за видеопамять, API или управление исполнением.

Процессы определения варианта ядра ОС Linux

Используется 6 процессов **следующего вида**:

1) **Конфигурирование архитектуры ОС.** Во время сборки Linux проверяет файл конфигурации Kconfig на наличие рекурсивных зависимостей и несуществующих переменных в коде

2) **Поиск «мертвого кода»** т.е. такого кода, в котором контроль не передается ни при каких обстоятельствах.

3) **Препроцессирование –** предварительная обработка элементов функции из ядра ОС, которое проводится перед компиляцией для получения кода версии программы или системы. Если находятся ошибки, то выдается соответствующая информация для мсправления.

4) **Компиляция** функции состоит в выдаче кода для нее или сведений о случайных ошибках при обнаружении неописанной переменной функции, отсутствие пунктуации в коде и т. д.

5) **Связывание** отдельных элементов ядра выполняет оператор Link. Он находит ошибки задания связи компонентов в интерфейсе или ошибки вызова компонентов из внешних библиотек.

6) **Обработка** ошибочных ситуаций.

7) **Определение конфигурации** архитектуры ОС проводится с целью разделения пространства элементов на классы эквивалентности и использования критерия охвата тестирования MC / DC с помощью двух основных понятий: *решение* и *условие*. Решение - это формула, состоящая из условий и передач управления блоку с решением. Условие - это логическая часть решения, которое соединяется с другими условиями. Каждое условие может влиять на итоговое значение принимаемого решения, фактически изменяет значение независимо от других условий.

8) **Тестирование** собранного варианта ОС для обработки ошибок с помощью LDV и CPAchecker. Инструмент LDV ставит метки кода в соответствии с заданными правилами, CPAchecker проверяет доступность меток и правильность выполнимости компонентов.

1.8.1. Описание эксперимательного варианта ядра OS Linux

Создание некоторого варианта ОС для класса прикладных систем (медицины, биологии и др.) основывается на анализе базовых функций ядра ОС и выбора из множества

компонентов ОС наиболее подходящих для оперативного управления прикладными системами. Исходя из публикаций установлено, что OS Linux содержит более 10 000 переменных и большое множество функциональных системных компонентов, обеспечивающих обработку разного рода заданий по функционированию любых прикладных систем. На ее основе выбраны необходимые компоненты ОС и создана модель MF с базовыми характеристическими компонентами ОС и модели системы *Msys,* включая множество функциональных M_{fi}, интерфейсных M_{io} и M_d работы с данными базового ядра ОС. Эти множества компонентов тестируются на правильность их идентификации, на наборах тестов и операций установления связей с соответствующими компонентами других множеств. После тестирования проводится операция *config* (M_{fi}, MI_{oi}, M_{di}) для получения конфигурационного файла варианта ОС.

Процесс формирования варианта ОС проводится с учетом параметров ядра для компонентов их реализации:

- общей настройки,
- поддержки загружаемых модулей и блоков,
- сетевых настроек,
- драйверов устройств и файловых систем,
- параметров безопасности и криптографии,
- библиотечных процедур, компонентов и др.

При создании конфигурации ядра варианта ОС используются эти параметры в зависимости от области предназначения прикладной системы. Например, ядро может включать в себя множество опций безопасности исходя из стека SELinux Национального агентства по безопасности NSA, ориентированных на безопасность функциональных компонентов OS.

Перед проведением конфигурационной сборки варианта системы OS проверяется файл /etc /udev /rules.d/70-persistent-net.rules и определяются имена сетевых устройств, а также схема именования правилами Udev. Выходной файла имеет блок комментариев, за которым следуют две строки для каждого сетевого адаптера. Первая строка сетевой карты - это описание и комментарии, показывающие идентификаторы оборудования и устройств согласно карты PCI и драйверов.

При задании имени интерфейса идентификатор аппаратного обеспечения не используются. Вторая строка - это правило Udev, которое соответствует сетевой карте с фактически присвоенным именем.

Некоторое программы ПО OS могут устанавливаться с помощью символических ссылок /dev /cdrom и /dev/dvd для устройства CD-ROM или DVD-ROM. Кроме того, могут помещаться символические ссылки в /etc/fstab. Udev в зависимости от возможностей каждого устройства.

Сценарий может работать в режиме «by-path» по умолчанию для устройств USB и FireWire, создаваемые правила которых зависят от физического пути к устройству CD или DVD. Сценарий может работать в режиме «по-id» (по умолчанию для устройств IDE и SCSI) и зависит от правил идентификационных строк, хранящихся на устройстве CD или DVD. Физический путь к устройству (порты и / или слоты) изменяются, например, при планировании перемещения диска в другой порт IDE или другой разъем USB режима «by-id».

Для каждого устройства находится соответствующий каталог в разделе / sys / class или / sys / block и для видеоустройств в разделе / sys / class / video4linux / videoX.

Интерфейсы сетевых сценариев задаются в файле / etc / sysconfig /. При этом файл ifconfig содержит настраиваемый интерфейс ifconfig.xyz в сетевой карте с именем eth0. Внутри этого файла содержатся атрибуты интерфейса IP-адрес, маски подсети и т. д. Приводится спецификация выходного файла в ЯП.

```
cat > /etc/inittab << "EOF"
# Begin /etc/inittab
id:3:initdefault:
si::sysinit:/etc/rc.d/init.d/rc S
l0:0:wait:/etc/rc.d/init.d/rc 0
l1:S1:wait:/etc/rc.d/init.d/rc 1
l2:2:wait:/etc/rc.d/init.d/rc 2
l3:3:wait:/etc/rc.d/init.d/rc 3
l4:4:wait:/etc/rc.d/init.d/rc 4
l5:5:wait:/etc/rc.d/init.d/rc 5
l6:6:wait:/etc/rc.d/init.d/rc 6
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
su:S016:once:/sbin/sulogin
1:2345:respawn:/sbin/agetty --noclear tty1 9600
2:2345:respawn:/sbin/agetty tty2 9600
3:2345:respawn:/sbin/agetty tty3 9600
4:2345:respawn:/sbin/agetty tty4 9600
5:2345:respawn:/sbin/agetty tty5 9600
```

```
6:2345:respawn:/sbin/agetty tty6 9600
# End /etc/inittab
EOF
```

С общей точки зрения модель OS Linux включает множество разных операционных артефактов (функций) и интерфейсов между ними:

$M_{oc} = (C_k, M_f, M_s, M_o, M_i)$, где

C_k – множество отдельных фрагментов, артефактов ОС;

M_s – множество характеристик ядра Linux (более 10000 для версии 4.11);

M_o – множество ограничений характеристик функций на глубине дерева зависимостей (глубина 8 для 22 ограничений);

M_i – множество интерфейсных характеристик (menuconfig) в ядре Linux.

Модель вариабельности OS согласно проведенных исследований и обсуждений на конференциях VAMOS получила следующее представление:

$MFvar = (SV_{inoc}; AV_{inoc})$, где

SV_{inoc} – подмодель вариабельности артефактов структуры ядра ОС Linux;

AV_{inoc} – подмодель вариабельности продукта ОС Linux.

Модель MF_{var} задает изменяемость отмеченных артефактов, определяет затраты и уменьшает стоимость варианта системы.

Подмодель $SV_{inoc} = ((G_{inoc}, TR_{inoc}), Con_{inoc}, Dep_{inoc})$,

где $G_{inoc} = (F_{inoc}, LF_{inoc})$ – граф артефактов ядра ОС Linux;

TR_{inoc} – набор связей между зафиксированными артефактами ОС Linux;

Con_{inoc} и Dep_{inoc} – предикаты ограничений и зависимостей между отдельными функциями на множестве артефактов ОС Linux.

AV_{inoc} определяет изменяемость структуры ОС из готовых ресурсов, которые хранятся в БД системы. Эта подмодель отображает характеристики артефактов и отношений между ними на дереве зависимостей. Точки вариантности, отмечающие изменяемые артефакты, обрабатываются конфигуратором и способствуют получению варианта ОС для конкретного применения.

Практическими экспериментами в системе ОС Linux определен набор функций и их характеристик. При генерации некоторого варианта ОС требуется извлечь из ядра системы необходимые готовые артефакты или функциональные элементы с их интерфейсами.

1.8.2. Конфигурирование архитектуры систем и ОС в Grid, Etics, BCD

Сборкой (англ. build) называется процесс получения продукта из исходного кода. Чаще всего включает компиляцию и компоновку, выполняется инструментами автоматизации.

В ходе сборки ядра Linux применяется утилита make, использующая специальные make-файлы, в которых указаны зависимости файлов друг от друга и правила для их удовлетворения. На основе информации о времени последнего изменения каждого файла make определяет и запускает необходимые программы. В данном случае make выступает в роли инструмента управления сборкой, относящегося к инструментам конфигурационного управления.

Можно использовать ряд стандартных операторов в системе Java, BSD, для конфигурации варианта ядра ОС операциями:

– config - традиционный способ конфигурирования. Программа выводит параметры конфигурации по одному, предлагая установить для каждого из них свое значение.

– oldconfig - файл конфигурации создаётся автоматически, основываясь на текущей конфигурации ядра.

– defconfig - файл конфигурации создаётся автоматически, основываясь на значениях по умолчанию.

– menuconfig - псевдографический интерфейс ручной конфигурации, не требует последовательного ввода значений параметров. Рекомендуется для использования в терминале.

– xconfig - графический (X, QT) интерфейс ручной конфигурации, не требует последовательного ввода значений параметров.

– gconfig - графический (GTK+) интерфейс ручной конфигурации, не требует последовательного ввода значений параметров. Рекомендуется для использования в среде GNOME и др.

Стандартный путь такого конфигурирования – использование утилит make с параметром, зависящим от выбранного инструмента конфигурации. Например, если выбрана конфигурация через config, то осуществляется команда make config. После выполнения этой команды запускается сборка с помощью make. В случае установки ОС на многоядерную машину можно задать количество потоков с тем, чтобы ускорить процесс сборки.

При разработке варианта ОС на базе Linux основным параметром, определяющим необходимый функционал, является область применения этой ОС.

Функциональные элементы OS Linux отвечают за определенную функциональную деятельность, а именно:

– видеопамять и работу с ней;

– HAL (Hardware Abstraction layer) — поддержку нескольких аппаратных архитектур, включая процессы, семафоры и др.;

– управление памятью;

– управление исполнением управления устройствами;

– виртуальные файловые системы;

– API (Application Programming Interface), IPC (Interprocess Communication) и т. п.

В маленьких встроенных системах (сайтах, приложениях) могут не использоваться функциональные модули, отвечающие за видеопамять, API, сценарии и др.

К настоящему моменту сформировались стандарты, с помощью которых можно проводить сборку вариантов ОС семейства Linux. К ним относятся стандарты**:**

- POSIX.1-2008.
- File system Hierarchy Standard (FHS) Version 3.0.
- Linux Standard Base (LSB) Version 5.0 (2015).

POSIX (англ. *Portable Operating System Interface* — переносимый интерфейс операционных систем) — набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой (системный API), библиотеку языка C и набор

приложений и их интерфейсов. Стандарт создан для обеспечения совместимости различных UNIX-подобных операционных систем и переносности прикладных программ на уровне исходного кода, но может быть использован и для не-Unix систем.

FHS (англ. *Filesystem Hierarchy Standard*, «стандарт иерархии файловой системы») — стандарт, унифицирующий местонахождение файлов и каталогов с общим назначением в файловой системе UNIX. На данный момент большинство UNIX-подобных систем в той или иной степени следует этим правилам. Например, обычная база данных о пользователях всегда хранится в файле /etc/passwd.

Linux Standard Base, LSB — совместный проект семейства операционных систем, основанных на Linux (то есть дистрибутивов Linux), при организации Linux Foundation, целью которого является стандартизация их внутренней структуры. LSB опирается на существующие спецификации, такие как POSIX, Single UNIX Specification, и другие открытые стандарты, при этом расширяя и дополняя их.

Для удовлетворения стандарта LSB будут использоваться пакеты:

LSB Core: Bash, BC, Binutils, Coreutils, Diffutils, File, Findutils, Gawk, Grep, Gzip, M4, Man-DB, Ncurses, Procps, Psmisc, Sed, Shadow, Tar, Util-linux, Zlib, *SB Runtime Languages: Perl.*

Вариант ОС строится на базе уже установленного дистрибутива Linux (такого как Debian, OpenMandriva, Fedora или openSUSE). Система Linux (хост) является отправной точки для предоставления необходимых программ, включая компилятор, компоновщик и оболочку, для создания нового варианта ОС.

При этом в хосте установлены следующие пакеты:

Bash-3.2, Binutils-2.25**,** Bison-2.7**,** Bzip2-1.0.4, Coreutils-6.9, Diffutils-2.8.1, Findutils-4.2.31, Gawk-4.0.1, GCC-4.9, включая C++ compiler, g++, Glibc-2.11**,** Grep-2.5.1a, Gzip-1.3.12, Linux Kernel-3.2, M4-1.4.10, Make-4.0, Patch-2.5.4, Perl-5.8.8, Sed-4.1.5, Tar-1.22, Texinfo-4.7, Xz-5.0.0.

Затем требуется скомпилировать цепочку, состоящую из инструментов компиляции LFS, таких как GCC, Glibc, Binutils и другие утилиты. Затем корневой каталог изменяется с помощью chroot и потом начинается сборка окончательного варианта ОС. Один из первых пакетов для компиляции — Glibc; после этого компоновщик toolchain должен быть настроен для привязки к недавно созданному glibc, так что все другие пакеты, которые будут составлять готовую систему, также могут быть связаны с ним. Во время фазы chroot функция хэширования bash отключается, а временная директория bin toolchain перемещается в конец PATH. Таким образом, сначала скомпилированные программы появляются в PATH, а новая система основывается на собственных новых компонентах.

Для создания экспериментального ядра ОС используется свободный пустой дисковый раздел OS Linux (около 6 гигабайт — ГБ) и хранения всех исходных архивов и компиляции пакетов. При настройке жесткого диска для использования системой Linux наиболее удобной установкой является начальная загрузка жесткого диска на диск хоста. Если на хосте уже есть один IDE-диск, который рассматривается как /dev/hda, например, IDE-диск может рассматриваться как hdb или hdc, в зависимости от настройки хоста. Затем мы можем форматировать и монтировать этот диск, как и на любом другом жестком диске. Единственное различие, однако, в том, что целевой диск, увиденный на хосте как вторичный диск, такой как /dev/hdb или /dev/hdc, скорее всего будет рассматриваться как hda. Это создает определенные проблемы при настройке загрузчиков. Кроме того, требуется достаточное свободное временное хранилище, и дисковое пространство для компиляции пакетов, например, рекомендуется использовать небольшой раздел диска для пространства подкачки -swap. Раздел подкачки для новой системы может быть таким же, который используется хост-системой.

Подкачка (swap) является важным компонентом большинства рабочих станций Linux и серверов. Это позволяет системе обращаться к большему количеству памяти, чем физически доступно, путем эмуляции дополнительной памяти на запоминающем устройстве. Однако большинство встроенных запоминающих устройств, таких как устройства flash и

DOC-девайсы, плохо адаптированы к этому использованию, поскольку они имеют ограниченные циклы стирания и записи. Так как созданное приложение мало контролируется, то необходимо уменьшить использование памяти приложений и иметь минимальный набор двоичных файлов.

Первоначально диск разбивается с помощью специальной утилиты, например, cfdisk или fdisk (жесткий диск), на котором будет создан новый раздел, например, /dev /sda для основного диска Integrated Drive Electronics (IDE).

После создания этого раздела создается файловая система, например, ext3 и ext4. Дадим характеристику файловых систем:

– ext2 — система для небольших разделов, которые обновляются нечасто и разбивается на дополнительные участки диска;

– ext3 — это апгрейд для ext2, который включает журнал для восстановления статуса раздела после отключения;

– ext4 — это версия семейства файлов ext, включающей несколько новых возможностей, например, наносекундные временные метки, создание и использование очень больших файлов (16 ТБ) и улучшение скорости.

Критерием для правильного выбора является список типов данных, хранящихся в рассматриваемой файловой системе, и их предполагаемое использование: для примера, исполняемые двоичные файлы и библиотеки, которые формируют программное обеспечение, использующее устройство, и файлы XML, содержащие переменные конфигураций.

Затем определяется комбинация типа файловой системы и устройства хранения данных, соответствующие требованиям.

Для получения нового ядра ОС необходимо, чтобы USB-драйверы (ehci_hcd, ohci_hcd и uhci_hcd) вводились как модули, загружаемые в правильном порядке, и модуль ehci_hcd загрузки ohci_hcd и uhci_hcd, чтобы избежать появления ошибочных ситуаций во время загрузки.

GRUB работает, записывая данные на первую физическую дорожку жесткого диска. Эта область не является частью какой-либо файловой системы. Программы получают доступ к модулям GRUB в загрузочном разделе — /boot/grub/.

Загрузочный раздел ОС надо выбрать (рекомендуемый размер — 100 МБ), загружать информацию для проведения конфигурационной сборки варианта ядра.

1.8.3. Апробация и создание экспериментальногоь варианта OS Linux (toolchain)

Раздел OS Binutils устанавливается первым и как Glibc выполняют различные функциональные тесты на ассемблере и компоновщике, чтобы определить, какие программные функции включены или отключены. Неправильно сконфигурированные GCC или Glibc могут привести к неочевидной поломке инструментальной цепочки, где такое воздействие может не выявиться до конца сборки всего дистрибутива. Сбой тестового набора позволяет выявить ошибки, прежде чем выполняется большая работа по установке некоторых функций ОС.

Раздел OS Binutils устанавливает свой ассемблер и компоновщик и устанавливаются заголовки API Linux. Это позволяет стандартной библиотеке Glibc взаимодействовать с функциями, которые предоставляет ядро OS Linux.

Для построения Glibc используется компилятор, бинарные инструменты и заголовки ядра. При этом Glibc использует компилятор, относящийся к параметру — host, переданному скрипту configure (например, компилятор будет i686-lfs-linux-gnu-gcc).

Бинарные инструменты и заголовки ядра более сложные в установке. После запуска *configure* проверяется содержимое файла config.make в каталоге glibc-build. Эти элементы являются важными и придают Glibc самодостаточность с точки зрения механизмов сборки. Во время вторичной установки Binutils используется переключатель конфигурации — with-lib-path для управления пути поиска библиотеки ld. Для вторичной установки GCC

необходимо внести изменение и заставить GCC использовать новый динамический компоновщик. Невыполнение этого требования приведет к тому, что сами программы GCC будут иметь имя динамического компоновщика из встроенного в них каталога хост-системы /lib, что приведет к поражению цели уйти от хоста. С этого момента основная инструментальная цепочка является самодостаточной и самостоятельной.

Для владения каталогом инструментов и всеми файлами и можно добавить пользователя lfs в новую систему LFS при создании файла /etc/passwd на хост-системе.

Создание нового варианта ядра OS Linux

Для запуска ядра ОС требуется наличие нескольких узлов устройств, в частности консольных и null-устройств. Узлы устройств должны быть созданы на жестком диске, чтобы они были доступны до запуска udevd, а также при запуске Linux с init = /bin/bash. Методом заполнения каталога /dev устройствами является подключение виртуальной файловой системы (например, tmpfs) в каталоге /dev и возможность динамического создания устройств в этой виртуальной файловой системе. Создание устройства выполняется во время загрузки Udev. Поскольку эта новая система еще не имеет Udev, то вручную устанавливается и заполняется /dev. Это достигается связыванием мониторинга каталога хоста /dev.

Bind mount — это особый тип монтирования, который позволяет создавать зеркало каталога или точки монтирования в каком-либо другом месте.

После этого необходимо войти в среду chroot, чтобы начать сборку и установку окончательного варианта системы LFS. Каталоги по умолчанию создаются с доступом 755, но это нежелательно для всех каталогов. В приведенных выше командах сделаны два изменения: один — для каталога пользователя root, а другой — для временных файлов.

Первое изменение режима гарантирует, что кто попало в /root. Второе изменение режима гарантирует, что любой пользователь может писать в директории /tmp и /var/tmp, но не может удалять из него файлы другого пользователя (так называемым «sticky bit»).

Дерево каталогов основано на Filesystem Hierarchy Standard (FHS). FHS также указывает необязательное существование некоторых каталогов, таких как /usr/local/games и /usr/share/games.

Некоторые программы используют пути к программам, которые еще не существуют. Чтобы эти программы заработали, необходимо создать ряд символических ссылок, которые будут заменены реальными файлами по ходу установки.

OS Linux поддерживает список смонтированных файловых систем в файле /etc/mtab. Современные ядра поддерживают этот список внутренне и предоставляют его пользователю через файловую систему /proc. Чтобы удовлетворить утилиты, ожидающие наличия /etc/mtab, нужно создать соответствующий Symlink.

Чтобы root мог войти в систему по имени «root», должны быть соответствующие записи в файлах /etc/passwd и /etc/group.

В соответствии с документацией устанавливаются следующие пакеты:

Linux-4.18.5 API Headers, Man-pages-4.16, Glibc-2.28, Zlib-1.2.11, File-5.34, Readline-7.0, M4-1.4.18, Bc-1.07.1, Binutils-2.31.1, GMP-6.1.2, MPFR-4.0.1, MPC-1.1.0, Shadow-4.6, GCC-8.2.0, Bzip2-1.0.6, Pkg-config-0.29.2, Ncurses-6.1, Attr-2.4.48, Acl-2.2.53, Libcap-2.25, Sed-4.5, Psmisc-23.1, Iana-Etc-2.30, Bison-3.0.5, Flex-2.6.4, Grep-3.1, Bash-4.4.18, Libtool-2.4.6, GDBM-1.17, Gperf-3.1, Expat-2.2.6, netutils-1.9.4, Perl-5.28.0, XML::Parser-2.44, Intltool-0.51.0, Autoconf-2.69, Automake-1.16.1, Xz-5.2.4, Kmod-25, Gettext-0.19.8.1, Libelf-0.173, Libffi-3.2.1, OpenSSL-1.1.0i, Python-3.7.0, Ninja-1.8.2, Meson-0.47.1, Procps-ng-3.3.15, E2fsprogs-1.44.3, Coreutils-8.30, Check-0.12.0, Diffutils-3.6, Gawk-4.2.1, Findutils-4.6.0, Groff-1.22.3, GRUB-2.02, Less-530, Gzip-1.9, IPRoute2-4.18.0, Kbd-2.0.4, Libpipeline-1.5.0, Make-4.2.1, Patch-2.7.6, Sysklogd-1.5.1, Sysvinit-2.90, Eudev-3.2.5, Util-linux-2.32.1, Man-DB-2.8.4, Tar-1.30, Texinfo-6.5, Vim-8.1.

Для получения нового ядра ОС необходимо, чтобы USB-драйверы (ehci_hcd, ohci_hcd и uhci_hcd) вводились как модули, загружаемые в правильном порядке, и модуль ehci_hcd

загрузки ohci_hcd и uhci_hcd, чтобы избежать появления ошибочных ситуаций во время загрузки.

GRUB работает, записывая данные на первую физическую дорожку жесткого диска. Эта область не является частью какой-либо файловой системы. Программы получают доступ к модулям GRUB в загрузочном разделе — /boot/grub/.

Загрузочный раздел ОС надо выбрать (рекомендуемый размер — 100 МБ), загружать информацию для проведения конфигурационной сборки варианта ядра.

Ниже приводится пример для корневого отдельного загрузочного раздела — sda2.

Файл GRUB устанавливается в /boot/grub в виде загрузочного трека /boot/grub/grub.cfg. На нем создан варианта ядра OS Linux (см. рис. 7):

boot/grub/grub.cfg

```
# Begin /boot/grub/grub.cfg
set default=0
set timeout=5
insmod ext2
set root=(hd0,2)
menuentry "GNU/Linux, Linux 4.18.5-lfs-8.3" {
  linux /boot/vmlinuz-4.18.5-lfs-8.3 root=/dev/sda2 ro
}
```

etc/modprobe.d/usb.conf

```
# Begin /etc/modprobe.d/usb.conf
install ohci_hcd /sbin/modprobe ehci_hcd ; /sbin/modprobe -i ohci_hcd ; true
install uhci_hcd /sbin/modprobe ehci_hcd; /sbin/modprobe -i uhci_hcd; true
# End /etc/modprobe.d/usb.conf
```

etc/fstab

```
cat > /etc/fstab << "EOF"
# Begin /etc/fstab
# file system mount-point type  options    dump fsck
#                order
/dev/<xxx>  /   <fff> defaults   1  1
/dev/<yyy>  swap   swap  pri=1    0  0
Proc   /proc  proc  nosuid, noexec, nodev 0  0
sysfs   /sys   sysfs nosuid,noexec,nodev 0  0
devpts   /dev/pts  devpts gid=5,mode=620  0  0
tmpfs   /run   tmpfs defaults   0  0
devtmpfs  /dev   devtmpfs mode=0755,nosuid 0  0
# End /etc/fstab
EOF
```

etc/shells

```
# Begin /etc/shells
/bin/sh
/bin/bash
# End /etc/shells
```

etc/inputrc

```
# Begin /etc/inputrc
# Modified by Chris Lynn <roryo@roryo.dynup.net>
# Allow the command prompt to wrap to the next line
set horizontal-scroll-mode Off
# Enable 8bit input
set meta-flag On
```

```
set input-meta On
# Turns off 8th bit stripping
set convert-meta Off
# Keep the 8th bit for display
set output-meta On
# none, visible or audible
set bell-style none
# All of the following map the escape sequence of the value
# contained in the 1st argument to the readline specific functions
"\eOd": backward-word
"\eOc": forward-word
# for linux console
"\e[1~": beginning-of-line
"\e[4~": end-of-line
"\e[5~": beginning-of-history
"\e[6~": end-of-history
"\e[3~": delete-char
"\e[2~": quoted-insert
# for xterm
"\eOH": beginning-of-line
"\eOF": end-of-line
# for Konsole
"\e[H": beginning-of-line
"\e[F": end-of-line
# End /etc/inputrc
```

etc/sysconfig/clock

```
# Begin /etc/sysconfig/clock
UTC=1
# Set this to any options you might need to give to hwclock,
# such as machine hardware clock type for Alphas.
CLOCKPARAMS=
# End /etc/sysconfig/clock
```

etc/inittab

```
# Begin /etc/inittab
id:3:initdefault:
si::sysinit:/etc/rc.d/init.d/rc S
l0:0:wait:/etc/rc.d/init.d/rc 0
l1:S1:wait:/etc/rc.d/init.d/rc 1
l2:2:wait:/etc/rc.d/init.d/rc 2
l3:3:wait:/etc/rc.d/init.d/rc 3
l4:4:wait:/etc/rc.d/init.d/rc 4
l5:5:wait:/etc/rc.d/init.d/rc 5
l6:6:wait:/etc/rc.d/init.d/rc 6
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
su:S016:once:/sbin/sulogin
1:2345:respawn:/sbin/agetty --noclear tty1 9600
2:2345:respawn:/sbin/agetty tty2 9600
3:2345:respawn:/sbin/agetty tty3 9600
4:2345:respawn:/sbin/agetty tty4 9600
5:2345:respawn:/sbin/agetty tty5 9600
6:2345:respawn:/sbin/agetty tty6 9600
```

```
# End /etc/inittab
```

etc/hosts

```
# Begin /etc/hosts
127.0.0.1 localhost
127.0.1.1 <FQDN> <HOSTNAME>
<192.168.1.1> <FQDN> <HOSTNAME> [alias1] [alias2 ...]
::1  localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
# End /etc/hosts
```

etc/sysconfig

```
ONBOOT=yes
IFACE=eth0
SERVICE=ipv4-static
IP=192.168.1.2
GATEWAY=192.168.1.1
PREFIX=24
BROADCAST=192.168.1.255
```

etc/passwd

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/dev/null:/bin/false
daemon:x:6:6:Daemon User:/dev/null:/bin/false
messagebus:x:18:18:D-Bus Message Daemon User:/var/run/dbus:/bin/false
nobody:x:99:99:Unprivileged User:/dev/null:/bin/false
```

etc/group

```
root:x:0:
bin:x:1:daemon
sys:x:2:
kmem:x:3:
tape:x:4:
tty:x:5:
daemon:x:6:
floppy:x:7:
disk:x:8:
lp:x:9:
dialout:x:10:
audio:x:11:
video:x:12:
utmp:x:13:
usb:x:14:
cdrom:x:15:
adm:x:16:
messagebus:x:18:
systemd-journal:x:23:
input:x:24:
mail:x:34:
nogroup:x:99:
users:x:999:
```

Рис.7. Описание базовых функций варианта ядра Linux

Полученный вариант OS Linux имеет все стандартные функции ядра Linux:

– функцию сжатия данных, файловую систему, командную строку, функции арифметического вычисления рациональных чисел (включая числа с плавающей точкой);

– функции чтения аудио-файлов, компилятор языка С и С++, систему защиты паролей, интерфейс установки пакетов pkg, список контроля доступа, текстовый редактор, возможность работы с сетью, поиск по системе;

– библиотеку функций баз данных, генератор хэш функций, XML-парсер, openSSL и некоторые другие вспомогательные функции.

Приведенный вариант OS Linux может использоваться для класса прикладных систем для предметных областей знаний. Он обладает необходимым набором средств и инструментов ОС для применения, как вариант ядра из семейства OS Linux.

1.9. Описание средств системы LEADT для извлечения знаний

Для извлечения знаний использовался аппарат Variability Mining, реализованный в LEADT. В нем содержатся средства поиска и анализа готовых ГОР в унаследованном коде ОС. Данный инструмент извлекает отдельные характеристики функций в Legacy code системы, восстанавливает архитектуру для практического использования и оформляет их в формате для представления в репозитории системы. Из выявленных функций формируется новый вариант системы, который удовлетворяет требованиям системы. Исследование и пробную реализацию выполнила магистрант Газизулина Р.Д.

Данная система обеспечивает извлечение объектов в унаследованном коде некоторой системы с помощью Variability Mining для:

– установления согласованности индикатора, основанного на изменчивости системы;

– извлечения особенностей характеристик на уровнях детализации продукта;

– использования домена знаний для установления отношений между разными объектами.

Ее основу составляет Variability Mining.

Основу метода Variability Minning составляет задача выпуска нового продукта за счет переиспользования старого кода с помощью полуавтоматических инструментов, позволяющих находить и извлекать требуемую информацию из готового старого Legacy-кода, формировать модель характеристик, документировать ее элементы и по этой построенной модели производить сборку, тестирование и поставку на продажу новых вариантов системы. То есть Variability mining является процессом выявления особенностей в Legacy-коде и перепись их в виде качественных дополнительных (или альтернативных) функций для повторного использования в продуктовой линии (например, Product Line). На основе этих функций создаются варианты или разные версии продукта для разных целей. Извлечение изменчивости (Variability Mining) на линии продукта связано с исследованием функции идентификации, реверс инжиниринга и восстановления архитектуры системы на основе обязательных и необязательных характеристик. Выявленные особенности в Legacy code переписываются в качестве дополнительных (или альтернативных) функций в репозиторий системы. Потом из выявленных функций делаются разные варианты системы. В этом случае оптимизируется производительность системы для передачи клиентам не только полного набора функций в каждом новом варианте системы.

Процесс извлечения и построение системы (The variability mining process consists) состоит из четырех шагов:

1. Экспертное описание релевантных features и их взаимосвязей.
2. Некий инструмент идентифицирует смысл каждой функции в унаследованном коде ядра.
3. Для каждой feature разработчик итеративно расширяет идентифицированный код до тех пор, пока он не станет согласованным и полным.

4. На заключительном этапе разработчики с помощью инструментов переписывают фрагменты кода таким образом, чтобы варианты системы с и без этих фрагментов кода могли быть сгенерированы.

Базовая модель описания выходного кода

Эта модель описывает элементы кода, features и отношения между ними.

Элементы кода. Для их представления и связи между ними используется стандартный статический граф зависимостей. В Java используются методы, поля, локальные переменные, параметры. Для совокупности элементов кода программы – $\boldsymbol{E}$ между этими элементами кода извлекается отношение вида - $\boldsymbol{R \subseteq E \times E}$.

Характеристики- Features. Знания области описываются как совокупность признаков $\boldsymbol{F}$ и отношений между ними, извлеченных из модели. Имеется 2 вида отношений: взаимное исключение ($\boldsymbol{M \subseteq F \times F}$) и импликация ($\Rightarrow$).

Спецификация. Строится карта взаимоотношений между элементами кода и features. Ей соотносят элементы кода и признаки $\boldsymbol{A \subseteq E \times F}$. Разработчик может помечать ее как неполную ($\boldsymbol{N \subseteq E \times F}$), тогда данный фрагмент кода будет отмечен, как не имеющий отношения к данной feature.

Используя введенные выше обозначения, получим $extent(f) = \{e \mid (e,f) \in \boldsymbol{A}\}$, $exclusion(f) = \{e \mid (e,f) \in \boldsymbol{N}\} \cup extent(g), (g,f) \in M$. Все элементы кода, которые не принадлежат *extent(f)*, или *exclusion(f)* являются кандидатами для будущего mining *f*. Моделируется приоритетная рекомендация как совокупность взвешенных ассоциаций элементов с признаками $recommend \subseteq \boldsymbol{E \times F} \times [0,1]$.

Система типов - это ключевой элемент системы и движущий фактор variability mining. Она обеспечивает согласованность, высокую детализацию и встраивание знаний о взаимосвязи между features предметной области. Главная идея заключается в том, чтобы искать ссылки в продукте, аналогично тому, как делает система типов. От вызова функции до ее объявления, от места доступа к переменной до ее объявления. Проверка типов может быть рассмотрена как функция, которая принимает элементы кода $\boldsymbol{E}$, аннотации $\boldsymbol{A}$ и модель вариабельности $\boldsymbol{VM}$ и выдает ряд сообщений об ошибке типа *(e,f)*, которым дается приоритет 1: $recommend_{TS} = typeerrors(E, A, VM) \times \{1\}$.

Топологический анализ

Основная идея топологического анализа заключается в хождении по графу зависимостей от текущей точки ко всем структурным соседним точкам. Алгоритм извлекает приоритеты и ранжирует рекомендации, используя метрику specificity и reinforcement.

Под specificity подразумевается, что элементы, которые относятся только к одному объекту, имеют больший приоритет, чем элементы, относящиеся ко многим.

Под reinforcement подразумевается, что элементы, относящиеся ко многим аннотированным элементам, имеют более высокий приоритет, так как они являются частью кластера feature:

$recommend_{TA} = \{(e, f, w) \mid e \in neighbors(extent(f)), w = weight_{TA}(e, f, extent(f)) - weight_{TA}(e, f, exclusion(f))\}$.

Итоговый результат анализа кода в Variability Mining

Приоритет каждой рекомендации строится на основе приведенных 3-х механизмов variability mining: система зависимостей, система типов, топологический анализ и текстовое сравнение в виде плагина для Eclipse. Это делается следующими процессами системы LEADT:

1. Моделирование features и их взаимосвязей в CIDE Variability Mining;
2. Ручная аннотация выделенных стартовых точек;

3. Расширение feature кода, следуя рекомендациям для каждой feature. Рекомендации подразделяются на полезные или неполезные. Каждую аннотацию LEADT учитывает построении рекомендаций.

4. Переписывание аннотированного кода, проводится в CIDE (Colored Integrated Development Environment).

Таким образом, на инструментальном средстве LEADT проведен процесс Variability Mining на примере одной Legacy системы для ОС и получена оценка изменчивости интеллектуального подхода с инструментом LEADT. При количественном анализе вариабельности изготовленной системы 97% кода выявлено 19 функций и получено следующее:

а) процесс Variability Mining и инструмент LEADT при полуавтоматическом нахождении документа ресурса дали хороший результат для решения такой задачи как извлечение кода,

б) определен показатель достоверности процесса изменчивости в рамках допустимого,

в) определено расширение существующей техники в домене знаний с детализацией, необходимой для некоторой продуктовой линии и построения ядра выходного кода.

1.10. Средства масштабируемости клиента и сервера для веб-систем и сайтов

В данном РФФИ-352-18 представлены два класса системных компонентов: клиент и сервер с расчетом на использование Больших данных. Этому посвящен доклад и статья на конференции Абрау-18 (Лаврищева Е.М., А.Г.Рыжов. Подход к моделированию систем и сайтов из готовых ресурсов // Научный сервис в сети Интернет: труды XX Всероссийской научной конференции (17-22 сентября 2018г. г. Новороссийск).-ИПМ им. М.В.Келдыша.-с. 321-345.). Множество клиентов и серверов в среде Интернет определяются на новых средствах коммуникации и взаимодействия. Сущность состоит в следующем. С учетом возрастающего количества пользователей сети и большим объемов данных Big Data и методов вычислений Cloud Computing предлагается усовершенствовать клиент–серверную архитектуру в направлении применения множества отдельных серверов для хранения и обработки больших данных и организации вычислений на суперкомпьютерах. Требуется оптимизировать нагрузки на системные ресурсы, сетевое оборудование для более эффективного использования серверов и клиентов сети. Увеличение количества доступных ресурсов предлагается путем вертикального и горизонтального масштабирования. Вертикальное масштабирование увеличивает производительность приложений, систем и веб-систем путем добавления аппаратных ресурсов серверов, а горизонтальное масштабирование для роста производительности систем.

Таким образом, современная клиент–серверная архитектура будет основываться на процессах:

Front-end для клиентских приложений, систем и **Back–end** для обслуживания серверных частей приложений. Для этих процессов предусматривается обеспечивается безопасность, защита и гарантия качественной работы в системе Интернет при создании как отдельных функциональных, системных и веб-сервисных компонентов, так и конфигурации системы.

Веб-сервер для выполнения трудоемких задач

При вполнении трудоемких и медленных задач, сообщение направляется на сервер очередей задачи, а сама операция переносится на отдельный веб-сервер, выполняющий задачи, которые работают независимо от основногоWeb-приложения. (рис. 8).

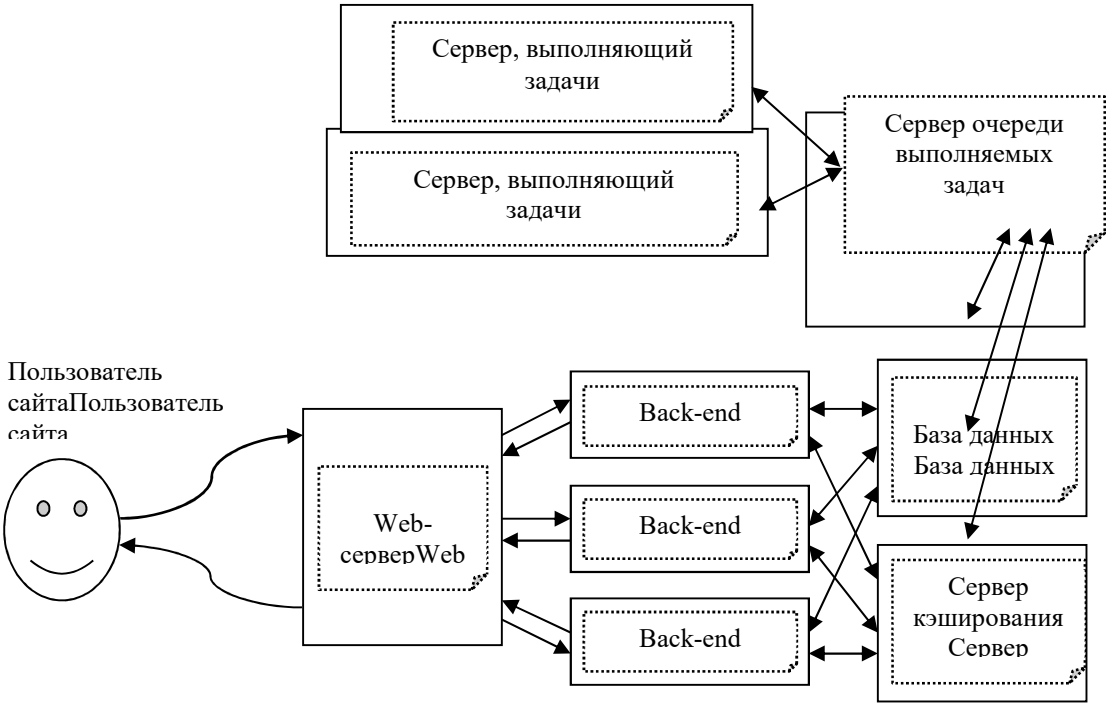


Рис. 8. Структура сайта с множеством серверов для выполнения трудоемких задач

Сервер очереди задач позволяет выполнять трудоемкие задачи асинхронно, передавая их на выполнение дополнительным серверам без замедления Web-приложения. Количество дополнительных серверов увеличивается, когда среднее количество задач в очереди будет постепенно расти.

В Front-end для оптимизации ресурсов, сокращения времени обслуживания запросов и возможности добавления/удаления этих устройств и повышения производительности Web-приложения проводится балансировка загрузки (load balancing) с помощью специального программного обеспечения или аппаратных средств. В случае Back-end необходимо разносить несвязанные данные в разных экземплярах БД, используя разные настройки, технологии и ЯП. В сети создается балансирующий узел, который перенаправляет запрос на один из Back-end серверов. Если этот сервер перестает справляться со всеми многочисленными запросами, следует использовать несколько балансирующих узлов, что повышает надежность и доступность системы. Сервер очереди задач позволяет выполнять трудоемкие задачи асинхронно, передавая их на выполнение вспомогательным серверам и не замедляя Web-приложение. Их количество может увеличиваться, когда среднее количество задач в очереди будет постепенно расти.

1.11. Обеспечение качества ПО, прикладных систем и Веб-систем

Качество - это совокупность свойств (показателей качества) ПО, которые обеспечивают его способность удовлетворять потребности заказчика, в соответствии с его назначением. Оно задано в стандарте ISO/IEC 9126 (1-2) или ГОСТ 2844–98 в виде модели качества с шестью показателями См. Статью 8.

Лаврищева Е. М., Карпов Л. Е., Томилин А. Н. Семантические ресурсы для разработки онтологии научной и инженерной предметных областей// Научный сервис в сети Интернет: труды XVIII Всероссийской научной конференции (19-24 сентября 2016 г., г. Новороссийск). — М.: ИПМ им. М.В. Келдыша, 2016.- с.126-138.- doi:10.20948/abrau-2016-2018.

Стандарт жизненного цикла (ЖЦ) систем ISO/IEC 12207 определил основные процессы ЖЦ разработки систем, организационные и дополнительные процессы, которые регламентируют планирование, управление качеством и оценку затрат на создание проекта. На процессах ЖЦ проводится анализ свойств качества отдельных элементов ПО, ориентированный на:

– достижение их качества в соответствии с требованиями и критериями заказчика и стандарта;

– верификацию и валидацию (V&V) промежуточных результатов ПО на процессах ЖЦ и оценивание степени достижения отдельных показателей;

– тестирование готовой системы, сбор данных об отказах, дефектах и др. ошибках и оценивание надежности по соответствующим моделям надежности с учетом результатов тестирования.

Модель качества ПО согласно стандарту задает шесть показателей (характеристик) q_1-q_6 (q-quality) качества:

q_1 - функциональность (functionality),
q_2 - надежность (realibility),
q_3 - удобство (usability),
q_4 - эффективность (efficiency),
q_5 - сопровождаемость (maitainnability),
q_6 - переносимость (portability).

Каждая характеристика q_i рассчитывается по специальным формулам и метрикам стандарта. Надежность оценивается согласно полученных на процессе тестирования ошибок, дефектов и отказов в ПО и по соответствующим моделям надежности (оценочным, измерительным и др.).

Данные по всем показателям качества q_1-q_6 оцениваются по формуле:

$$q_1 = \sum_{j=1}^{6} a_{1j} m_{1j} w_{1j}$$

где a_i - атрибуты каждого показателя качества (i=1-6); m_i – метрики каждого атрибута качества; w_i - вес каждого атрибута показателя качества системы. Полученные значения по показателям модели качества входят в сертификат качества продукта.

Заключение по подразделу проекта РФФИ 352-2018

В данном проекте представлена теории и методов моделирования изменяемых сложных систем (программных, информационных и операционных) из накопленных готовых системных ресурсов (ГОР) и создаваемых программных ресурсов (модулей, объектов, компонентов, сервисов и др. артефактов). Теория реализована в ОКМ и обеспечивает моделирование прикладных систем и средства для обеспечения вариабельности (изменяемости) систем по общепризнанной модели MF и модели системы (Msys) для создаваемых прикладных и Legacy-systems (таких как OS Linux, IBM, MS, Intel и др.). Рассмотрены процессы: верификации моделей, готовых компонентов типа reuses и ГОР; тестирования ГОР и их сборку (конфигурационную сборку) по моделям MF и Msys; обработки данных (в том числе и Big Data), используемых в системах с учетом стандарта ISO/IEC 11404 – General Data Types, 2007. Отработанные теории, методы и средства отработаны на примере создания технологии программирования веб-систем и варианта ядра OS Linux в среде Semantic Web Internet.

Определены модели характеристик масштабированных клиент-серверных архитектур при создании современных веб-систем. Приведено описание открытых моделей SOA и SCA для описания сервисов, серверов и клиентов. Дано описание разных методов сборки систем из ГОР (link, make, config и др.). Приведен вариант веб-системы для прикладных систем. И

описан метод извлечения знаний из Legacy-system/ Обоснованы перспективы развития технологии программирования прикладных систем с обеспечением безопасности, надежности и качества в клиент-серверной архитектуре. В работе приведены с аннотациями опубликованные статьи соучастников проекта, а также доклады на Всероссийских и зарубежных конференциях.

Таким образом, разработана оригинальная методология моделирования систем из готовых системных, функциональных и сервисных компонентов – ГОР (КПИ, reuses, objects, services, components и др.) Интернет. Эта методология реализована в виде технологии с процессами последовательного создания Веб системы в среде Интернет:

– определения модели вариабельности FM (Feature Model) для класса ОС и веб-систем;

– верификация проверки правиль¬ности моделей FM и систем ОС, Веб;

– тестирования и отработка отдель¬ных артефактов и ГОР со сбором данных об возникающих ошибках, дефектах и аварийных ситуациях для последующего внесения изменений;

– конфигурирование варианта системы из ГОР, их интерфейсов и моделей с помощью стандартного оператора config;

– интеграционное тестирования конфигурации системы;

– оценивание выходного файла на качество по стандарту ISO/IEC 9000 (1-4) quality в клиент-серверной архитектуре и определением надежности работы всей системы.

Данная методология использует механизмы взаимодействия разнородных программ и систем между собой и возможность их переноса в другую операционную среду; обеспечивает работы с данными, которые передаются через интер¬фейсы или выбираются из баз данных, онлайн-хранилищ данных в Cloud Computing, Azure, Big-data и др. Реализованы на основе системных и сервисных средств средства хранения отдельных ГОР систем, представленных в стандарте WSDL в репозиториях или е-бибиотеках, которые можно выбирать для последующего применения новых в веб-системах разного назначения.

Отдельные аспекты анализа направлений моделирования систем в современном мире представлены в книге Лаврищева Е.М. «Программная инженерия и технология программирования сложных систем» (www.biblio-online.ru, www.urait.ru).

Проведенное по данному проекту теоретическое и практическое моделирование программных и операционных систем завершилось созданием экспериментального вариантов ОС Linux и Веб-систем из ГОР с использованием объектно-компонентной парадигмы, современных моделей MF, Msys и процессов верификации, конфигурации и тестирования систем с оценкой качества выходного файла.

В сравнении с другими зарубежными технологиями данная методология имеет ряд преимуществ:

1. Наличие математического аппарата графового представления моделей объектов и представления связанных объектов матрицей смежности с их доказательством с помощью алгебраической теории достижимости (см. с.34-56 в книге «Программная инженерия и технология программирования сложных систем» - www.urait.ru; www.biblio- online.ru).

2. Формальное описание функциональных задач прикладных областей в ОКМ и аппарат преобразования к ЯП компонентов и возможностью верификации их правильности в современной системе model cheking;

3. Применение в управлении фабриками программ дисциплин (менеджмента, экономики, инженерии и др.), предложенных в статье – Classification of software engineering disciplines E. M. Lavrischeva 2008, Volume 44, Number 6, Pages 791-796.

4. Дисциплины вошли в Curricule Computer Science -14. - Theory and practice of software factories K. M. Lavrischeva, 2011, Volume 47, Number 6, Pages 961-972).

5. Применение в практике программирования операций сборки (assembler) и стандартной операции config (ISO/IEC 82896-2012, configuration) для сборки проверенных, оттестированных компонентов в выходной конфигурационный файл. - Ekaterina

M.Lavrischeva. Assemblling Paradigms of Programming in Software Engineering.- 2016, 9, 2016.- p.296-317, http://www.scrip.org/journal/jsea, http://dx.do.org/10.4236/jsea.96021.

Впервые на международной конференции “Science and Information-15, London прозвучала идея автоматизации ЖЦ (стандарта ISO/IEC 12207 Cycle Life 2007) - Lavrischeva Ekaterina. Ontological Approach to the Formal Specification of the Standard Life Cycle, Conference "Science and Information Conference-2015”, July 28-30, London, UK, www.conference.thesai.org.- p.965-972. Там на конференции IEEE предложил сделать патент. Патент сделан на русском языке в 2018 (см. файл Патент онтологии ЖЦ-2018).

С учетом возрастающего количества пользователей сети и большим объемов данных Big Data и методов вычислений Cloud Computing предлагается усовершенствовать клиент–серверную архитектуру в направлении применения множества отдельных серверов для хранения и обработки больших данных и организации вычислений на суперкомпьютерах. Предложена современная клиент–серверная архитектура, основанная на процессах: Front–end для клиентских приложений, систем и Back–end для обслуживания серверных частей приложений. Для этих процессов обеспечивается безопасность, защита и гарантия качественной работы в среде Интернет ви процессе создания как отдельных функциональных, системных и веб-сервисных компонентов, так и конфигурации Веб-системы и Веб-приложений. Разработчикам предоставляется технология создания прикладных систем для ряда областей биологии, медицины, генетики и др.

Отчеты по проекту РФФИ 352-2016 содержат статьи по следующим темам:

2016

1. Assembling Paradigms of Programming in Software Engineering
2. Семантические ресурсы для разработки онтологии научной и инженерной предметных областей
3. Теория объектно-компонентного моделирования программных систем (ОКМ)
4. Верификация и анализ вариабельных операционных систем
5. Моделирование семейств программных систем

2017

1. Развитие теории программ и систем в СССР. История и современные теории программ и систем
2. Научные основы программной инженерии
3. Аспекты моделирования вариабельных программных и операционных систем
4. Designing variability models for software, operating systems and their families
5. Фундаментальные основы программной инженерии
6. Подходы к представлению научных знаний в Интернет науке
7. Применение теории общих типов данных стандарта ISO/IEC 11404 GDT к Big Data
8. Static Verification of Linux Kernel Configurations.

Итоговый, 2018

1. Development of the Theory Programs and Systems in the USSR. History and Modern Theory
2. Анализ методов оценки надежности оборудования и систем. Практика применения методов
3. Подход к моделированию систем и сайтов из готовых ресурсов
4. Informatics. Formation of Computer Software and Technologies of Software Systems
5. Конфигурационная сборка варианта ядра Linux для прикладных систем

Литература к разделу 3

1. Лаврищева Е.М. Грищенко В.Н. Сборочное программирование.1991.

2. Липаев В.В. Позин Б.А., Штрик А.А.Технология сборочного программирования.1992.

3 Лаврищева Е.М. и Петренко А.К. Моделирование систем и их семейств. Труды ИСП РАН, 2016, том 28. вып. 6. - с. 180-190. DOI: 10.15514/ISPRAS-2016-28(6)-4.

4. Classification of software engineering disciplines. E. M. Lavrischeva 2008, Volume 44, Number 6, Pages 791-796, Software–Hardware Systems.

5. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование. Основы индустрии программных продуктов, 2009.

6. Theory and practice of software factories K. M. Lavrischeva, 2011, Volume 47, Number 6, Pages 961-972.

7. Лаврищева Е.М. Теория объектно-компонентного моделирования программных систем. Препринт ИСП РАН, № 29, 2016 г. - М.: 48 с. ISBN 078-5-91474-025-9.13

8. Газизуллина Р.Д. Современные подходы и методы моделирования изменяемых программных систем.-МФТИ, Факультет управления и прикладной математики, кафедра информатики и вычислительной математики, 2017.-46 с.

9. Лаврищева Е.М. Программная инженерия и технология разработки сложных систем, Юрт. biblio-0nline.ru, 2017.

10. Е.М. Лаврищева, В.С. Мутили, А.Г. Рыжов**. «**Аспекты моделирования вриабельных программных и операционных систем.- Сб. XIX Всероссийский научной конференции «Научный сервис в сети Интернет», Новороссийск, 18-23 сентября 2017.-327-341;

11. Е.М. Лаврищева, Л. Е. Карпов, А. Н. Томилин. Семантические ресурсы для разработки онтологии научной и инженерной предметных областей "Научный сервис в сети Интернет" 19-24 сентября 2016, Абрау-16.с.

12. Е. М. Лаврищева, Л. Е. Карпов, А. Н. Томилин. Подходы к представлению научных знаний в интернет науке. Сб. XIX Всероссийский научной конференции «Научный сервис в сети Интернет», Новороссийск, 18-23 сентября 2017.-с.310-326)..

13. Лаврищева Е.М. Рыжов А.Г. Применение теория общих типов данных стандарта ISO/IEC 12207 GDT применительно к Big Data.- The cconference “Actual problems in science and ways their development”, 27 decem. 2016, http://euroasia-science.ru. p.99-110.

14. E.M. Lavrischeva, V.S. Mutilin, A.G. Ryzhov. Designing variability models for software, operating systems and their families, 2017, Сборник ИСП РАН, 2017, issue 5, 2017, pp.93-109.- DOI: 14.15514/ISPRAS- 2017(5).

15. Kozin S.V., Mutilin V.S. Static Verification of Linux Kernel Configurations. Trudy ISP RAN/Proc.ISP RAS, vol. 29, issue 4, 2017, pp. 217-230. DOI: 10.15514/ISPRAS-2017-29(4)-14

16. Козин. С.В. Конфигурационная сборка варианта ядра Linux для прикладных систем.-Труды ИСП РАН, 2018,- Том 29. Вып.3.-с.

17. Кулямин В.В., Лаврищева Е.М., Мутилин В.С., Петренко А.К. Верификация и анализ вариабельных операционных систем. Труды Института системного программирования РАН, том 28, вып. 3, 2016, стр. 189-208. DOI: 10.15514/ISPRAS-2016-28(3)-12.

18. Лаврищева Е. М., Карпов Л. Е., Томилин А. Н. Семантические ресурсы для разработки онтологии научной и инженерной предметных областей// Научный сервис в сети Интернет: труды XVIII Всероссийской научной конференции (19-24 сентября 2016 г., г. Новороссийск). — М.: ИПМ им. М.В. Келдыша, 2016.- с.126-138.- doi:10.20948/abrau-2016-8.

19. Лаврищева Е.М., А.Г.Рыжов. Подход к моделированию систем и сайтов из готовых ресурсов. Научный сервис в сети Интернет: труды XX Всероссийской научной конференции (17-22 сентября 2018г. г. Новороссийск).-ИПМ им. М.В.Келдыша..-с. 321-345

20. E.M.Lavrischeva A.K.Petrenko. Informatics. Formation of computer software and technologies of software systems. Trudy ISP RAN/Proc. ISP RAS, 2018, vol. 5. Issue 2.

21. Газизуллина Р.К. Современные подходы и методы моделирования изменяемых программных систем, Магист. Работа МФТИ, 2017.- 46с.

22. Лаврищева Е.М. Фундаментальные основы программной инженерии.- Сборник научных трудов 5-международной конференции «Актуальные проблемы системной и программной инженерии», АПСПИ-2017, 14-16 октября 2017.-с.163-177.

Статьи по тематике данного проекта в иностранных журналах

1. Ekaterina M.Lavrischeva. Assemblling Paradigms of Programming in SoftwareEngineering.- 2016, 9.-2016.-p.296-317, http://www.scrip.org/journal/jsea, http://dx.do.org/10.4236/jsea.96021.

2. Ekaterina M. Lavrischeva. Ontology of Domains. Ontological Description Software Engineering Domain—The Standard Life Cycle, Journal of Software Engineering and Applications, 2016, 8, p.1-15. Published Online July 2015 in SciRes.http://www.scirp.org/journal/jseai

3. Lavrischeva Ekaterina. Ontological Approach to the Formal Specification of the Standard Life Cycle, Conference "Science and Information Conference-2015", July 28-30, London, UK, www.conference.thesai.org.- p.965-972.

4. E.M. Lavrischeva, V.S. Mutilin, A.G. Ryzhov. Designing variability models for software, operating systems and their families, 2017, Сборник ИСП РАН, 2017, issue 5, 2017, pp.93-109.- DOI: 10.15514/ISPRAS- 2017(5).

5. Lavrischeva E.M. Development of the theory programs and systems in the USSR. History and modern theory.- Sorucom-2017, IEEE Springer-2017. P.31-47.

6. Лаврищева Е.М. Фундаментальные основы программной инженерии.- Сборник научных трудов 5- международной конференции «Актуальные проблемы системной и программной инженерии». - Сборник трудов АПСПИ-2017, 14-16 октября 2017.-с.163-177.

7. Lavrischeva E.M., Petrov I.B. Ways of Development of Computer Technologies to Perspective Nano, Agenda - Future Technologies Conference (FTC) 2017, 29-30.-November 2017| Vancouver, Canada, p. 539-549.

8. E.M.Lavrischeva. Science of computer programs and systems in XX-XXI centuries: Past, Present, Future.- European Journal Mathematics and Computer Science.- Vol.5 N 1.2018.-ISSN 2059-9951.- p.67-87.

9. E.M. Lavrischeva. Scientific Basis of System Programming.- Journal of Software Engineering and Applications (JSEA), Vol. 11 No. 8 of August issue, 2018.-N 11.-p.408-434, ISSN online 1945-3124, ISSN Print 1945-3116.

10. E. M. Lavrischeva.The Scientific basis of software engineering.- Inteional Journal of Applied and Natural Sciences (IJANS) ISSN(P): 2319-4014; ISSN(E): 2319-4022 Vol. 7, Issue 5, Aug – Sep.- 2018; p. 15-32.

11. Lavrischeva E.M. Science of the computer programs and systems in XX-XXI centuries: past, present, future. European Journal of Mathematics and Computer Science Vol. 5 No. 1, 2018. ISSN 2059-9951, Progressive Academic Publishing, UK.- Page 67-92, www.idpublications.org.

Доклады соавторов и руководителя проекта по тематике проекта РФФИ:

1. Е.М. Лаврищева, Л. Е. Карпов, А. Н. Томилин. Семантические ресурсы для разработки онтологии научной и инженерной предметных областей, Доклад на конференцib "Научный сервис в сети Интернет" 19-24 сентября 2016, Абрау-16.

2. Лаврищева Е.М. Теория объектно-компонентного моделирования вариабельных программных систем. Доклад на Международной научно-практической конференции «Теория активных систем (ТАС-2016), 16-17 ноября 2016.- ИПУ РАН им. В.А.Трапезникова.- Презентация.

3. Лаврищева Е,М, Научные основы построения программных и информационных систем.-Х11 Международная научно-практическая конференция «Современные информационные технологии и ИТ-образовании», 25-26 ноября 2016.- Мин.образования России. МГУ им. Ломоносова, факультет вычислительной математики и кибернетики.- Презентация и публикация доклада в Сборнике. с.50-56.

4. Лаврищева Е.М. Петренко А.К. Моделирование систем и их семейств. Научная конференция Открытые системы ИСПРАН, 24-27 ноября 2016. Презентация и публикация -

Труды ИСП РАН, 2016, том 28, вып. 6, 2016, стр. 49-64. ISSN 2220-6426 (Online), ISSN 2079-8156 (Print). DOI: 10.15514/ISPRAS-2016-28(6)-4.

5. Лаврищева Е.М. Научные основы программ и технологии программирования систем.- Доклад 15.03.2017. Российская Академия наук, Центральный дом учёных. Презентация 103слайда on www.ispras.ru/Lavrischeva

6. Е. М. Лаврищева, Л. Е. Карпов, А. Н. Томилин. Подходы к представлению научных знаний в Интернет науке. Сб. XIX Всероссийский научной конференции «Научный сервис в сети Интернет», Новороссийск, 18-23 сентября 2017.-с.310-326. Презентация доклада.

7. Е.М. Лаврищева, В.С. Мутилин, А.Г. Рыжов**.** Аспекты моделирования вариабельных программных и операционных систем.- Сб. XIX Всероссийский научной конференции «Научный сервис в сети Интернет», Новороссийск, 18-23 сентября 2017.-327-341.

8. Лаврищева Е.М. Научные основы программной инженерии (Понятия, Парадигмы, Технологии, CASE-средства).- Theoretical and Applied Aspects of Program Systems Development» (TAAPSD'2017), 4-8дек. 2017. P.201-214. Презентация доклада.

9. Лаврищева Е.М. Развитие теории программ и систем в СССР. История и современные теории.- Сборник SORUCOM-17, 2017. Развитие ВТ в России и в странах бывшего СССР. История и перспективы.- 3-5 октября 2017.-с.162-176. Презентация

10. Лаврищева Е.М., Пакулин Н.В., Рыжов А.Г., Зеленов С.В. Анализ методов оценки надежности оборудования и систем. Практика применения методов. 5 Научно-практическая конференция - OS DAY, Москва, 17-18мая 2018. Презентация доклад и публикация на анг. языке.-Труды ИСП РАН, том5 DOI:10.15514/ISPRAS-2018-30(3), 2018.- p.(http://0x1.tv/20180517F)

11. Лаврищева Е.М. Фундаментальные основы программной инженерии.- 5-международная конференция «Актуальные проблемы системной и программной инженерии». - АПСПИ-2017, 14-16 октября 2017.- Презентация и публикация в сборнике научных трудов АПСПИ-2017 с.163-177.

12. Лаврищева Е.М. Арутюн А.И. Информатика и ЭВМ-70. Анализ и аспекты развития.- Открытой конференции ИСП РАН им. В.П.Иванникова.-ISPRAS-2018, 22-23.11.2018. Презентация доклад и публикация статьи «Informatics -70. computerization aspects of programming software and informatic systems technologies.- ISP RAN/Proc. ISP RAS, 2018, vol. 30, issue 6, pp. 3-44. (http://0x1.tv/20181122AF)

13.Лаврищева Е.М., А.Г.Рыжов. Подход к моделированию систем и сайтов из готовых ресурсов.- .XX Всероссийская конференция , 17-22 сентября 2018г. г. Новороссийск.-ИПМ им. М.В.Келдыша.- Презентация доклада. Публикация в сборнике.-с. 321-345.

14. Лаврищева Е.М. Современные системы искусственного интеллекта.-Симпозиум, Искусственный интеллект: различные подходы к его воплощению (компьютерно0-сетевые, нейро-сетевые, квантовые технологии и другие.- Москва, 13 октября 2018.-Федеральное государственное образовательное учреждение при Правительстве Российской Федерации. Колледж информатики и программирования. Презентация доклада

Сопоставление полученных результатов с мировым уровнем

Дано научное обоснование теоретического пути развития технологии программирования, начиная с появления первых ЭВМ и вплоть до современных парадигм программирования, объектно-компонентного метода (ОКМ), обеспечивающего логико-математическое моделирование различных видов программных и операционных систем из готовых ресурсов (объектов, компонентов и сервисов) в виде графа объектов, в том числе и Веб-приложений. В отличие от принятого для производства программных продуктов ProductLine и диаграммного UML3.2, метод ОКМ является более простым способом и ориентирован на построение графовой объектной модели (ОМ) и модели характеристик MF. Разработан механизм преобразования объектной графовой модели к компонентной модели (СМ), элементами которой становятся трансформированные к программным компонентам методы и функции ОМ. Показывается, что ОМ и СМ эквивалентны. Предусмотрены

операции модификации этих моделей и формирования с помощью математических операций вариантов выходных конфигурационных файлов.

Отметим, что компонентное проектирование получила внедрение в проекте Информатизация НАНУ (2007-2010).

Вариант метода на компонентной основе защищен в докторской диссертации Грищенко В.Н (Теоретические и прикладные основы компонентного программирования, 2007). Затем отработанный метод стал частью ОКМ. Теория ОКМ отображена в ряде статей за рубежом (Lavrischeva E.M., Kolesnik A.L., Stenyashin A.U. Object-component Development of Application and systems. Theory and Practice» в журнале USA "Software Engineering and Application" (2014). Метод ОКМ получила признание в Украине, в США и в Лондонском журнале "London press" 4 .11.2016. Главный автор статьи Лаврищева Е.М. сделали членом «Quarterly Franklin Membership» (ID #5134421UK) с правом бесплатного публикования статей в журнале «London Journal of Research in Computer Science and Technology» (LJCST). Отработана теория ОКМ и представлена в препринте «Теория объектно-компонентного метода моделирования программных систем».- ИСП РАН, 2016.- 48 с.

В 2018 году на данном проекте проведено усовершенствование ОКМ, суть которого состоит в адаптации моделей систем и MF в классе ОС и Веб-систем. Дана характеристика уточненных моделей и подхода к организации процесса генерации новых вариантов ОС и сборки Веб-систем. Приводится описание операций верификации, тестирования и конфигурирования готовых ресурсов ОС для получения рабочего конфигурационного файла. Описаны основные положения проектирования операционных и сервисных из готовых сервисных и операционных ресурсов.

Результаты проектирования задач проекта в 2017 опубликованы в статьях.

Методы и подходы, использованные в ходе выполнения Проекта и оценка степени оригинальности

Основу моделирования систем и их семейств составляют модели систем и вариабельности MF задаваемые новым оригинальным методом ОКМ.

Ранее созданный ООП и UML обеспечивают диаграммное проектирование ПС - громоздкое и трудоемкое, постоянно усовершенствованного.

Начиная с 2009 года, международные комитеты ACM и IEEE предложили развивать теорию и методы в рамках нового направления SEMAT (Software Engineering Theory and Methods). С этого времени для проектирования систем предлагается математическая теорию множеств, логика и алгебраического проектированию разного рода компьютерных систем.

Метод ОКМ соответствует требованиям SEMAT. Он основан на логико-алгебро-математическом аппарате, теории доказательства и верификации программных ресурсов и систем. Метод обеспечивает построение модели MF и графовой ОМ, в вершинах которой находятся отдельные объекты системы, а на дугах отношения (связи) между ними MF. По этим моделям дается спецификация отдельных объектов, операций их обработки и конфигурирования к выходному коду.

В результате многолетнего применения отдельных аспектов метода ОКМ сформирован **новый подход** к графовому моделированию систем из готовых программных ресурсов – Reuses. Эти объекты задают функции и методы предметной области и преобразуются к программным компонентам согласно компонентной теории ОКМ. Ресурсы собираются в конфигурационный файл, который тестируется и проверяется на правильность выполнения функций.

Математическая теория ОКМ получила признание за рубежом. Ряд европейских организаций стран - Германия, Англия, Китай, Индия, Бразилия предлагают совместную работу для практического использования создания систем, а также прочитать курс лекций.

Данный метод докладывался и обсуждался в Доме ученых РАН и на Международных конференциях:

Corucom-2017 «Развитие ВТ в России и в странах бывшего СССР. История и перспективы»,

АПСПИ-2017 - 5-межд. конференция «Актуальные проблемы системной и программной инженерии»,

Абрау-2017 - XIX Всероссийский научной конференции «Научный сервис в сети Интернет.

Научные и инженерные подходы к разработке массовых систем представлены в монографии Лаврищевой Е.М. «Программная инженерия и технология программирования сложных систем..-Учебник, Москва «Юрайт», 2017, biblio-online.ru.

Отдельные аспекты метода ОКМ опубликованы в статье Springer.

ОКМ признан как метод математического проектирования сложных систем с доказательством правильности реализованных функций системы.

В этом состоит новизна и оригинальность данного проекта.

Научные работы по проекту РФФИ 352 в 2016-2018

1. **Lavrischeva Ekaterina.** Ontological Approach to the Formal Specification of the Standard Life Cycle, Conference "Science and Information Conference-2015", July 28-30, London, UK, www.conference.thesai.org.- p.965-972.

2. **Лаврищева Е.М.** Научные основы программ и технологии программирования систем.- Доклад 15.03.2017. Российская Академия наук, Центральный дом учёных. Презентация
http://www.ispras.ru/publications/2017/the_scientific_foundations_of_programs_and_technology_programming_systems/ 2017-2022 - rio@mipt.-49c.

3. **Лаврищева Е.М.** Развитие теории программ и систем в СССР. История и современные теории программ и систем.- Сборник SORUCOM-2017.- Развитие ВТ в России и в странах бывшего СССР. История и перспективы.- 3-5 октября 2017.-с.162-176.

Аннотация. Приведен анализ теории программ советских ученых (Ляпунова, Ершова, Янова, Глушкова, Ющенко, Самарского, Липаева и др.). Определена сущность теории программ, технологии программирования, синтеза, сборки и доказательства программ и систем (1963-1990). Представлены элементы теории инженерии программных продуктов - Software Engineering (1980-2016) и новые перспективные теории и методы моделирования изменяемых систем из готовых программных ресурсов (объектов, компонентов, сервисов и др.), Определена конфигурационная сборка ресурсов по моделям в варианты выходного кода продуктов и систем и процесс верификации моделей систем и модели характеристик MF, а также теория сборки готовых ресурсов в систему по верифицированным моделям и аппарата алгебраических систем с механизмами трансформации передаваемых данных между объединяемыми ресурсами через интерфейс. Предложен подход к тестированию отдельных ресурсов систем и создания конфигурационного выходного файла.

Ключевые слова: Теория, методы, научные основы, программные системы, построение, верификация, доказательство, моделирование, интероперабельность.

4. **Лаврищева Е.М.** Научные основы программной инженерии (Понятия, Парадигмы, Технологии, CASE-средства.- Theoretical and Applied Aspects of Program Systems Development» (TAAPSD'2017), 4-8 дек. 2017. - с. 201-214.

Аннотация. Определяются научные понятия и фундаментальные основы Software Engineering (SE). Базовые понятия - это объекты, модули, программы, системы; процессы разработки объектов. Фундаментальную основу SE составляют: метод сборки модулей; дисциплины SE (научная, инженерная, экономическая, управленческая и др.); парадигмы программирования модулей, объектов, компонентов и др.; жизненный цикл (стандарт ISO/IEC Life Cycle 12207); линии изготовления систем, фабрики программ и AppFabric; новая логико-математическая теория объектно-компонентного моделирования (ОКМ) систем; верификация и тестирование изготовленных систем. Инструменты поддержки научных и фундаментальных основ на сайте http://7dragons.ru/ru.

Ключевые слова: наука, концепция, формализмы модулей, метод сборки, теория, ОКМ, ОСМ, Product Line, Life Cycle, model FM, configuration, verification/ realization.

5. **Е.М. Лаврищева, В.С. Мутилин, А.Г. Рыжов.** Аспекты моделирования вариабельных программных и операционных систем.- Сб. XIX Всероссийский научной конференции «Научный сервис в сети Интернет», Новороссийск, 18-23 сентября 2017.-327-341.

Аннотация. Рассматриваются подходы к заданию вариабельности программных и операционных систем (ПС), и их семейств (СПС). Предложена модель характеристик (Feature Model), которая включает базовые элементы для ПС и структура ядра OS Linux, на основе которой формируется вариант модели ОС из системных и функциональных элементов ядра ОС. Предложена модель конфигурационной сборки элементов ПС и OS Linux. Определена концепция задания модели вариабельности для OS Linux, основанные на графовой структуре системы из готовых ресурсов (ГОР) – функциональных и интерфейсных элементов для ПС, а также системных и функциональных элементов ядра OS Linux, с помощью которых генерируются варианты ОС. Определен подход к управлению вариабельностью и конфигурационной сборкой ОС с помощью объектно-компонентного метода (ОКМ) для получения вариантов выходного продукта. Предложен подход к тестированию вариантов ОС.

Ключевые слова: программная система, операционная система, семейство программных систем, вариабельность, функциональный и интерфейсный элемент, модель характеристик, управление конфигурацией, верификация, тестирование.

6. **E.M. Lavrischeva, V.S. Mutilin, A.G. Ryzhov.** Designing variability models for software, operating systems and their families, 2017, Сборник ИСП РАН, 2017, issue 5, 2017, pp. 93-110. DOI: 10.15514/ISPRAS-2017(5)-6.

Abstract. The complexity of existing Legacy systems and the difficulty of amending it led to the development of the new concept of variability of systems specified by a model of the characteristics of FM (Feature Model). In the paper, we discuss the approaches to formal definition of FM and creating on its basis variants of program systems (PS), operating systems (OS) and families of program systems (FPS) for PS and OS. We give methods of manufacturing of PS in the Product Family/Product Lines, the conveyor of K.Czarnecki for assembling of artifacts in the space of problems and solutions, logical-mathematical modeling of PS from the functional and interface objects by Object-Components Method (OCM), extraction of the functional elements from OS kernel to FM for the generation of new variants of the OS. We discuss approaches for formalization of variability of legacy and new PS and their FPS. The new concept of management of variability systems with help OCM is defined. The approach to verify models of the FM, PS, FPS and OS and to configuration of functional and interface objects for obtaining the variants of the resulting product are proposed. We elaborate the characteristics for the testing process of variants of the PS, OS and FPS.

Keywords: variability model; software systems; family of systems; configuration; variant; functional; interface element; requirement; management.

7. **Лаврищева Е.М.** Фундаментальные основы программной инженерии.- Сборник научных трудов 5-международной конференции «Актуальные проблемы системной и программной инженерии». - Сборник трудов АПСПИ-2017, 14-16 октября 2017.-с.163-177.

Аннотация. Определены базовые понятия и фундаментальные основы Software Engineering (SE). Базовые понятия – это объекты, модули, программы, системы; процессы разработки объектов. Фундаментальную основу SE составляют: метод сборки (конфигурирования) модулей; дисциплины SE (научная, инженерная, экономическая, управленческая и др.); парадигмы программирования модулей, объектов, компонентов и др.; жизненный цикл (стандарт ISO/IEC Life Cycle 12207); линии изготовления систем; фабрики программ и AppFab; логико-математическая теория объектно-компонентного моделирования (ОКМ) изменяемых систем; технология тестирования систем и оценивания качественных показателей.

8. **Е. М. Лаврищева, Л. Е. Карпов, А. Н. Томилин.** Подходы к представлению научных знаний в Интернет науке. Сб. XIX Всероссийский научной конференции «Научный сервис в сети Интернет», Новороссийск, 18-23 сентября 2017.-с.310-326.

Аннотация. Рассматриваются подходы к формированию вариабельности действующих и создаваемых программных систем (ПС), и их семейств (СПС). Предложенная модель характеристик (Feature Model) K. Pohl в проекте SEI Product Line и Product Family, а также модель конвейерной сборки готовых артефактов в пространстве задач и решений K. Czarnecki послужили исходной концепцией для создания модели вариабельности для программных и операционных систем (ОС). Определены новые модели ПС и ОС, включающие готовые ресурсы (ГОР) – функциональные и интерфейсные элементы, а также модель характеристик (МХ) элементов, по которым генерируются варианты ПС из их семейств. Определена концепция управления вариабельностью и конфигурационной сборкой ПС из ГОР в объектно-компонентном методе (ОКМ) для получения вариантов выходного продукта. Предложен подход к тестированию вариантов ПС.

9. **Лаврищева Е.М., Рыжов А.Г.** Применение теории общих типов данных стандарта ISO/IEC 12207 GDT к Big Data", «Актуальные проблемы в современной науке и пути их решения», 27 декабря 2016, http://euroasia-science.ru

Аннотация. Рассматриваются вопросы применения теории общих типов данных (GDT) к Большим Данным (Big Data). Подан базовый аппарат представления типов данных (ТД) GDT (стандарта ISO/IEC 11404-2007) и теория генерации нестандартных ТД GDT. Определены операции и набор функций трансформации данных с одной платформы на другую. Предложен подход к анализу неструктурированных данных и генерации с помощью функций, аналогичных функциям библиотеки CTS (Common Type System) VS.Net.

Ключевые слова: общие типы данных; простые, сложные, генерируемые, неструктурные данные; трансформация; генерация; большие данные; библиотеки функций.

10. **Lavrischeva E.M.** Development of the theory programs and systems in the USSR. History and modern theory.- Sorucom-2017, IEEE Springer-2017.

Abstract. The theory of the programs and systems of Soviet scientists (A.A.Lyapunov, A.P.Ershov, Yu.I.Janov, D.M. Glushkov, E.L.Yushchenko, V.V. Lipaev, etc.) is given. The essence of the theory of programs, programming techniques, synthesis, Assembly, and composition of program systems (1963-1990) is defined. Basic concepts of the theory of engineering software (1980-2016) and SEMAT (2009) are presents. A new promising theory and methods for modeling changing systems from finished software resources (objects, components, services, etc.) and their configuration in the building output code products and systems are defined.

Keywords — theory, methods, science, reuses, object, Programming technology, software engineering, product, modeling, verification, testing, configuration.

11. **Е.М.Лаврищева, А.Г.Рыжов.** Подход к моделированию систем и сайтов из готовых ресурсов. Научный сервис в сети Интернет: труды XX Всероссийской научной конференции (17-22 сентября 2018 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2018. — С. 321-345. — ISSN URL: http://keldysh.ru/abrau/2018/theses/50.pdf. ISBN книги doi:10.20948/abrau-2018-50 серийного издания

Аннотация. Рассматривается компонентный подход к созданию систем и сайтов из готовых ресурсов (компонентов, объектов, сервисов и reuses). В основе подхода лежит графовая и компонентная модель (КМ), включающая функциональные, системные, сервисные и интерфейсные программные ресурсы и алгебру их обработки. Функции объектов-компонентов описываются в языках программирования (ЯП), а их интерфейсы в языках API, IDL, WSDL и др. Они реализуются в клиент–серверной архитектуре и взаимодействуют между собой через интерфейс. Дан анализ моделей систем, веб-сайтов и приложений, выполняемых в двухуровневой клиент-серверной архитектуре и представлен современный путь развития этой архитектуры в связи с увеличением числа пользователей, работающих с современными огромными объемами данных Big Data и Cloud Computing. Готовые ресурсы проверяются на правильность их спецификации в ЯП, тестируются и

собираются сведения об ошибках для их исправления и оценки надежности. Готовые ресурсы конфигурируются в систему и проверяются на функциональность и вариабельность. Переформатирован сайт ИТК в виду http://7dragons.ru в ГДР. Он, включает готовые ресурсы, артефакты и процессы сборки программных систем с использованием сервисно–компонентных ресурсов Интернет SOA, SCA, SOAP и др. при создании Веб–систем и сайтов.

Доклады на Всероссийских конференциях

1. Лаврищева Е.М. Научные основы программ и технологии программирования систем.- Доклад 15.03.2017. Российская Академия наук, Центральный дом учёных. 103 слайда - www.ispras.ru/Lavrischeva

2. Лаврищева Е.М. Фундаментальные основы программной инженерии. АПСПИ-2017, 14-16 октября 2017 -5-международной конференции «Актуальные проблемы системной и программной инженерии».

3. Е.М. Лаврищева (Л. Е. Карпов, А. Н. Томилин). Подходы к представлению научных знаний в Интернет науке. XIX Всероссийский научной конференции «Научный сервис в сети Интернет», Новороссийск, 18-23 сентября 2017.

4. Лаврищева Е.М. Научные основы программной инженерии (Понятия, Парадигмы, Технологии, CASE-средства). Theoretical and Applied Aspects of Program Systems Development» (TAAPSD'2017), 4-8 дек. 2017.

5. Лаврищева Е.М. Развитие научных основ программ и систем в СССР. История и современные теории.- Доклад на SORUCOM-2017.- Развитие ВТ в России и в странах бывшего СССР. История и перспективы.- 3-5 октября 2017.

6. Е.М. Лаврищева (В.С. Мутили, А.Г. Рыжов). Аспекты моделирования вариабельных программных и операционных систем.- Сб. XIX Всероссийский научной конференции «Научный сервис в сети Интернет», Новороссийск, 18-23 сентября 2017.

Метод ОКМ встоен в в SEMAT. ОКМ описан логико-алгебро-математическом аппарате, обеспечивает доказательства и верификацию программных ресурсов и систем из компонентов. Он исновывается на моделях MF и графовой ОМ, в вершинах которой находятся отдельные объекты системы, а на дугах отношения (связи) между ними MF. По этим моделям проводится спецификация отдельных объектов, и их сборка для получения выходного кода конфигурационной структуры. В результате многолетнего применения отдельных аспектов метода ОКМ сформирован новый подход к графовому моделированию систем из готовых программных компонентов – Reuses. Ресурсы собираются в конфигурационный файл, который тестируется и проверяется на правильность выполнения функций отдельных ресурсов компонентов, преобразовании типов данных собираемых компоненгтов.

Математическая теория ОКМ получила признание за рубежом. Ряд европейских организаций Германии, Англии, а также Китая, Индии, предлагают совместную работу для практического использования, а также чтения лекций студентам.

Данный метод обсуждался в Доме ученых РАН и на Международных конференциях:

– Corucom-17 «Развитие ВТ в России и в странах бывшего СССР. История и перспективы»,

– АПСПИ-2017 - 5-межд. конференция «Актуальные проблемы системной и программной инженерии»,

– Абрау-17 - XIX Всероссийский научной конференции «Научный сервис в сети Интернет

– Научные и инженерные подходы к разработке массовых систем представлены в монографии Лаврищевой Е.М. «Программная инженерия и технология программирования сложных систем..-Учебник, Москва*Юрайт*2017, biblio-online.ru. -431с.

Отдельные аспекты метода ОКМ были опубликованы. ОКМ признан как метод математического проектирования сложных систем с доказательством правильности реализованных функций программных и технических систем.

Краткое назначение конечной продукции, технологии или услуг, которые будут производиться с применением новых методов моделироваания и программирования в рамеках проекта РФФИ 352 : 2016-2018).

По данному проекту РФФИ отработана новая теория и метод (парадигма) объектно-компонентного (ОКМ) моделирования сложных программных, информационных и операционных систем и Веб систем с использованием готовых ресурсов (КПИ, объектов, компонентов, сервисов, аспектов и др.), которые накопливаются в Библиотеках Интернет и интегрируются методом сборки (link, make, config), входящих в стандартной конфигурационной сборки (ISO/IEC 828: 1999-2012) в среде Интернет. В сравнении с коммерческой парадигмой Agile ОКМ имеет преимущества:

1) Математический аппарат графового представления моделей объектов и представления связанных объектов матрицей смежности с их доказательством в алгебраической теории достижимости;

2) Формальное описания задач прикладных областей ресурсами в современных ЯП с верификацией правильности описания в современной системе model cheking;

3) Применения установившихся операций сборки (link, make, config, building, waver) конфигурационной сонфигурационной структуры по стандарту ISO/IEC 828:1996-2012 (configuration), проверенных и оттестированных компонентов в выходной структуры конфигурационного файла прикладной системы знаний.

Статья по ОКМ - Lavrischeva E.M. “Object-Component Development of Application and Systems. Theory and Practice» сделана для общества Quarterly Franklin Membership (Membership D#513442|UK) в 2016 году).

Доклады опубликованы в соответствующих конференциях и в журналах Scopus и Web of Sciences.

Адреса (полностью) ресурсов в Интернете, подготовленные по Проекту РФФИ

1) http://0x1.tv/20181122AF Conf. ISPRAS. OPEN -18,

http://0x1.tv/20180517F Conf OS DAY –Надежность-18.

www.biblio-online.ru/ www.urait.ru

Книга Лаврищева Е.М. Программная инженерия и технология программирования сложных систем (См. файл заключение на проект РФФИ).

– Свидетельство о государственной регистрации программ для ЭВМ № 20018615442.

«Реализация метода онтологического моделирования домена ЖЦ стандарта ICO/IEC 12207» от 20.03.2018 (файл презентации Онтология ЖЦ в Лондоне в 2015г.).

E.M.Lavrischeva A.K.Petrenko. Informatics. Formation of computer software and technologies of software systems. Trudy ISP RAN/Proc. ISP RAS, 2018, vol. 5. Issue 2.

4) Наличие математического аппарата графового представления моделей объектов и представления связанных объектов матрицей смежности с их доказательством с помощью алгебраической теории достижимости (см. с.34-56 в книге «Программная инженерия и технология программирования сложных систем» - www.urait.ru; www.biblio- online.ru)

5) Формальное описание функциональных задач прикладных областей в ОКМ и аппарат преобразования к ЯП компонентов и возможностью верификации их правильности в современной системе model cheking;

6) Применение в управлении фабриками программ дисциплин (менеджмента, экономики, инженерии и др.), предложенных в статье – Classification of software engineering disciplines E. M. Lavrischeva 2008, Volume 44, Number 6, Pages 791-796.

Дисциплины вошли в Curricule Computer Science -14.

– Theory and practice of software factories K. M. Lavrischeva, 2011, Volume 47, Number 6, Pages 961-972).

7) Применение в практике программирования операций сборки (assembler) и стандартной операции config (ISO/IEC 82896-2012, configuration) для сборки проверенных, оттестированных компонентов в выходной конфигурационный файл.

– Ekaterina M.Lavrischeva. Assemblling Paradigms of Programming in Software Engineering.- 2016, 9, 2016.- p.296-317, http://www.scrip.org/journal/jsea, http://dx.do.org/10.4236/jsea.96021.

8) Впервые на международной конференции “Science and Information-15, London прозвучала идея автоматизации ЖЦ (стандарта ISO/IEC 12207 Cycle Life 2007).

– Lavrischeva Ekaterina. Ontological Approach to the Formal Specification of the Standard Life Cycle, Conference "Science and Information Conference-2015”, July 28-30, London, UK, www.conference.thesai.org.- p.965-972. Там на конференции IEEE предложил сделать патент. Патент сделан на русском языке в 2018 (см. файл Патент онтологии ЖЦ-2018).

9) В книге Лаврищева Е.М. Программная инженерия и технология программирования сложных систем, Юрайт www. biblio-0nline.ru, www.urait.ru 2018 описан метод сборки ресурсов с ипользованием интерфейса объединяемых компонентов.

10). Лаврищева Е.М., Петренко А.К. Технология сборки интеллектуальных и информационных ресурсов Интернет // Научный сервис в сети Интернет: труды XXI Всероссийской научной конференции (23-28 сентября 2019 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2019. — С. 469-488. — URL: http://keldysh.ru/abrau/2019/theses/93.pdf doi:10.20948/abrau-2019-93

11). Лаврищева Е.М., Зеленов С.В., Рыжов А.Г.. Теория моделирование программно-технических систем с обеспечением безопасности и надежности в среде Web-services Интернет. Труды ИСП РАН, Том 30. № 6.- с.156-187.

12). Е.М.Лаврищева, И.Б.Петров. Моделирование технических и математических задач прикладных областей знаний на ЭВМ. Труды ИСП РАН 2020, Том 32 № 6 с.167-182.

13). Модельный подход к обеспечению безопасности и надежности Web-сервисов. Е.М. Лаврищева, С.В. Зеленов. Труды ИСП РАН 2020, Том 32 № 5. С.151-163..

14). Лаврищева Е.М. Теория графового моделирования сложных систем из модульных элементов для прикладных областей. Austria-science»1 часть №28/2019.- p.12-30. http://austria-science.info

15). Е.М.Лаврищева, И.Б.Петров. Теория моделирования технических и математических задач предметных областей знаний. Евроазиатское Научное Объединение. 2021.№ 1- (1).-с.35-43.

Раздел 4. Форма 504. Проект РФФИ 19-01-00206 (2019-2021). Модели, методы и средства обеспечения надежности и качества технических и программных средств

Лаврищева Е.М., Зеленов С.В., Морозов С.В., Мутилин В.С., Козин. С.В., Рыжов А.Г.

Основными задачами данного проекта в 2019г. были: исследование моделей и методов надежности и качества применительно к техническим и программным системам (ТПС); отработка графового метода моделирования систем из модулей и развитие парадигм программирования; усовершенствование отечественного метода сборки модулей, КПИ, reuses ТПС и других областей знаний; проведение верификации и тестирования модулей, КПИ и ТПС для обнаружения ошибок (дефектов и отказов) и их исправления; оценивание по соответствующим моделям надежности и качества модулей, КПИ; проведение конфигурационной сборки варианта ядра OS Linux с оценкой качества. По основным задачам проекта сделаны научные доклады и опубликованы 7 статьей в трудах научных конференций России: Научный сервисов Интернет-2019; APSSE-19 и CEUR-2019; ISPRAS OPEN-2019; Austria-2019: статьи по теории графового моделирования модулей, КПИ, reuses ТПС и в журнале «Computer and Information Sciences» – vol.12. №4 2019 and TMLAI Vol 7 No 6 Dec. 2019 "The Theory Graph Modeling and Programming Paradigms of Systems from Modules to the Application Areas". Описание перспективной парадигмы объектного и компонентного моделирования (ОКМ) и сборки объектов и компонентов, получившей признание за рубежом (пришло более 10 заявок по электронной почте от специалистов Европы, США, Китая, Тайваня и др.).

Models, methods and means of ensuring the reliability and quality of technical and software tools

The main objectives of this project in 2019 were: research of models and methods of reliability and quality in relation to technical and software systems (TPS); development of graph method of modeling of systems of their modules and programming paradigm; improvement of the domestic method of Assembly of modules, CPU, reuses, TPS and other areas of knowledge; verification and testing of modules, CPU and TPS to detect errors (defects and failures) and their correction; evaluation on the corresponding models of reliability and quality of modules, CPU; Scientific reports were made on the main tasks of the project and 5 articles were published in the proceedings of scientific conferences in Russia, CEUR, Austria, APSSI and article on the theory of graph modeling of TPS in journal «Computer and Information Sciences» – vol.12. №4 2019 (ISSN: 1913-8989, E-ISSN: 1913-8997) - "The Theory Graph Modeling and Programming Paradigms of Systems from Modules to the Application Areas" and TMLAI Vol. 7 No 6 Dec 2019 (DOI: https://doi.org/10.14738/tmlai.74.6782). The paradigm of object-component modeling (OCM) and Assembly of objects and components are described in foreign journals it are receive more than 10 applications by email from experts in Europe, the United States, China, Taiwan, etc.

Введение

Понятие моделирования технических объектов возникло с давних времен. Модель (парадигма) включает основные объекты, закономерностей их развития и связи между ними.

Модель определяется на множестве величин, которые надо определить. И как говорил Аль Хорезми, надо разработать численный алгоритм реализации разных величин модели. Алгоритм решения задачи декомпозируется на отдельные подзадачи и задание связей между ними. Для проведения вычислительного эксперимента определяются: математическая модель - алгоритм – программа (А.А.Самарский). Эта триада обозначает теорию эксперимента, численное его моделирование и описание алгоритма вычисления задачи по программе на ЭВМ. К модели информационных объектов относятся формулы, набор правил для их вычисления и соглашения о связях между ними. Для каждой научной дисциплины (математика, информатика, физика, биология, медицина и др.) должна создаваться концептуальная модель. Одним из способов ее создания является онтология, термин которой возник в 1613г. и означает концептуализацию знаний о некоторой предметной области в виде модели, включающей множество объектов (из тезауруса предметной области) и связи между ними. Т.е. концептуальная модель имеет вид:

КМ = <X, R, F>,где X – конечное множество понятий предметной области;

R – конечное множество отношений между понятиями;

F – конечное множество функций (алгоритмов) их интерпретации.

КМ модель задает информацию о понятиях и фиксирует их как знания о соответствующей научной предметной области знаний или дисциплине.

С момента появления вычислительной техники и компьютеров понятие модели и моделирования широко вошли во все сферы технического и математического программного обеспечения (МПО). Возникла научная системная дисциплина, которая включает:

– теоретические методы вычислительной математики (линейная алгебра, аналитическая геометрия, дифференциальное, интегральное исчисление и т.п.);

– функции диспетчирования, мониторинга системными компонентами ПО (трансляторами, отладчиками, Базами Данных и т.п.);

– системные функции управления операционной системой (ОС) и компонентами ПО (компиляторами, отладчиками, верификаторами и др.) и поддержки создания математических предметных областей, прикладных систем и сетей (специальность 05.13.11);

– компиляторы (трансляторы, интерпретаторы) ПО проводят анализ правильности описания алгоритмов программ в ЯП (Prolog, Smalltalk, C++, Python и др.) и трансформацию программ к промежуточному виду или к коду компьютера для последующего выполнения;

– средства взаимодействия, коммуникации сервисных, сервисно-компонентных и системных ресурсов Интернет для создания предметных областей в среде Интернет.

Данная системная дисциплина основывается на теории программирования граф-схем ТПС *(*компьютеров, аппаратуры, приборов) а также изготовление трансляторов и компонентов ОС и систем. Формальными средствами системного программирования являются: теория алгоритмов, теория анализа систем и логико-математический графовый аппарат представления отдельных компонентов ПО и систем компьютеров.

Сервисное и сервисно-компонентное программирование основано на системных, сервисных и компонентных ресурсах Интернет. Сервисы Интернета представляют собой набор ресурсов, которые реализуют некоторые общие функции в службе Интернет (сервисы транзакций, именования, безопасности, защиты, надежности и др.). Веб-сервис имеет URL-адрес, интерфейс и механизм взаимодействия с другим сервисом через протоколы Интернет или связи с другими программными элементами БД и деловыми операциями. Обмен данными между веб-сервисом и КПИ осуществляется с помощью XML-документов, оформленных в виде сообщений. Веб-сервисы обеспечивают решение задачи *интеграции (сборки) приложений и систем* разной природы, являясь при этом инструментом построения распределенных систем. К основным средствам описания и разработки новых систем для предметных областей относятся:

1) язык XML для описания и построения SOA-архитектуры;

2) язык WSDL (Web Services Description Language) для описания веб-сервисов и их интерфейсов в XML, а также типов данных, сообщений и протоколов связи сервисов;

3) SOAP для определения форматов запросов к веб-сервисам;

4) SCA для создания более сложной системы на основе компонентов и сервисов;

UDDI для описания и интеграции сервисов, их хранения в библиотеках

Для обеспечения безопасности, защиты и надежности создаваемых технических и программных систем (ТПС) для научных областей знаний выполняются процессы:

1) Верификации и тестирования сервисных ресурсов и создаваемых из них Web-приложений ТПС с целью сбора данных об исключительных ситуациях (ошибках, дефектах и отказах, атаках и др.) в специальных таблицах при моделировании и функционировании задач систем.

2) анализа рисков и угроз ресурсов и ТПС для обеспечения устойчивой и безотказной работы ТПС, а также оценки надежности (Гост 351901-2002) работы веб-серверов, серверов приложений, систем управления базами данных (БД) и серверов клиента сервера при обмене сообщениями.

3) Оценивание влияния сформировавшихся ошибочных ситуаций на взаимодействие клиента с сервером (и наоборот) и сервера с клиентом при функционировании отдельных задач и ресурсов ТПС.

4) Оценивание надежности на основе собранных сведения об ошибках, дефектах и отказах, а также сертификата качества при отсутствии ошибок.

1. Научные основы технологии и инженерии программирования систем.

Анализ состояния и развития

Первым ученым, который рассматривал теоретическое программирование, как часть математической науки, объектом изучения которой являются математические абстракции алгоритмов программ на формальном языке, был академик А.П.Ершов (1974) и академик В.М.Глушков. А.П.Ершов считал, что цель математики состоит в формировании математически точного определения понятия программы, метода представления и реализации функций (задач) предметных областей знаний в ЯП.

Задача и метод формулируются в рамках теории предметной области, в которой формальная модель понятий сочетается с общематематическим знанием этой предметной области знаний.

Алгоритмизация программы – это отображение точного знания о задачах и способах выполнения операций над элементами области знаний последующей обработки в компьютерной среде.

Термин «программа» впервые возник на первых ЭВМ СССР, для которых начали сразу программировать математические, физические и производственные задачи. Первоначально алгоритм математической задачи задавался с помощью схем Ляпунова, в виде конечного ориентированного графа, в вершинах которого размещались функции математической задачи, которые задавались формальными символами, формулами и математическими операциями из множества базовых. Для схемы была определена полная система преобразований формул в последовательность выполняемых операций на ЭВМ для получения выходных значений заданных переменных.

Такую схему программы кодировали вручную, чтобы решить математическую задачу.

Как писал академик Глушков В.М. в статье (Об одном методе автоматизации программирования.- М.: Проблемы кибернетики, 1957, №2.-с.181-184) задача математика заключалась в том, чтобы расчленить предстоящие вычисления математических задач на

отдельные элементарные операции для ЭВМ, как при ручном счете. Набор таких описаний для аналитических вычислений задач решения дифференциальных уравнений (например, дифференцирования и интегрирования элементарных функций) оформлялись как стандартные подпрограммы Библиотеки и обрабатывались программирующими программами (ПП) на первых ЭВМ. Так для решения задачи по методу Адамса-Штермера или Дирихле с помощью стандартных подпрограмм задавались граничные значения с помощью кусочно-элементарных вычислений. Для автоматизации новых математических задач создавались специализированные ПП, которые реализовывали новые формулы метода решения и их размещение в Библиотеки. На первых порах наука программирования включала решение следующих задач программирования для ЭВМ:

– описания базовых понятий объектов (программ, объектов, модулей, компонентов, сервисов и др.);

– метода алгоритмизации задач предметных областей;

– формального задания систем программирования ПО для обработки программ в ЯП, проверки правильности описания объектов предметной области и ОС управления программами и данными.

Инженерия и технология программирования

В советский период использовались термины – техника и технология. Термин техника всеобщий, он употреблялся в сфере технических наук, связанных с изучением и созданием технических устройств, оборудования, автомобилей, самолетов и т.д. Двигателем прогрессивного развития любой научной области знаний является технология, в том числе и при создании ЭВМ, автоматизации математических задач, систем управления предприятиями и организациями, АСУ ТП, САПР и др.

Программная инженерия (Software engineering) – способ, техника создания программного обеспечения (ПО) для вычислительных машин;

Технология (от др.-греч. τέχνη — искусство, мастерство, умение; λόγος — мысль, методика, способ производства) — совокупность методов, процессов и материалов, используемых в какой-либо отрасли деятельности, а также научное описание производственных процессов.

Технология программирования – это совокупность методов, способов, приемов, средств автоматизации, технологического оснащения и порядка их использования для разработки, производства программных продуктов (ПП) в заданных требованиях к показателем качества. Этот термин начал широко использоваться в СССР, начиная с 1959 года и стал основным направлением создания математических задач и ПО ЭВМ в СССР.

Таким образом, **инженерия и технология** опирались на фундаментальную экспериментальную основу и в результате сформировались научные дисциплины в информатике и в Computer Science. Так термин «инженерия» определен в рамках Computer Science: Computer Engineering, System Engineering, Software Engineering для конструирования компьютеров и программного обеспечения (ПО) и систем.

Инженерия компьютеров – это теория и принципы построения компьютеров, Фреймворков, суперкомпьютеров, вычислительных кластеров и их системного ПО. Основами дисциплинами конструирования ЭВМ - теории Тьюринга, фон Неймана, автоматов, алгоритмов, кибернетики и др., а также использование логико-математического аппарата и теории анализа систем. С помощью этих средств строятся компьютеры, фреймворки, многопроцессорные, макроконвейерные машины, устройства, микроприборы, макроприборы и т.п.

Системная инженерия – это теория, методы и принципы построения ПО компьютеров, информационных, автоматизированных систем для прикладных областей научных знаний. Она представляет собой междисциплинарный подход, включающий теоретические положения, методы и средства, направленные на создание и объединение системных и программистских решений для проектирования задач в разных предметных

областях знаний (математики, физики, биологии, медицины и др.) с целью их решения на компьютерах. Главной задачей системной инженерии является (системное программирование) создание базового ПО компьютеров (ОС, Translators, Compilers, monitors, BD, SUBD и др.) для реализации разных задач организации вычислительного процесса их решения. *Системное инженерия* – это методы, средства и инструменты для создания информационных, вычислительных и интеллектуальных (биологических, генетических и др.) систем на компьютерах и суперкомпьютерах. Системная инженерия получила развитие при создании разного рода АС, физических экспериментов и проблемных областей (физика, биология, медицина и др.) в рамках Европейского проекта Grid, в котором участвуют научные организации более 150 стран мира.

Инженерия ПО – это система методов и дисциплин планирования, разработки, эксплуатации и сопровождения ПО, предназначенного для промышленного производства (www.swebok.com). Эта инженерия (или системное программирование) охватывает все аспекты создания ПО от начала формулирования требований и до разработки, сопровождения и окончательного списания его. Основу этой научной дисциплины составляет теория алгоритмов и программирования, методы вычислений и коммуникаций технических и программных средств. Массовое сопровождение готовых ПП проводится по плану и менеджменту процессов и ресурсов, верификации и тестирования, оценивания рисков и качества. Инженерия ПО – базовая основа операционных, программных и информационных технологий (ИТ).

Системное программирование (СП). Его основу составляет ПО и ОС, трансляторы с ЯП, Отладчики, тестировщики и компоновщики из простых программ более сложные. Далее рассматривается история возникновения системного программирования для отечественных ЭВМ. Методы СП обеспечивают постановку задач предметной области, поиск принципиальных технологических решений по моделированию системы и изготовлению, проведению анализа и программ и данных на правильность задания с устранением ошибок и формирования варианта ПО ОС с новыми программами.

Таким образом, в 70-годах XX века в связи с появлением ЭВМ для решения различных научно-технических и системных задач в ряде институтов СССР был создан отдел ***программирования*** ПО компьютеров, ОС, компиляторов и диспетчеров (на ВЦ Украины Ющенко Е.Л.- !959: ИПМ М.Р. Шура-Бура – 1962; СОАН ВЦ – А.П.Ершов -1960; ЛГУ С.С.Лавров – 1964 и др.). Они реализовывали СП для разных отечественных ЭВМ. И сегодня Институт системного программирования, созданный в 1994, занимается СП и по отдельным аспектам получил видное место в мировом научном и индустриальном сообществе по технологическим основам СП. ИСП занимается вопросами программирования базового ПО - Software (ОС, трансляторов, отладчиков, верификаторов и др.), технологией программирования прикладных систем и инструментов программирования (анализаторов, верификаторов, тестировщиков и др.) на коммерческой основе.

В современных условиях главной задачей функционирования ПО нам компьютерах является обеспечение безопасности и надежной работоспособности ОС и систем программирования, борьба с киберугрозами, атаками и исправления разного рода ошибок, возникающих при выполнении прикладных задач.

Рассмотрим базис ПО - систем программирования и ОС для отечественных ЭВМ

Системы программирования для отечественных ЭВМ

В 60-х появились языки программирования - ЯП (Алгол, Кобол, Фортран, Пролог, Смолток, Ада, Модула и др.), которые содержали формальные средства для описания алгоритмов и данных независимо от кодов ЭВМ. Чтобы ЭВМ понимал программу в некотором ЯП потребовалось разрабатывать компиляторы (трансляторы) для преобразования формального описания программ в ЯП в код ЭВМ. В стране были реализованы *системы программирования*, включающие ОС для управления ПО компьютера

и трансляторов (1960-1966) первоначально с универсального ЯП Алгол-60 (ТА) в ряде институтов АН СССР: ТА-1 (Лаврова С.С.), ТА-2 (Шура-Бура М.Р.), ТА-3- Альфа (Ершов А.П); ТА-4 Алгол и Кобол, (Ющенко Е.Л.) и др. (http://ox,tv/20181122AF), Отечественные и зарубежные трансляторы проводили анализ программ в ЯП и трансформацию формального описания алгоритма программы в код соответствующей ЭВМ (IBM, MS, М-20, БЭСМ и др.) с проверкой правильности описания программы в ЯП и сбора данных о возникающих ошибках, выполняемых с помощью отладчиков систем программирования, Обнаруженные ошибки устранялись и программа транслировалась для выполнения и последующей проверки и поиска семантических ошибок.

ЯП (1-4 поколения) и отечественные системы программирования (трансляторы, отладчики) и средства тестирования использовались на ЭВМ до 1992, после чего начался период информатизации и формирование новых подходов к реализации экономических, хозяйственных, банковских и разных научно-технических задач в основном на зарубежных компьютерах и системах программирования (IBM, MS.NET, Apple, Unix, Linux и др.).

Операционные системы на отечественных ЭВМ

Для отечественных ЭВМ, М20, БЭСМ-6, Стрела, Днепр-2 и др. были разработаны операционные системы (ОС), которые управляли созданием и выполнением систем программирования, и решением задач по программам, представленных в ЯП (Алгол-60, Фортран, Кобол).

1) ОС- 68/Диспетчер 68 (Л.Н.Королев, В.П. Иванников, МГУ и др.) для БЭСМ-6 выполняла функции:

- Управление каналами связи, устройствами и ресурсами;
- Мультипрограммное решение задач;
- Параллельное выполнение задач и устройств;
- распределение и буферизация операций ввода – вывода данных и программ;
- диалоговый режим работы, разработчиков программ.

Академик Иванников В.П. отдал много лет своей жизни для создания ОС на БЭСМ-6 и развития методов поиска ошибок в исходных программах на ЯП. Вариант ОС-68 получил развитие НД-70 и использовался для управления полетом космических аппаратов, баллистических и телеметрических программных комплексов более 20 лет. Дальнейшим развитием ОС была система управления многомашинными комплексами АС-6, которая, кроме названных функций, выполняла объединение и взаимодействие разных рода ЭВМ (например, БЭСМ-6, УВК Днепр-2 и др.).

2) ОС ИПМ БЭСМ-6 функционировала с 1970 г. (И.Б. Задыхайло, С.С. Камынин и Э.З. Любимский). В этой ОС реализовано управление ресурсами со средствами синхронизации процессов и управления аварийными ситуациями, а также управление системой программирования ТА-2 и отладкой программ на Алгол, Фортран и АЛМО. Были разработаны компоненты ОС – монитор, супервизор, обеспечивающий совместную работу разных устройств, систему прерывания и синхронизации сумматора, регистров, внешних устройств и АЦПУ.

3) ОС Монитор - Дубна БЭСМ-6 (Н.Н. Говорун и В.П. Шириков) выполняла функции управления заданиями, созданием и ведением библиотечных программ, в том числе и Европейских программ CERN, трансляцию программ на ЯП (Фортран, Алгол, Паскаль и др.). Был реализован аппарат управления системой прерывания, защитой памяти, режимом мультидоступа в реальном времени и разделении времени и др.

Для отечественных компьютеров было создано ПО (ОС, трансляторы, обслуживающие программы), которые управляли вычислительным процессом.

4) АИСТ-0 для разделения времени на М-220 и Минск-22 (Ершов А.П., Поттосин И.В., Бабецкий Г.И. и др., 1967-1971) и включала диспетчер системы, системные программы

общего назначения и языково-независимые средства оптимизации в системе БЕТА. Последняя, позволяет транслировать программы с высоким уровнем оптимизации на языках Фортран, Алгол-68, Симула-67, Паскаль и др. Средства создания многоязычной транслирующей системы, превосходили существующие за рубежом. Они были встроены в ДИСПАК БЭСМ-6 и внедрены в проектах ВПК Министерства радиопромышленности.

5) Система программирования для Минск-3.1, 3.2 (Алгол, Кобол) под руководством Приживайского В.И. и группой программистов Романовский С.А. и др.

6) СП Ленинград СП с языков Алгол-60, 68 под руководством Терехова А.Н. и др.

Таким образом, для отечественных ЭВМ были сделаны ОС и системы программирования высокого уровня, которые до 1992 года использовались практически в нашей стране при создании на ЭВМ программных, информационных и прикладных систем во многих организациях страны.

1.1. Математическая теория моделирования прикладных систем Г. Буча (1987)

Математическая теория моделирования систем из объектов Г. Буча основана на детонационной теории Фреге и базовых понятиях ООП вида:

Класс – совокупность объектов с общими свойствами.

Метод (или функция) с операциями над экземплярами объектов класса.

Наследование – сохранение атрибутов и операций родительского объекта класса.

Инкапсуляция – доступность к методам объекта с изоляцией особенностей реализации на компьютере.

Полиморфизм – возможность оперировать с объектами класса без ограничений.

Объекты группируются в классы. *Класс* - это множество объектов, имеющих общие переменные, структуру и поведение. Объект - экземпляр класса. Его поведение определяется операциями создания, уничтожения и сериализации. Внешние переменные и методы класса задаются в интерфейсе для обеспечения взаимодействие экземпляров. Объект задается логико-математическими формализмами в графовой ОМ. Согласно теории Г. Буча весь материальный мир состоит из объектов разной природы, которые связаны между собой некоторым множеством отношений: <объектная ориентация> = <обьекты> + <наследование>.

Любой экземпляр определенного класса предметных областей обладает всеми методами, которые определены в этом классе. Экземпляр класса может быть приведен в соответствие с одним из суперклассов и интерфейсов, реализованных в этом классе. Каждое понятие предметной области – это объект, а вся область - это совокупность объектов со связями. В качестве объекта выступают как абстрактные образы, так и конкретные физические предметы или группы предметов с указанными общими свойствами и функциями.

Развитием ООП является UML (Unified Modeling Language). Он позволяет проектировать системы с помощью визуальных диаграмм (use cases) вида:

- вариант использования;
- класс;
- состояние, деятельность, взаимодействие;
- последовательность и кооперация;
- реализация компонентов и развертывания для выполнения и др.

Диаграммы или прецеденты use case применяются для создания следующих моделей прикладной системы:

1) *структурная* (статическая) модель, которая задает понятия системы, включая классы, интерфейсы, отношения и атрибуты;

2) *модель поведения* (динамическая), которая задает поведение объектов с помощью диаграмм взаимодействия и изменения состояний объектов и систем;

3) *модель функционирования,* которая задает операции вычисления элементов прикладной системы.

Данные модели и модель ОМ задают каркас проектируемой системы. Для определения поведения классов объектов используются диаграммы состояний и деятельности. Объекты ОМ могут располагаться в узлах компьютерной сети и выполняться в соответствующей операционной среде Интернет.

1.2. Методы математической спецификации задач предметных областей знаний

К формальным методам спецификации относятся: Венский метод VDM (Vena Development Method) D. Biorner, Z-метод (I.R. Abrial, B. Meyer), RSL, RAISE, Monadic Logic2 (Bruno) и др.

Языки спецификации задач классифицируются по представлению предметных областей:

– модель-ориентированные спецификации. Их целью является построение абстрактной модели специфицированной ИС и описание состояний объектов (*state-based*) с помощью аксиом, ориентированных на свойства объектов. Например, VDM, Z, RAISE, B и др.;

– алгебраические методы спецификации систем в терминах действий (*action-based*), ориентированные на описание наборов базовых типов данных и построении модели системы. Например: OBJ, Larch, Anna, Clear, ateCharts, CSP, CCS.

Языки спецификаций классифицируются следующим образом:

– аксиоматические спецификации;
– спецификация моделей;
– алгебраические спецификации;
– темпоральные (временны́е) логические спецификации;
– параллельные спецификации;
– монадическая спецификация логики второго порядка.

Аксиоматические спецификации определяют поведение объектов с помощью логических формул, а также задают входные, промежуточные и выходные утверждение. Спецификация включает операции связи между входными и выходными параметрами, аксиомы с пред- и постусловиями. Эти спецификации поддерживают языки: VDM, Anna и Z.

Спецификации описания модели описывают поведение в терминах модели. Спецификация использует четко определенные типы (множества, последовательности, отношения, функции) и определяет операции на моделях. Спецификация включает модель типа, инвариант, пред- и постусловия. Особенностями таких спецификаций является явное моделирование состояний в моделях, возможность выполнения и построения объектов в иерархическом порядке. Спецификации абстрактной модели поддерживаются на языках VDM, Z, , RAISE, UML/OCL.

Алгебраические спецификации определяют поведение совокупности отношений эквивалентности, описывающих свойства объектов и операции над ними. Спецификация включает функции и отношения. Особенностью таких спецификаций является наличие описаний функций и абстрактных типов данных. Они поддерживают такие языки, как OBJ, Larch, Anna, Clear.

Темпоральные (временные) спецификации описывают поведение системы набором состояний и определяют операции перехода между состояниями или отношениями. Они задают общее пространства состояний, текстовые и графические заметки, модульность и управление системами. Временны́е логические спецификации поддерживаются языками Temporal Logic Specification (TLS), Petri nets, GIL CSP, UML/OCL (диаграммы конечных

автоматов, деятельности, коммуникации, взаимодействия, последовательности, синхронизации).

Параллельные спецификации определяют действия в терминах параллельных событий. Спецификация включает состояния, переходы, события, упорядочения и синхронизацию. Особенности таких спецификаций — мощный механизм спецификации, уровень синхронизации спецификации. К этим спецификациям относятся: StateCharts, CCS, UML/OCL (диаграммы конечных автоматов, деятельности).

Монадическая спецификация логики второго порядка атрибутных грамматик для задания ЯП семантическими правилами представления программ (AhoLSU, Cre). Правила связывают вершины графа с деревом вывода и задают traces1, «следы» графа, которые представляются графиками для использования конечными автоматами(DiekRoz). Язык формального описания схем программ и трасс служит для моделирование биологического развития объектов (Bruno Courcelle and Joost Engelfriet Graph Structure and Monadic Second-Order Logic, a Language Theoretic Approach (http://www.vikipedia.org/logic_of_graths). Этот язык используется для описания программ компиляторов. К средствам описания относятся ЯП и их грамматики, теория автоматов и преобразователи текстов. В этом языке можно задавать последовательные и параллельные программы в классе биологических, медицинских и других прикладных задач.

Язык VDM появился в Венской лаборатории (1990) как метод описания транслятора с языка ПЛ/1 и систем со сложными структурами данных. VDM имеет математическую символику для спецификации данных задач: X — натуральные числа с нулем; N — натуральные числа без нуля; Int — целые числа; Bool — булевый тип; Qout — строки символов; Token — знаки и специальные обозначения операций. *Функция* задает вычисление математических задач (например, минимальное значение двух переменных: min (N_1, $N_2 = N_3$). *Объекты* - множества, деревья, последовательности, отображения и другие формируемые новые сложные объекты. При работе с объектами множества используются математические операции $\in$, $\subseteq$, $\cup$, $\cap$ и др., а над числами - арифметические операции (+, - x, / и др.).

При реализации спецификаций задач средствами VDM используются *предусловия, постусловия, аксиомы и утверждения,* необходимые для проведения доказательства правильности выполнения программы. Язык VDM активно использовался для описания моделей математических задач в академических и научных организациях.

Утверждение задает описание операций по проверке правильности программы в разных ее точках. Операторы программы изменяют ее состояние, а операции утверждений извлекают информацию из нее после изменения состояния (например, после операции работы с БД). На основе этой информации устанавливается правильность или неправильность выполнения операции.

Постусловие — это предикат, который является истинным после выполнения предусловия, завершения текущей операции и выполнения инвариантных свойств программы.

Метод VDM ориентирован на пошаговую детализацию спецификации программ. На первом уровне строится грубая спецификация — модель программы в языке VDM, которая постепенно уточняется, пока не получится окончательный текст в ЯП. Разработка спецификации проводится по следующей схеме:

– выбор терминов для спецификации программ;

– описание понятий и объектов с использованием денотат, идентифицируемых с помощью некоторого имени;

– описание инвариантных свойств программы;

– определение операций (например, ввод, удаление объектов и др.), которые изменяют состояние программы и сохраняют инвариантные свойства.

При переходе от одного шага детализации к другому модель программы детализируется и постепенно становится ближе к конечному описанию. Функции программы уточняются при детализации структуры программы и задания поведения модели.

Пример. Описание задачи поиска имени компонента C в каталоге (catalR) репозитария компонентов; сравнение заданного имени в запросе пользователя, извлечение и передача пользователю.

1. repoz :: = developers : dl → **Init —set**
2. catalR : cat → **Init —set**
3. role : **inst** → facet
4. facet :: = autors: Milk
5. title : N
6. user :: = developer : **Itn**
7. free : **Bool**,

где developers — разработчик компонента C;

facet — переменная, в которую посылается код компонента, выбранного из каталога catalR репозитория repoz при совпадении имен в каталоге и запросе;

role — переменная, в которой хранится текущий элемент из репозитария, найденный по фасете компонента с номером N для user;

authors: Milk — имя разработчика компонента;

free : Bool — переменная, которая используется для задания признака: компонент не найден или к нему никто не обращался.

Формальное описание моделей в *Z*

Язык спецификации *Z*-схема задает модель системы в виде ациклического графа с помощью композиционной теории и задания внешних характеристик и параметров моделей. Созданная модель системы включает:

– совокупность *Z*-схем с набором деклараций и ограничений;

– инварианты схем для верификации модели.

При определении обобщенной модели выполняется:

– выбор модуля из библиотеки шаблонов, переименование шаблонов в соответствии с определением узла графа;

– связь модулей через интерфейс *порта* в модульную структуру системы и после конкретизации *слота* отмечаются меткой в *Z*-схеме;

– верификация связей модулей, их атрибутов и типов данных;

– создание нового шаблона, инкапсуляция его интерфейса и соответствующей схемы.

Графовая модель Z задается в виде ациклического графа, узлы которого — модули, а дуги — интерфейс между модулями. Каждому модулю модели соответствует сервис провайдера (*provider*) и потребителя (*consumer*) услуг. Дуга графа от модуля *M* к *N* определяет интерфейс, который может предоставлять модуль *N* для модуля *M*. Концепция сервиса поддерживается протоколами сети при взаимодействии компонентов, которые транспортируются через сеть в целях отправки множеств значений параметров серверу сети.

Спецификация интерфейсов модулей удовлетворяет следующим свойствам:

– разделение (*separable*) модулей с точки зрения проектирования и спецификации без описания среды модуля;

– композиция (*composition*), когда модули соединяются с помощью механизмов композиционной теории идентичной сборочной теории в виде системы с гарантированными связями между ними.

Для взаимодействия модулей используются сетевые протоколы (TCP, UDP и др.) для нахождения требуемого сервиса. Модель интерфейса задает связь модуля со средой в виде дискретного события (*event*), которое возникает только тогда, когда модуль и среда

действуют вместе, и создают событие, на стороне интерфейса. Таким образом, интерфейс специфицирован как множество последовательностей событий для наблюдения за поведением модулей модели. Спецификация S включает в себя все возможные случаи поведения и задает разные ситуации, которыми могут быть:

– событие интерфейса или состояние наблюдателя системы;

– анализ, который является конечным или представлен неопределенной последовательностью;

– формальные средства определения этой последовательности;

– условия выполнения событий.

Эти предположения разрешают обеспечить разноуровневую систему взаимодействия модульных элементов модели через механизм протоколов и систему наблюдения за событиями.

Проект модели VGM системы состоит из двух модулей: модуля управления CONT и модуля распределения памяти STOR. Модуль CONT имеет следующую спецификацию действий:

CONT = (coin ——> reguest ——> response ——> choc ——> cont)

Общее назначение этой спецификации состоит в том, что потребитель mou вставля данные в coin для отправки запроса (*reguest*) модулю памяти. Этот модуль дает ответ (*response*) на запросы и имеет следующую спецификацию:

STOR = (reguest ——> response ——> stor)

Модель VM определяет параллельную обработку модулей CONT, STOR, а также связь со средой. При этом спецификация VM имеет вид:

M = (CONT || STOR) \ {reguest, response} = (coin ——> choc ——> VM).

Язык Clear – это алгебраический язык для описания виртуальной графовой модели VGM с отмеченными узлами в вершинах и их внешними характеристиками и параметрами. По этой модели осуществляется описание интерфейса для связи вершин графа и описанием ситуаций или событий при их выполнении. Эти средства стали использоваться при описании программ и систем в ПЛ.1 и др. ЯП на объектной основе.

Monadic Logic2 (http://www.vikipedia.org/logic_of_graths) для описании программ компиляторов. К средствам описания относятся ЯП и их грамматики, теория автоматов и преобразователи текстов. В этом языке можно задавать последовательные и параллельные программы в классе биологических, медицинских и других прикладных задач.

1.3. Метод графового моделирования программных структур

Теория программирования – это математическая наука, объектом изучения которой являются математические абстракции функций программ с определенной логической и информационной структурой, ориентированной на исполнение на ЭВМ. С появлением ЯП начали разрабатываться новые методы анализа алгоритмов задач прикладных областей (ПрО), декомпозиции областей на отдельные функциональные объекты задач, отображения их в вершинах графа для создания по нем сложной структуры ПрО (комплекс программ, агрегат, большая программа, система и др.).

Этот метод возник при участии в разработке комплексок программ Прометей, Руза, Яуза в проекте ВПК под руководством В.В. Липаева в 80-х годах XX столетия. Функциональные элементы разрабатываемых новых систем конкретного применения начали называть называть программными модулями, потом объектами, компонентами, сервисами и др. (Липаев В.В. История развития отечественного программирования для специализированных ЭВМ в 50-80-е годы.- Синтег. Москва.-2003.)

Графовая структура систем начала использовать в рамках проекта ВПК с помощью модулей, которые адекватны объектам, компонентам, сервисам, аспектам, используемых в сложных программных структурах и комплексах, создаваемых из них(см. Лаврищева Е.М.

Грищенко В.Н. Связь разноязыковых модулей в ОС ЕС (IBM).-1980.-Москва.- Финансы и стастика.-137с.).

Модуль – это программный элемент, описанный в некотором ЯП, и отображает реализацию некоторой функции ПрО со свойствами связанности с другими элементами по данным, задаваемым в интерфейсной части описания. С математической точки зрения программный ***модуль*** отображает множества исходных данных X во множество выходных данных Y в виде:

$M : X \rightarrow Y.$

На X, Y и M накладывается ряд ограничений и условий, позволяющих сделать модуль самостоятельным программным элементом среди других названных объектов. К *видам связей* модулей через link входных и выходных параметров относятся:

1) связь по управлению: $CP = K_1 + K_2$,

где K_1 - коэффициент механизма вызова, K_2 – коэффициент перехода от среды вызывающего модуля к среде вызываемого;

2) связь по данным: $CI = \sum_{i=1}^{n} K_i F(x_i)$,

где K_i – весовой коэффициент i-го параметра, $F(x_i)$ – функция элемента с параметром x_i. Коэффициенты $K_i = 1$ – для простых типов переменных и $K_i > 1$ – для сложных типов переменных (массив, запись и др.). $F(x_i) = 1$, если x_i – простая переменная и $F(x_i) > 1$ – если сложная.

Программные графовые структуры из модулей

Программная, модульная структура задается *графом* $G = (X, E)$,

где X – конечное множество вершин; E – конечное подмножество прямого произведения X x X x Z на множестве отношений на дугах графа. Программная структура представляет пару

$S = (T, \chi)$,

где T – модель программной модульной структуры; χ – характеристическая функция, заданная на множестве вершин X графа G.

Значение характеристической функции χ определяется так:

$\chi(x) = 1$, если модуль с вершиной $x \in X$ включен в состав модульной системы;

$\chi(x) = 0$, если модуль с вершиной $x \in X$ не включен в состав модульной системы и к нему нет обращения из других модулей.

Определение 1. Две модели программных структур $T_1 = (G_1, Y_1, F_1)$ и $T_2 = (G_2, Y_2, F_2)$ тождественны, если $G_1 = G_2$, $Y_1 = Y_2$, $F_1 = F_2$. Модель T_1 изоморфна модели T_2, если $G_1 = G_2$ между множествами Y_1 и Y_2 существует изоморфизм φ, а для любого $x \in X$ $F_2(x)=\varphi(f_1(x))$.

Определение 2. Две программные структуры $S_1 = (T_1, \chi_1)$ и $S_2 = (T_2, \chi_2)$ тождественны, если $T_1 = T_2$, $\chi_1 = \chi_2$ и структуры S_1 и S_2 *изоморфны*, то T_1 изоморфна T_2 и $\chi_1 = \chi_2$.

Понятие изоморфизма программных структур и их моделей используется при спецификации уровня абстракции, на котором определяются операции над этими структурами. Для изоморфных объектов графа операции будут интерпретироваться одинаково без ориентации на конкретный состав программных элементов при условии, что такие операции определяются над парами (G, χ).

Программный модуль описывается в ЯП и имеет раздел интерфейса, в котором задаются внешние и внутренние параметры для обмена данными между связанными модулями через операции вызова Call/RMI и др.

Интерфейс обеспечивает связь разнородных программных модулей по данным и способу их отображения системами программирования в ЯП операционной среды компьютера. Основные функции: передача данных между программными элементами (модулями) - преобразование данных к эквивалентному виду и переход от среды и платформы

вызываемого модуля к вызывающему и обратно. Функции преобразования отличающихся, неэквивалентных типов данных проводится с помощью ранее разработанной библиотеки 64 примитивных функций для сборки разнородных типов данных ЯП в системе АПРОП и вошедших в общие системные среды ОС (IBM, MS, Oberon, UNIX и др.).

Сборка программных модулей выполняется операциями (link, make, assembling, config) с использованием специальных программ библиотеки редактора связей в общесистемных средах (ОС ЕС, IBM, MS и др.), конструкторов сложных программ в ОС РВ для СМ ЭВМ, комплексаторов модулей для МВК «Эльбрус» и др. К специальным программам относятся интерфейсные функции преобразования типов данных библиотек ОС.

Далее рассматривается математическая теория графового моделирования программных структур из модульней, построения разных структур программ с помощью математических операций (объединения, проекции, разности и др.) и реализации связей модулей графа с механизмом преобразования передаваемых данных между вершинами графа.

Определение модульных структур по графу

Для представления программных структур из модульных элементов используется математический аппарат теории графов, в котором граф *G* трактуется как пара объектов G = (X, E), где X – конечное множество вершин, а E – конечное подмножество прямого произведения $X \times X \times Z$ - дуг графа, соответствующих конечной вершине (рис. 1).

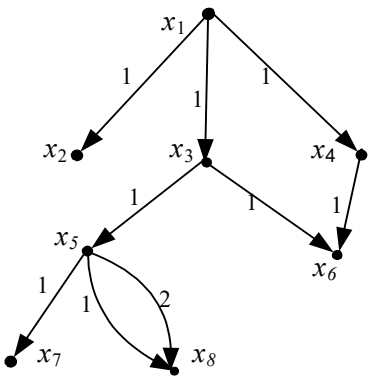


Рис.1. Граф программы из модулей x_8

Множество дуг графа имеют вид: E = *{(*x_1, x_2, 1), *(*x_1, x_3, 1), ..., *(*x_5, x_8, 1), *(*x_5, x_8, 2)}. Исходя из этого определения, можно сказать, что граф G является мульти графом, так как две его вершины могут быть соединены несколькими дугами. Для отличия таких дуг вводится их нумерация целыми положительными числами – 1, 2… (рис.1) и вершины графа x_1, x_2, ..., x_8 образуют множество *X.* Из модуля, соответствующего вершине x_5, имеются два оператора вызова к модулям с вершинами x_7, x_8.

Определение 3. Программный агрегат – это пара *S = (T, χ),* где *T* – модель программной структуры агрегата из модулей; χ – характеристическая функция, определенная на множестве вершин *X* графа модульной структуры G. Значение функции χ определяется следующим образом:

- *χ(x) = 1*, если модуль, соответствующий вершине $x \in X$, включен в состав агрегата;
- *χ(x) = 0*, если модуль, соответствующий вершине $x \in X$, не включен в состав программного агрегата, но к нему имеются обращения из других модулей, ранее включенных.

Определение 4. Модель программной структуры агрегата – это объект, описываемый тройкой *T = (G, Y, F),*

где *G = (X, E)* – ориентированный граф, являющийся графом модульной структуры; *Y* – множество модулей, входящих в программный агрегат; *F* – функция соответствия, ставящая каждой вершине *X* графа *G* элемент множества *Y.*

Функция *F* отображает *X* на *Y:* $F : X \rightarrow Y$. (1)

В общем случае элементу из *Y* может соответствовать несколько вершин из множества *X* (что характерно для динамической структуры агрегата).

Граф программных агрегатов обладает следующими свойствами:

1) граф *G* имеет один или несколько элементов связности, каждая из которых представляет ациклический граф, т. е. не содержит ориентированных циклов;

2) в каждом графе *G* выделена единственная вершина, которая называется корневой и характеризуется тем, что входящих в нее дуг не существует и соответствующий ей модуль программного агрегата выполняется первым;

3) циклы допускаются только для случая, когда некоторая вершина имеет рекурсивное обращение к самой себе. Обычно такая возможность реализуется компилятором с соответствующего ЯП и данный тип связи не рассматривается межмодульным интерфейсом. Поэтому такие дуги не включаются в граф. Исключение из рассмотрения других типов циклов связано с тем, что некоторые модули должны будут запоминать историю своих вызовов, чтобы правильно вернуть управление, а это противоречит свойствам модулей;

4) пустой граф G_0 соответствует пустой программной структуре.

Далее граф G будем использовать для иллюстрации математических операций над модульными структурами. На рис.2. показаны три вида графов и приводится их описание.

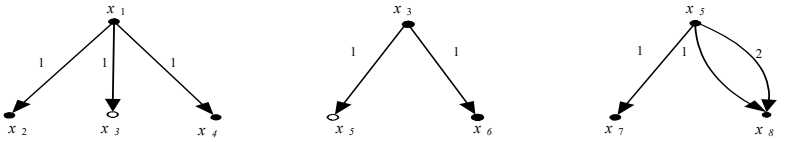


Рис. 2. Графы модульных структур трех видов

Подграф - это фрагмент программного агрегата $G^s = (X^s, E^s)$, для функций которого выполняется одно из двух условий:

$C(S) = 1$, если $\chi(x) = 1$ для любых x из X;

$C(S) = 0$, если существует x такое, что $\chi(x) = 0$;

$R(S^s) = 0$, *если структура* из модулей входит в состав программной структуры более высокого уровня и $R(S)=1$, если программный агрегат готов к выполнению.

С учетом этих комбинации C и R подграф может быть: открытый $(C = 0, R = 0)$; замкнутый сверху $(C = 0, R = 1)$; замкнутый снизу $(C = 1, R = 0)$.

Граф программного модуля (m) представляется в виде: $G^m = (X^m, E^m)$. В нем содержится единственная вершина $x \in X^m$, для которой $\chi(x_j)=1$. Данная вершина является корневой. Дуга вида (x_j, x_e, k), означает вызов модуля к соответствующей вершине x_j, т.е. к модулю с вершиной x_l.

Темный кружок на графе соответствует вершине, для которой $\chi(x) = 1$; светлая – $\chi(x)=0$.

Граф программы $G^p = (X^p, E^p)$, в которой выполняется $C(S^p) = 1$; $R(S^p) = 1$. Пример графа такой программной модульной структуры приведен на рис. 1.

Граф комплекса $G^c = (X^c, E^c)$ состоит из n компонентов связности $(n > 1)$, каждая из которых является графом и включает: $G^c = G_1^p \bigcup G_2^p \bigcup_{, \dots ,} \bigcup G_n^p$, где $X^c = X_1^p \bigcup X_2^p \bigcup \dots \bigcup X_n^p$ и $E^c = \Gamma_1^p \bigcup \Gamma_2^p \bigcup \dots \bigcup \Gamma_n^p$. Данные определения графа программного модуля, программы и комплекса используются для проведения процесса сборки модулей.

1.3.1. Матричное представление графов программных структур

Для определения основных операций над программными структурами используется математический аппарат матричного представления графов в виде матрицы смежности и достижимости. Т.е. граф представляется матрицей $M = m(i, j)$ *смежности,* в которой $m_{ij}=1$ тогда и только тогда, когда в графе имеется ребро, ведущее от вершины I в вершину j. Матрица *инцидентности* (достижимости) $M = m(i, j)$ задает $m_{ij}=1$, если ребро графа j входит в вершину I и

m_{ij}=0 в остальных случаях. Элемент матрицы смежности m_{ij} определяет количество операторов вызова с индексом i к модулю с индексом j.

Кроме матрицы смежности (вызовов) используется характеристический вектор $V_i = \chi(x_i)$ для задания i -элементов. Для графа модульной структуры (рис. 1) характеристический вектор V и матрица смежности M имеют вид:

$$V = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \qquad M = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad (2)$$

Проведем анализ матриц смежности и характеристических векторов для подграфов и графов программных структур из модулей, соответствующих различным типам – программа, комплекс, агрегат и др. Для подграфов (рис.2) векторы и матрицы имеют вид:

$$V_3^s = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad M_3^s = \begin{pmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}; \quad V_1^s = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad M_1^s = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix};$$

$$V_5^s = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \quad M_5^s = \begin{pmatrix} 0 & 1 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \qquad (3)$$

Для графа программы (рис. 1) характеристический вектор и матрица вызовов совпадают с V и M соответственно и определяют форму (2), в ней все элементы V равны единице.

В случае комплекса характеристический вектор и матрица вызовов имеют следующий вид:

$$V^c = \begin{pmatrix} V_1^p \\ V_2^p \\ \dots \\ V_n^p \end{pmatrix}, \quad M^c = \begin{pmatrix} M_1^p & 0 & \dots & 0 \\ 0 & M_2^p & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & M_n^p \end{pmatrix} \qquad (4)$$

Здесь V_i^p и M_i^p ($i = \overline{1, n}$) обозначают характеристический вектор и матрицу смежности для графа i-й программы, входящей в граф комплекса. В дальнейшем матричное представление используется при выполнении математических операций над программными структурами.

Отношение достижимости графов программных структур

Пусть $G = (X, E)$ – граф программной структуры из модулей x_i, x_j – вершин, которые принадлежат X. Если в графе G существует ориентированная связь от x_i к x_j, то вершина x_j - достижима из вершины x_i.

Справедливо следующее утверждение: если вершина x_j достижима из x_i, а x_l – из x_j, то x_l достижима из x_i. Доказательство этого факта очевидно.

Рассмотрим бинарное отношение на множестве X, которое определяет достижимость одной вершины графа к другой.

Введем обозначение $x_i \rightarrow x_j$ - достижимость вершины x_j из x_i.. Отношение транзитивно. Обозначим через $D(x_i)$ множество вершин графа G, достижимых из x_i. Тогда равенство

$$\overline{x_i} = \{x_i\} \cup D(x_i) \qquad (5)$$

определяет транзитивное замыкание x_i по отношению к достижимости вершин. Докажем следующие теоремы.

Теорема 1. Для выбранного элемента связности графа программной структуры любая вершина достижима из корневой, соответствующей данной вершине графа, т. е. выполняется равенство (x_1 – корневая вершина). $\overline{x_1} = \{x_1\} \cup D(x_1) = X.$ (6)

Доказательство. Предположим, вершина x_i ($x_i \in X$) недостижима из x_1. Тогда $x_i \notin \overline{x_1}$ и множество $X' = X \setminus \overline{x_1}$, не пусто. Поскольку выбранный элемент графа связанный, то существуют вершина $x_j \in \overline{x_1}$ и цепь $H(x_i, x_j)$, ведущая от x_i к x_j. Исходя из ацикличности графа G, в X' должна существовать простая цепь $H(x_l, x_j)$, где в вершину x_l не входят дуги (данная цепь может быть пустой, если X' состоит только из x_i). Рассмотрим цепь $H(x_l, x_j) = H(x_l, x_i) \cup H(x_i, x_j)$. Это означает, что модуль x_j достижим из вершин x_1 и x_i и обе вершины не содержат входящих дуг. А это противоречит определению графа программной структуры из модулей с единственной корневой вершиной. Теорема доказана.

Результаты данной теоремы важны для обоснования требования отсутствия ориентированных циклов в графе программной структуры относительно понятия достижимости. Рассмотрим граф, приведенный на рис. 3. Из этого рисунка видно, что граф содержит ориентированный цикл и модули, соответствующие вершинам x_4, x_5, x_6 никогда выполняться не будут.

Таким образом, результаты теоремы 1 усиливают требование об отсутствии ориентированных циклов в графе программы.

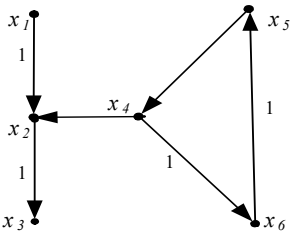


Рис. 3. Граф с ориентированным циклом

Проведем анализ матричного представления отношения достижимости для графа программной структуры (рис.1 с матрицей достижимости A), которая имеет вид (7).

Коэффициент a_{ij} = 1, если модуль, соответствующий индексу l, достижим из модуля, соответствующего индексу i. Результаты основаны на следующей теореме из теории графов.

A=

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad (7)$$

Теорема 2. Коэффициент m_{ij} l-й степени матрицы смежности M^l определяет количество различных маршрутов, содержащих l дуг и связывающих вершину x_i с вершиной x_j –ориентированного графа.

Рассмотрим следующие три следствия из этой теоремы.

Следствие .1. Матрица $\overline{M} = \sum_{l=1}^{n} M^l$, где M – матрица смежности ориентированного графа с n вершинами совпадает с точностью до числовых значений коэффициентов с матрицей достижимости A.

Доказательство. В ориентированном графе, содержащем n вершин, максимальная длина пути без повторяющихся дуг не может превышать n. Поэтому последовательность степеней матрицы смежности M^i, где i = 1,2, …, n определяет количество всех возможных

путей в графе с количеством дуг ≤ n. Пусть коэффициент $\overline{m_{ij}}$ матрицы M отличен от нуля. Это означает, что существует степень матрицы M^i, у которой соответствующий коэффициент m_{ij} также отличен от нуля. Следовательно, существует путь, идущий от вершины x_i к x_j, т.е. вершина x_j достижима из x_i. Данное следствие определяет связь матрицы вызовов графа модульной структуры М, совпадающей с матрицей достижимости A, и определяет алгоритм построения последней.

Следствие.2. Пусть для некоторого i в последовательности степеней матрицы смежности M^i существует коэффициент $m_{ii} > 0$. Тогда в исходном графе существует ориентированный цикл.

Доказательство. Пусть $m_{ii} > 0$ для некоторого l. Следовательно, x_l достижимо из x_i, т.е. существует цикл. Согласно данной теореме, цикл имеет l дуг (в общем случае повторяющихся).

Следствие 1.3. Пусть n-я степень матрицы смежности M^n ациклического графа совпадает с нулевой матрицей (все коэффициенты равны нулю).

Доказательство. Если граф ациклический, то в нем максимально простой путь не может иметь больше чем $n - 1$ дуг. Если в M^n имеется коэффициент, отличный от нуля, то должен существовать путь, состоящий из n дуг. А этим путем может быть только ориентированный цикл. Следовательно, все коэффициенты M^n для ациклического графа равны нулю. Данное следствие предоставляет необходимое и достаточное условие отсутствия циклов в графе модульной структуры.

Для ациклических графов отношение достижимости эквивалентно частично строгому порядку. Транзитивность отношения достижимости рассмотрено выше.

Антисимметричность следует из отсутствия ориентированных циклов: если вершина x_j достижима из x^l, то обратное неверно.

Введем обозначение $x_i > x_j$ если вершина x_j достижима из вершины x_i.

Пусть G = (X, Γ) – ациклический граф, соответствующий некоторой программной структуре. Рассмотрим убывающую цепь элементов частично упорядоченного множества X:

$x_{i1} > x_{i2} > \ldots > x_{in}$. …,

где через “>” обозначено отношение достижимости. Поскольку X конечно, то цепь обрывается $x_{i1} > x_{i2} > \ldots > x_{in}$. Вершина x_{in} не имеет исходящих дуг, т. е. элемент x_{in} минимальный (модуль, который не содержит обращения к другим модулям). Максимальным элементом X есть корневая вершина.

1.3.2. Математические операции над элементами графа

Математические операции (U, ∩, /, +, - , P, C, R) над графов выполняются на уровне абстрактных программных структур, которые приводят к изменению элементов графа и характеристических функций: $S = (G, \chi)$.

Пусть $S_1 = (G_1, \chi_1)$ и $S_2 = (G_2, \chi_2)$ – два графа программных структур $G_1 = (X_1, E_1)$ и $G_2 = (X_2, E_2)$ соответственно. Введем следующие обозначения:

- $D(x)$ – множество вершин, достижимых из вершины x;
- $D^*(x)$ – множество вершин, из которых достижима вершина x.

Для одинаковых вершин, входящих в графы G_1 и G_2, используются одинаковые обозначения. Основные операции над программными структурами рассматриваются ниже.

Операция *объединения* $S = S_1 \cup S_2$ (9)

предназначена для формирования графа структуры комплекса и формально определяется следующим образом S_1 и S_2 – любые программные структуры, удовлетворяющие определениям п.1:

$$G = G_1 \oplus G_2, \quad X = X_1 \oplus X_2, \quad E_1 \oplus E_2 , \qquad (10)$$

где символ $\oplus$ обозначает прямую сумму при условии:

$\chi(x) = \chi_1(x)$, если $\chi \in X_1$,

$\chi(x) = \chi_2(x)$, если $\chi \in X_2$.

Одинаковые вершины, входящие в G_1 и G_2, в операции объединения программных структур представляются разными объектами. При этом характеристический вектор и матрица смежности программной структуры S определяются так:

$$V_{1,2} = \begin{pmatrix} V_1 \\ V_2 \end{pmatrix}, \; M_{1,2} = \begin{pmatrix} M_1 & 0 \\ 0 & M_2 \end{pmatrix}, \qquad (11)$$

где $V_{1,2}$ и $M_{1,2}$ – характеристические векторы и матрицы смежности модульных структур S_1 и S_2 соответственно. Операция объединения ассоциативна, но не коммутативна – порядок следования операндов определяет порядок выполнения модульных элементов комплекса. Необходимо отметить, что если операнды S_1 и S_2 удовлетворяют условиям определения программных структур, то результат S также будет удовлетворять требованиям. Операция объединения увеличивает число элементов связности графа. Кроме того, графы структур сами могут иметь несколько элементов связности. Для остальных операций графы операндов и результата имеют единственный элемент связности.

Операция соединения. Обозначим через x_i и x_j корневые вершины графов G_1 и G_2 программных структур S_1 и S_2 соответственно. Операция соединения

$$S = S_1 + S_2, \qquad (12)$$

выполняется, если данные структуры удовлетворяют следующим условиям:

- множество $X' = X_1 \cap X_2$ не пусто;
- вершина $x_j \in X'$ и $\chi(x_j) = 0$; $D^*(x) \cap D(x) = 0$ для каждого $x \in X'$, где $D^*(x) \in X_1$ и $D(x) \in X_2$;

$$G = G_1 \cup G_2, \; X = X_1 \cup X_2, \; E = E_1 \cup E_2, \qquad (13)$$

Характеристическая функция χ выполняется при условии:

$$\chi(x) = \chi_1(x), \text{ если } x \in X_1 \setminus X';$$

$$X(x) = max\,(\chi_1(x), \chi_2(x)) > \text{ если } x \in X',$$

$$\chi(x) = \chi_2(x), \text{ если } x \in X_2 \setminus X'.$$

Первое условие означает, что в графах G_1 и G_2 имеются общие вершины. Согласно второму условию корневая вершина G_2 принадлежит общей части и для S_1 объект, соответствующий x_j, еще не включен в программную структуру. Третье условие запрещает существование циклов в графе результата. Действительно, если существует $x_n \in D^*(x) \cap D(x)$,то $x_n > x$ и $x > x_n$, то это означает существование цикла. Если S_1 и S_2 удовлетворяют приведенным выше условиям, то операция соединения частичная.

Определим принадлежность результата операции соединения к классу программных структур. Поскольку X' не пусто, то граф G имеет один элемент связности. Корневой вершиной графа G является x_i. Сам граф G не имеет ориентированных циклов, т. е. ациклический. Таким образом, S принадлежит к классу рассматриваемых программных структур. Операция соединения не коммутативная и в общем случае не ассоциативна. Чтобы показать что операция не ассоциативна, рассмотрим результат $S = (S_1 + S_2) + S_3$, где корневые вершины графов G_2 и G_3 входят в состав вершин графа G_1 и $X_2 \cap X_3 \neq 0$. Тогда результат операции соединения $S_2 + S_3$ не определен.

Операция проекции. Пусть $S_1 = (G_1, \chi_1)$ – программная структура и $x_i \in X_1$. Операция проекции этой структуры на вершину графа S_1 обозначается как $S = P_{rxi}(S_1)$ и определяется так:

$$G(X, E), \; X = \overline{x}_i, \; E = \{(x_i, x_j, K) \mid x_i, x_j \in X\}, \qquad (14)$$

и для характеристической функции имеет вид $\chi(x) = \chi_1(x)$, если $x \in X$. Операция проекции определяет программную структуру S_1 в структуре S. Проверим принадлежность структуры S к классу рассматриваемых программных структур. Если граф структуры S_1 связан ациклически, то теми же свойствами будет обладать и граф S. Существует единственная корневая вершина x_i в графе G. Таким образом, программная структура S принадлежит рассматриваемому классу.

Операция разности для программных структур определяется следующим образом. Пусть $S_1 = (G_1, \chi_1)$ – программная структура и $x_i \in X_1$. Операция разности выполняется над этой структурой и ее проекцией на вершину x_i соответствующего графа (x_i не является корневой вершиной графа G_1). Формально операция разности программной структуры имеет вид:

$$S = S_1 - P_{r_{xi}}(S_1), \qquad (15)$$

определяется следующим образом: $G = \{X, E\}$, $X = (X_1 \setminus \overline{x_i}) \cup X'$ (16)

$E = \{(x_i, x_j, K) \mid x_i, x_j \in X\}$,

где множество X' состоит из таких элементов, для которых

$$X' = \{x'_j \mid (x_1 \in X_1 \setminus x_i) \,\&\, (x'_j \in \overline{x_i}) \,\&\, (x_1, x'_j, K) \in E\} \qquad (17)$$

Здесь, характеристическая функция χ определяется так: $\chi(x) = \chi_1(x)$, если $x \in X_1 \setminus \overline{x_i}$; $\chi(x) = 0$), если $x \in X'$. Во множество X включаются вершины, которые не вошли во множество $\overline{x_i}$, и те из вершин $\overline{x_i}$ в которые входят дуги из вершины $X_1 \setminus \overline{x_i}$ (множества X').

Характеристическая функция для элементов $x' \in X'$ равна нулю. Операция разности является обратной к операции соединения, т. е. выполняется равенство:

$$S - P_{r_{xi}}(S) + P_{r_{xi}}(S) = S. \qquad (18)$$

Проверим принадлежность S, определенной в (15), к классу программных структур. Если граф G, связный и ациклический, то этими же свойствами будет обладать граф G_1. Корневая вершина G совпадает с корневой вершиной G_1. Таким образом, S удовлетворяет условиям определения программной структуры, приведенным в п.1.

Пусть S^* - множество программных структур, заданное прямым произведением $G^* \times \chi^*$, где G^*– множество графов и χ^* множество характеристических функций.

Обозначим через $\Omega = \{U, \cap, /, +, -\}$ - множество математических операций над программными структурами и P, C и R - предикаты:

$\Omega = \{U, \cap, /, +, -, P, C, R\}$. (19)

Таким образом, определена алгебраическая система $\Sigma = (S^*, \Omega)$ над множеством программных структур и операций над ними (объединение, соединение, проекции и разности).

1.3.3. Характеристика простых и сложных графовых структур

Среди многообразия программных структур выделяются три основные – простая, сложная структура с динамическим вызовом модульных элементов из внешней среды и динамическая структура. Основное назначение различных структур – наиболее оптимальное использование основной памяти во время выполнения агрегата.

Простая структура. Агрегат с простой структурой создается в процессе сборки модулей на основе операций link вызовов. Объем основной памяти, занимаемой агрегатом с простой структурой, постоянен и равен сумме объемов отдельных модулей: $V_s = \sum_{i=1}^{n} v_i$,

где v_i – объем памяти, занимаемый i-м модулем. Соответствующий граф модульной структуры всегда связанный

Сложная структура. Агрегат сложной структуры с динамическим вызовом модулей в общую память создается в процессе сборки модулей. В таком агрегате связи между модулями не такие жесткие и их последовательность определяется входящими в цепочку модулей. Модули в процессе выполнения загружаются в основную память в момент обращения к ним. После завершения работы память освобождается и используется для загрузки другого модуля.

Как и для случая простой структуры, граф сложной программной структуры также связный (рис.4) и отражается в матрице смежности (2).

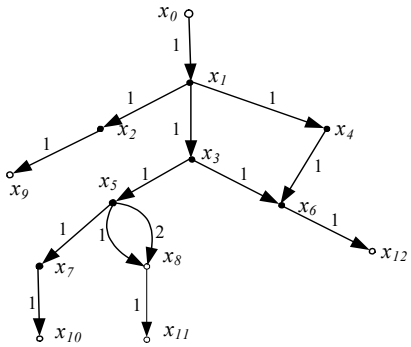


Рис. 4. Граф модифицированной модульной структуры

Объем требуемой основной памяти зависит от количества и состава модулей и максимальный объем памяти равен сумме отдельных модулей: $V_0^{\max} = V_s = \sum_{i=1}^{n} v_i$.

Минимальный объем памяти, требуемый при выполнении агрегата вычисляется по алгоритму Флойда, определяющим кратчайший путь в графе, в котором каждой дуге соответствует весовой коэффициент, называемый длиной дуги.

Для применения алгоритма Флойда выполняются следующие преобразования.

1). Дополним граф новыми вершинами и дугами. Вершинами являются $x_0, x_{n+1}, \ldots, x_{n+m}$, где m-количество концевых вершин. К новым дугам относятся $(x_0, x_1, 1), (x_{r1}, x_{n+1}, 1), \ldots, (x_{rn}, x_{n+rn}, 1)$. В них x_1 соответствует главному модулю, а все x_i – концевым вершинам. После выполнения операций граф модульной структуры (рис. 1) приводится к графу рис. 5 с вершинами $x_0, x_9, x_{10}, x_{11}, x_{12}$. В нем вершинам соответствуют весовые коэффициенты: $v_0 = v_9 = v_{10} = v_{11} = v_{12} = 0$.

2). Каждой дуге вида (x_i, x_j, k) ставится в соответствие коэффициент $v_{ij} = \dfrac{v_i + v_j}{2}$.

Рассмотрим все маршруты, ведущие от x_0 к одной из остальных дополнительных вершин. Длина кратчайшего пути маршрута вычисляется так:

$$l_{0,n+p} = v_{01} + \ldots + v_{rp,n+p} = \frac{v_0 + v_1}{2} + \ldots + \frac{v_{2p} + v_{n+p}}{2} = \frac{v_0}{2} + v_1 + \ldots + v_{rp} + \frac{v_{n+p}}{2} = v_{1+\ldots+} v_{rp}.$$

Эта длина $l_{0,\,n+p}$ будет равна сумме объемов памяти модулей для пути $x_1, \ldots, x_{rp}$.

Таким образом, применяя алгоритм Флойда к графу на рис. 2, мы решаем задачу вычисления объема памяти для максимальной цепочки.

3). Матрицу смежности заменим матрицей путей. Для каждого $m_{ij} > 0$ на соответствующем месте будет находиться значение v_{ij}. Значения $m_{ij} = Ø$ заменяются на $-\infty$. Программа, реализующая алгоритм Флойда, имеет следующий вид (предполагается, что матрица путей описана как двухмерная матрица $n \times n$):

```
for k = 1 to n do
for i = 1 to n do
for j = 1 to n do
if M[i, j ] < M [ i , k] + M[k, j] then
M [i, j]: = M [ i , k] + M[k, j].
```

В результате работы этого алгоритма будет построена матрица максимальных путей. Максимальное из значений $l_{0,n+p}$ будет определять минимальный объем памяти V_0^{min} для агрегата с перекрытием памяти. Самую сложную структуру для значений $V_0^{min} \le V_0 \le V_0^{max}$ можно построить, следуя алгоритмам, предложенным в.

Качественная зависимость V_0 от числа динамических участков представлена на рис.5.

Здесь n – число модулей в агрегате. Несмотря на различный вид кривых, они имеют общую закономерность – любое V_0 заключено между значениями v_0^{max} и v_0^{min}.

Динамическая структура. Механизм динамических связей между модулями отличен от механизма вызова CALL. Загрузка в основную память динамических объектов происходит при обращении к ним. По аналогии назовем объем, загружаемый при однократном обращении динамического элемента, имеет свою программную структуру, для которой составляется матрица смежности. Если в разных динамических структурах встречаются одинаковые модули, то они являются разными объектами. Для иллюстрации используется исходный граф (рис.1). Пусть из модуля, соответствующего вершине x_1, динамически вызывается модуль, соответствующий вершине x_3. Полученный измененный граф приведен на рис. 6. Пунктирная стрелка обозначает динамический вызов. Модуль, соответствующий вершине x_6, встречается дважды.

Построим матрицу смежности для данного агрегата. Каждому динамическому элементу будет соответствовать своя клетка. Чтобы отличить динамический вызов CALL, соответствующие элементы матрицы будут содержать отрицательные числа, абсолютные значения которых задают количество динамических вызовов между данными пары модулей.

Матрица смежности будет иметь вид:

$$M=\begin{pmatrix} x_1 & x_2 & x_4 & x_6 & x_3 & x_5 & x_6 & x_7 & x_8 \\ 0 & 1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (20)$$

Исследуем качественную зависимость объема оперативной памяти от количества динамических сегментов (рис.5. и 6). При одном компоненте в программном агрегате простой структуры имеем $V^1_d = V_s$. Если каждый динамический компонент состоит из одного модуля, то по модифицированному алгоритму Флойда находится максимальный путь и $V_d^n = V_0^{min}$.

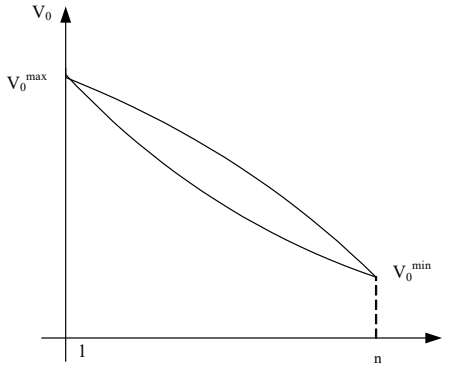


Рис.6. Граф модульной структуры с динамическим вызовом

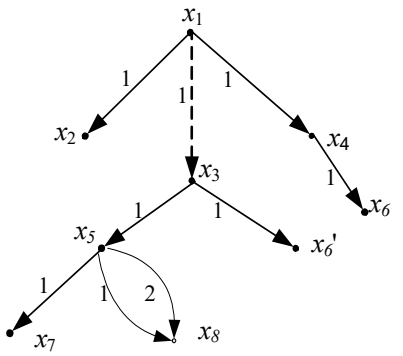


Рис. 7. Графики качественной зависимости V_a от количества подграфов

Для промежуточных значений зависимость имеет более сложный характер.

На рис. 8 представлены две кривые и n – число модулей в программном агрегате.

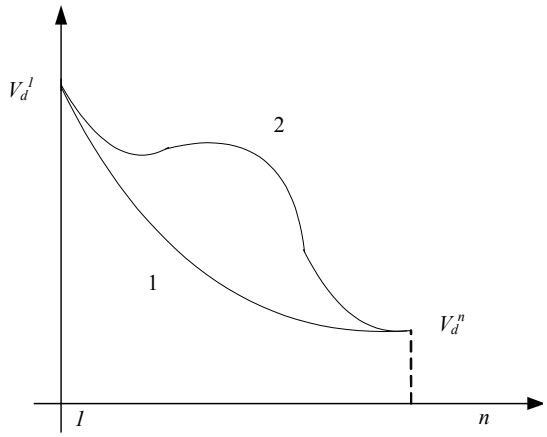


Рис.8. Графики зависимости V_a от количества динамических элементов

За счет дублирования модулей происходит увеличение объема основной памяти ОС.

Таким образом, кривая *1* характерна для программных агрегатов графов в виде дерева, что гарантирует, что в графе нет одинаковых модулей.

Несмотря на недостаток динамической структуры в плане экономии памяти, есть существенное преимущество – независимость от редактирования связей. Каждый динамический объект может быть изменен, а редактирование связей в ОС не требуется.

Кривая *1* задает зависимость, при которой в разных сегментах нет одинаковых модулей. Кривая *2* описывает зависимость для случая, когда у разных сегментов имеются одинаковые модули. Требуемая для них память увеличивается за счет дублирования таких модулей. Однако зависимость 2 характерна для случая, когда в динамических структурах нет одинаковых модулей и они написаны в ЯП высокого уровня. Эти модули обрабатываются утилитами – управление памятью, ввод-вывод, обработка аварийных ситуаций и т.д.

1.3.4. Операции связывания (сборки) модульных структур

Модульная структура, представленная графом G и определенная на множестве модулей $Y – \{y_1, y_2, ..., y_m\}$, описана практически в ЯП (ПЛ/1, Фортран, Кобол, Ассемблер для ОС ЕС) и была основой для проведения сборки модулей в программный агрегат. При этом каждая пара модулей y_i, y_j $(i, j$ – языки из множества $L)$ связаны отношением вызова CALL на основе которого формируется модуль связи y'_{ij}. В общем случае для простых структур программ агрегат, который содержит операторы связи (вызова) Link и операции прямого и обратного преобразований типов данных, передаваемых от вызывающего модуля (в i-языке) вызываемому модулю (в j-языке) и обратно.

Язык ЯП позволяет описать паспорта модулей в классе указанных языков. В паспорте модуля указываются имена вызываемых модулей. Это позволяет по паспорту главного (корневого) модуля построить программную структуру агрегата (типа Prog, комплекс Comp, пакет - Pac). Явное описание модульной структуры связано с представлением графа в виде матрицы смежности, особенности представления которого приведено выше.

Оператор связи модулей Link (make, config, assembling, waver, config и др. после 1994) имеет вид:

LINK <тип агрегата> <имя агрегата> (<имя главного модуля>, <дополнительный список имен модулей>) <режим выполнения>.

При построении конкретных программных структур, вершины графа – модули помечаются специальными символами ρ, обозначающим:

ρ = ¤ – формирование некоторого сегмента, начиная с имени модуля, который этот символ помечает;

ρ = * – начало динамического сегмента с вершины, помеченной в графе этим символом;

ρ = + – отмечает в графе G модуль, который должен быть сформулирован как главный модуль подзадачи;

ρ = / – означает включение отладочной среды в агрегат.

Используя эти обозначения, граф, ранее приведенный на рис.1, примет новый вид, изображенный на рис. 9. В нем для фрагмента, заданного графом $E = \{(x_5, x_7, 1), (x_5, x_8, 1), (x_5, x_8, 2)\}$ и помеченного именем x_5 со знаком ¤, в модуле связи x'_{58} будет сформирован фрагмент из операторов, обеспечивающих вызов с перекрытием памяти. Программа и комплекс задаются уникальным именем, соответствующим генерируемому корневому модулю.

Для графа E = $\{(x_4, x_6, 1)\}$ в модуле связи x'_{46} будет сформирован фрагмент из операторов с динамическим вызовом.

Пример. Для пары модулей, заданных на графе (рис.9) вершинами x_4, x_6 для части агрегата с модулем связи изображен на рис. 10. Аналогично реализуются и другие link графа на рис.9.

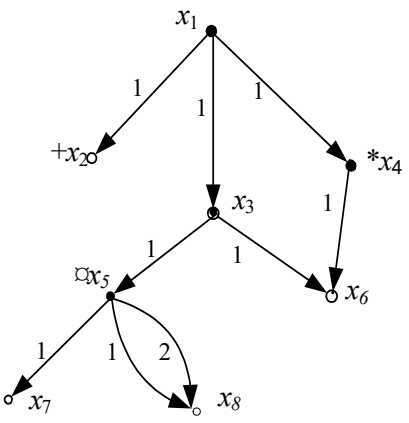


Рис.9. Граф программного агрегата с управляющими отметками на графе

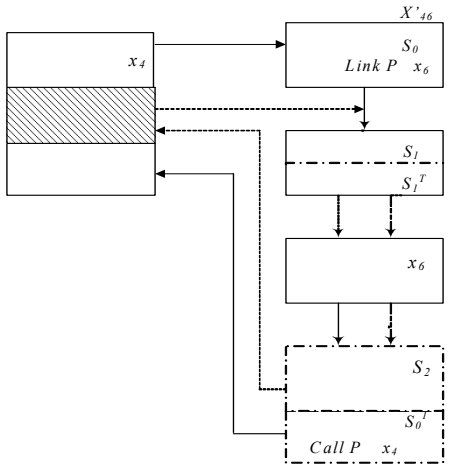


Рис.10. Граф модульной структуры с динамическим вызовом

Таким образом, для пары модулей x_i, x_j создается модуль связи x_{ij} вида:

$$x'_{ij} = S_0\ (S_1 \times S_1^T)\ (S_2 \times S_2^T)\ S_0^I,$$

где S_0 – фрагмент агрегата, задающий среду функционирования модуля x_j;

S_1 – фрагмент агрегата, включающий последовательность обращений к функциям из множества {P, C, S}, каждая из которых осуществляет необходимое преобразование фактических параметров при обращении на x_j -модуль;

S_1^T, $S_2^T \neq \emptyset$ означает наличие средств создания отладочной среды (по ρ=1 в графе модульной программы) для вершин x_j и x_i соответственно;

S_2 – система с фрагментом операторов по обратному преобразованию типов данных, передаваемых из x_j в x_i после его выполнения;

S_0^I – фрагмент программной структуры с операторами эпилога для вершины x_i, обеспечивающей восстановление среды.

Унификация элементов типа модульный элемент (объект) состоит в задании описаний паспортов модулей, как специального раздела модуля, содержащего следующую информацию:

– имя паспорта, которое совпадает с именем модуля;

– ЯП спецификация модуля и список формальных параметров;

– описание типов данных параметров, указанных в паспорте и др.

Паспорт описывается на специальном *языке сборки* link, содержащем:

– подмножество проекций языков из *L,* соответствующее описанию типов данных параметров из списка фактических и формальных параметров;

– математические операции над графом и связи модулей в сложную структуру.

Процессы построения программных графовых структур:

1. Ввод описания модуля в языке программирования (ЯП) *L′* и проведение синтаксического контроля.
2. Выбор необходимых модулей и интерфейсов из хранилищ и их размещения в графе.
3. Трансляция модулей агрегата в ЯП.
4. Генерация модулей связи для каждой взаимосвязанной пары модулей графа и их отладка.
5. Сборка элементов графа в готовую структуру, редактирование связей модулей в среде ОС (IBM, MS, Oberon, Unix и др.).
6. Тестирование системы на наборах данных и оценка показателей надежности агрегата.

Имя агрегата после сборки заносится в загрузочную библиотеку. Если создается некоторый фрагмент, в дальнейшем включаемый в более сложный агрегат, то его имя должно совпадать с именем главного модуля.

В связи с переходом в среду Интернет и проведения конфигурационной сборки предусматриваются средства обеспечения безопасности, защиты данных и оценки качества. С учетом возрастающего количества пользователей сети Интернет и большим объемов данных (Big Data) усовершенствуется клиент–серверная архитектура, которая должна оптимизировать нагрузку на программные ресурсы и данные.

1.3.5. Подход к графовому моделированию сложных структур ОКМ

Начиная с 2000-х годов, после широкого распространения ООП и языка моделирования UML программных и технических систем повсеместно велись разработки сложных систем из объектов. Одновременно проводились работы по изменению ранее изготовленных Legacy-систем (VAMOS, 2004-2018, Reusesbility, 1987-2018 и др.) и вновь создаваемых. В рамках этих направлений в технологии программирования использовалась графовая теория модульных программных структур, которая после 2002 года начала развиваться для объектных и компонентных структур прикладных систем – ОКМ. Граф прикладной системы определяются с помощью математического логико-алгебраического аппарата на четырех уровнях моделирования (рис.11).

Рис.11. Уровни моделирования графа объектов

Остановимся на краткой характеристике уровней моделирования графовой структуры.

I. **На обобщающем уровне** определяется граф G_{t1} из базовых понятий – объектов-функций, каждый их которых задается в соответствии с теорией Фреге (рис.12).

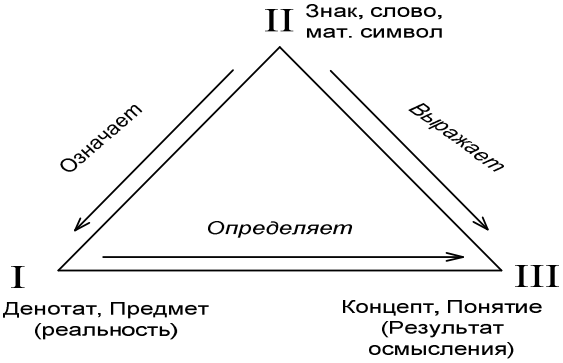


Рис. 12. Треугольник Фреге для определения объекта

Объект, исходя из треугольника Фреге, - это <символ, денотат, концепт>, в котором денотат задает смысл/ семантику объекта. Одному объекту могут соответствовать несколько концептов ПрО. Для выявленных объектов проводится их конкретизация с помощью математических операций (объединения, пересечения, разности, симметричной разницы и др.) создания классов объектов (объединенного класса с помощью операции прямой суммы, класса пересечения объектов множества, агрегированного класса, как подмножество декартова произведения нескольких других множеств и др.).

Каждый объект принадлежит некоторому множеству $O= (O_0, O_1 ,..., O_n)$, где $O_i = O_i (Name_i, Den_i, Con_i)$, а $Name_i$, Den_i, Con_i соответственно задают – символ (имя), денотат и концепт. Элементы множеств имеют отношение связи через интерфейсный объект.

II. **На структурном уровне** определяется расположение объектов в графе G_{t2} объектной модели (ОМ), в вершинах которого располагаются объекты, а дуги задают связь объектов друг с другом. Множество объектов определяется как графическая структура из объектов-функций и предикатов определенной сигнатуры с помощью свойств и характеристик (0-арной, унарной и бинарной операции) для задания связей между парами объектов.

Объект, имеющий одновременно статус множества (класса) и элемента какого-либо множества, имеет внешние, внутренние свойства и характеристики. Объекты обобщающего уровня, заданные множеством

$$O=(O_0, O_1, O_2 . .., O_n), O'=(O_1, O_2, , ..., O_n) \quad (19)$$

трансформируются к виду:

$$\forall i \exists j [(i>0) \& (j>0) \& (i \neq j) \& (O_i \in O_j)]. \quad (20)$$

Результатом моделирования является граф G_{t3} (рис.3) с отношениями и свойствами:

множество вершин графа задает взаимно однозначное отображение объектов;

для каждой вершины существует хотя бы одна связь с другой вершиной графа ($\rightarrow$);

существует лишь одна вершина O_1 графа G_{t3} (рис.13), которая имеет статус множества объектов, отображающих систему в целом.

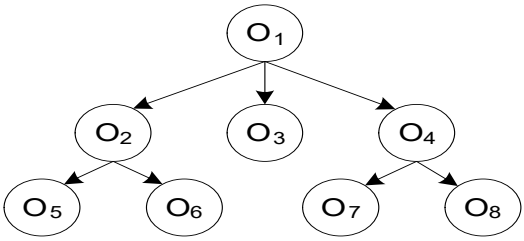


Рис.13. Структура графа G ={O}

III. **На характеристическом уровне** граф G_{t3} уточняет связи и отношения, задаваемые интерфейсом, содержащим вызовы объектов с передачей данных от одного объекта к другому. На этом уровне для объектов (1.1) задается множество предикатов $P'=(P_1, P_2, \ldots P_r)$ помощью *0-арных, унарных* и *бинарных* операций, принимающих значение истины: $Con_i = \{O_{ik}\}$ і $P_k(O_i) = true$. Между объектами устанавливаются отношения типа «род-вид".

Аксиома 4.1. Каждый объект имеет хотя бы одну характеристику, которая задает семантику и уникальную идентификацию во множестве объектов системы.

Характеристики задают множество свойств, которое определяется типом объекта и принадлежностью только одному объекту модели ОМ. Характеристика может быть внешней и внутренней. Любой объект множества или его элемент может иметь только одну внешнюю и внутреннюю характеристику.

На этом уровне уточняется отношение связь-интерфейс, в котором передаются внешние характеристики другим объектам, связанным на графе отношением принадлежности. К ранее построенному графу *G* добавляются интерфейсы (рис.13):

$$G = \{O, I, R\}, \qquad (20)$$

где *O* – множество объектов-функций, *I* – множество интерфейсов, *R* – множество отношений (relations) между объектами (рис. 14). В граф G входят интерфейсные объекты $I = (O'_5, O'_6, O'_7, O'_8)$, которые выполняют вызов объектов и передачу им данных в требуемом формате.

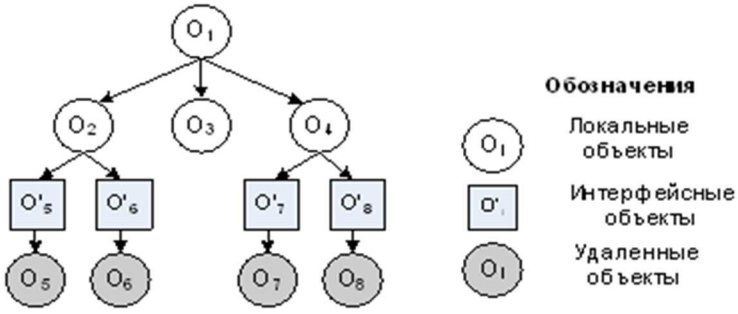


Рис. 14. Граф G на множестве объектов и их интерфейсов

Вершины данного графа *G* залают объекты – $O_1, O_2, O_3, O_4, O_5, O_6, O_7, O_8$ и интерфейсы – $O'_{25}, O'_{26}, O'_{47}, O'_{48}$, которые размещаются в репозитории системы, а дуги соответствуют отношениям между объектами и интерфейсами.

Элементы графа $O_1 – O_8$ описываются в ЯП, а интерфейсы $O'_{25}, O'_{26}, O'_{47}, O'_{48}$ в специальном языке интерфейса IDL (Interface Definition Language). Параметры внешних характеристик передаются между объектами через интерфейсы и помечаются как *in* (входной), *out* (выходной), *inout* (входной и выходной).

Интерфейсы графа содержат описание передаваемых данных, операторы удаленных вызовов RPC/RMI/Call с передачей данных. При необходимости передаваемые данные

преобразуются (например, брокером ORB) к соответствующим форматам среды выполнения функции объекта.

По графу G (рис.14) можно построить программы $P_1 - P_5$ с помощью $\cup$ операции объединения (сборки) *link*:

$P_1 = O_2 \cup O_5$, *link* $P_1 = In\ O'_{25}\ (O_2 \cup O_5)$;

$P_2 = O_2 \cup O_6$, *link* $P_2 = In\ O'_{26}\ (O_2 \cup O_8)$;

P_3 ;

$P_4 = O_4 \cup O_7$, *link* $P_4 = In\ O'_{47}\ (O_4 \cup O_7)$;

$P_5 = O_4 \cup O_8$, *link* $P_4 = In\ O'_{48}\ (O_4 \cup O_8)$;

$P_0 = (P_1 \cup P_2 \cup P_3 \cup P_4 \cup P_5)$.

Результат связи двух объектов (например, *link* P_2 содержит интерфейс $In\ O'_{26}$). В нем задается список входных интерфейсов, совпадающих с множеством интерфейсов объекта–приемника, а также список выходных интерфейсов, которое совпадает с множеством выходных интерфейсов объекта–передатчика. Программы P_1 - P_5 образуют сложную систему P_0. Объекты каждой программы $P_{1\text{-}5}$ транслируются в язык компьютера. При этом проверяется правильность формального описания объектов и задания их интерфейсов. Каждая программ $P_{1\text{-}5}$ после трансляции тестируется на правильность выполнения на соответствующих данных.

Окончательная программная система P_0, заданная математической операцией *объединения* отдельных программ $P_{1\text{-}5}$ в структуру $P_0 = (P_1 \cup P_2 \cup P_3 \cup P_4 \cup P_5)$ является результирующей и сохраняется в библиотеке или в репозитории операционной среды.

Для передачи заказчику требуется провести тестирование каждой программы из $P_{1\text{-}5}$ на множестве наборов тестов и данных. При возникновении ошибок должны исправляться те объекты, в которых найдены ошибки, дефекты или отказы. Потом проводится, согласно требований стандартов Guality (ISO/IEC 9126, 9000, 1-4), комплексное тестирование собранной методом конфигурационной сборки (стандарта IEEE 828-2012) с измерением показателей качества (надежность, функциональность и др.) и выдачи сертификата на продукт.

Аксиома 4. 2. Расширенный граф *G* с объектов и интерфейсов, структурно упорядочен (наверх), проконтролирован на полноту, избыточность и дублирующие элементы.

Объекты могут иметь несколько интерфейсов, которые могут наследовать интерфейсы других объектов. Множество объектов и интерфейсов графа *G* отмечаются общими характеристиками объектов в ОМ. Они считаются достоверными, если выполняется условие: каждая внутренняя характеристика эквивалентна внешней характеристике объекта. Если это условие не выполняется, то такой элемент удаляется из множества *O* и из графа соответственно.

IV. **На поведенческом уровне** в графе объектов определяется их поведение в зависимости от событий, которые они могут создать при выполнении в конкретной среде.

Входными данными для этого уровня являются данные модели ОМ на предыдущих уровнях. На данном уровне определяется последовательность действий по изменению состояний объектов и перехода состояний с учетом содержащихся и сформированных бинарных предикатов, связанных свойствами с объектами системы, и их состояниями.

Контроль графа объектов осуществляется логико-алгебраическими операциями. Они выявляют ситуации нарушения структурной упорядоченности объектов и обеспечивают группирование объектов с помощью бинарных отношений типа *'is-a'* и для пары "объект-множество – объект-элемент" выполняются операций: э*кземпляризации* и *классификации.* объектов по их внутренним и внешним свойствам.

Паре "объект-множество – объект-множество" соответствуют операции: *обобщение* и *специализация*. Для реализации бинарного отношения *'Part–of'* пары "объект-элемент – объект-элемент" используются операции:

– *агрегации* объектов по их внешним характеристикам;

– *детализации* (операция обратная операции агрегации).

Операции позволяют определить объекты графа и правильность выполнения операций образования отдельных программ при создании сложной программной структуры.

1.3.6. Теория преобразования данных в методе сборки

Метод сборки разнородных модулей в новые программные структуры включает в себя математические формализмы определения связей (по данным и по управлению) между объектами сборки и генерации интерфейсных модулей для каждой пары соединяемых модулей. Связь модулей задается оператором Link или CALL, в котором задаются формальные и фактические параметры.

Сущность метода сборки (объединения) пары модулей в разных ЯП состоит в определении взаимно однозначного соответствия между множеством фактических параметров $V = \{v_1, v_2, ..., v_к\}$ вызывающего модуля и соответствующим множеством формальных параметров $F = \{f_1, f_2, ..., f_к\}$ вызываемого модуля, а также в отображении типов данных одних параметров в другие. Если отображение не удается выполнить, то задача автоматизированной связи данной пары модулей считается неразрешимой.

Для простых и сложных типов данных модулей в классе фундаментальных типов данных (ФДТ) построены алгебраические системы.

Для каждого типа данных ФДТ алгебраические системы содержат множество значений и операций над ними. Каждая операция устанавливает соответствие передаваемых данных. Если они одинакового типа, то производится обмен. Если передаваемые данные не совпадают (по типу, количеству, формату), то требуется произвести их преобразование (туда и обратно). Каждому преобразованию соответствует некоторая функция, осуществляющая изоморфного отображения одного типа к другому в рамках алгебраической системы.

Например, преобразование между типами данных массив и запись сводятся к изоморфному отображению множества значений с помощью операций изменения уровня структурирования данных – селектора и конструирования. Для массива операция селектора сводится к отображению множества индексов на множество значений элементов массива.

Формально преобразование типов данных в ЯП состоит в построении и реализации:

1. Набора операций преобразования типов данных $T = \{T_\alpha^t\}$ для множества ЯП $L = \{l_\alpha\}_{\alpha=1, n}$.

2. Функций отображения простых типов данных для каждой пары взаимодействующих модулей в $l_{\alpha1}$, $l_{\alpha2}$ и применения операций селектора S и конструктора C для отображения сложных структур данных между собой и сложных типов данных в более простые ЯП.

Формальное преобразование типов данных с помощью алгебраических систем для каждого типа данных задается в виде; T_α^t: $G_\alpha^t = < X_\alpha^t, \Omega_\alpha^t >$, где t – тип данных ($t = b, c, i, r, a, z, u, e$); X_α^t – множество значений, которые могут принимать переменные; Ω_α^t – множество операций над типами данных t.

Простым и сложным типам данных современных ЯП ставятся в соответствие алгебраические системы: $\Sigma_1 = \{G_\alpha^b, G_\alpha^c, G_\alpha^i, G_\alpha^r\}$, $\Sigma_2 = \{G_\alpha^a, G_\alpha^z, G_\alpha^u, G_\alpha^e\}$. (21)

где Σ_1 – соответствует классу простых типов данных, Σ_2 - сложным типам данных.

Каждый элемент класса простых и сложных типов данных определяется на множестве их значений и операций над ними: $G_\alpha^t = < X_\alpha^t, \Omega_\alpha^t >$, l_α - язык, t - тип данных; Ω_α^t – множество операций над типами данных t.

1.3.7. Операции изоморфного отображения передаваемых данных

Преобразования каждого t типа данных двух алгебраических систем проводится для совместимых по типу данных в двух разных ЯП.

В классе систем Σ_1 и Σ_2 преобразование типов данных $t \to q$ для пары языков l_t и l_q обладает такими свойствами отображений:

1) $G_\alpha{}^t$ и $G_\beta{}^q$ – изоморфны (q определено на том множестве, что и тип t);

2) между $X_\alpha{}^t$ и $X_\beta{}^q$ существует изоморфизм, для которых множества $\Omega_\alpha{}^t$ и $\Omega_\beta{}^q$ разные, если набор операций $\Omega = \Omega_\alpha{}^t \cap \Omega_\beta{}^q$ не пуст, то рассматриваем изоморфизм между

$G_\alpha{}^{t'} = <X_\alpha{}^t, \Omega>$ и $G_\beta{}^{q'} = <X_\beta{}^q, \Omega>$. И такое преобразование сводится к случаю 1).

3) между множествами $X_\alpha{}^t$ и $X_\beta{}^q$ может не существовать изоморфного соответствия. Тогда требуется построить такое отображение между $X_\alpha{}^t$ и $X_\beta{}^q$, чтобы оно было изоморфным. Если такое отображение существует (в каждом конкретном случае оно может быть разным), то имеем условие случая 1) с соответствующими изменениями в определении систем (21);

4) мощности алгебраических систем должны быть равны $|G_\alpha{}^t| = |G_\beta{}^q|$.

Любое отображение 1), 2), 3) сохраняет линейный порядок, так как алгебраические системы линейно упорядочены.

Лемма 1. Для любого изоморфного отображения φ между алгебраическими системами $G_\alpha{}^t$ и $G_\beta{}^q$ выполняются равенства $\varphi(X_\alpha{}^t\,.\,_{\min}) = = X_\beta{}^q\,.\,_{\min}$, $\varphi\,(X_\alpha{}^t\,.\,_{\max}) = X_\beta{}^q{}_{\max}$.

Доказательство леммы тривиальное и простое. При условии, когда одно или два вышеприведенных равенства не выполняются, тогда для основных множеств алгебраических систем изменяется линейный порядок, что противоречит определению.

Формальные условия преобразования типов данных определяются теоремами 1–5.

Теорема 1. Пусть φ – отображение алгебраической системы $G_\alpha{}^c$ в систему $G_\beta{}^c$. Для того чтобы φ было изоморфизмом, необходимо и достаточно, чтобы φ изоморфно отображало $X_\alpha{}^c$ на $X_\beta{}^c$ с сохранением линейного порядка.

Необходимость. Пусть φ – изоморфизм. Тогда при отображении сохраняются все операции множества $\Omega = \Omega_\alpha{}^c = \Omega_\beta{}^c$, в том числе и операция отношения, которая определяет линейный порядок $X_\alpha{}^c$ и $X_\beta{}^c$.

Достаточность. Пусть φ изоморфно отображает $X_\alpha{}^c$ на $X_\beta{}^c$ с сохранением линейного порядка. Операция отношения выполняется соответственно принципу упорядоченности. Операцию succ докажем с помощью леммы, согласно которой выполняется равенство $\varphi\,(X_\alpha{}^c\,.\,_{\min}) = X_\beta{}^c{}_{.\min}$.

Последовательно применяя операцию succ к этому равенству и учитывая линейную упорядоченность $X_\alpha{}^c$ и $X_\beta{}^c$ ($x < \mathrm{succ}(x)$), получаем, что для любого $x_\alpha{}^c \in X_\alpha{}^c$ и $x_\alpha{}^c \neq X_\alpha{}^c{}_{\min}$ из равенства $\varphi\,(X_\alpha{}^c) = x_\beta{}^c$, где $x_\beta{}^c \in X_\beta{}^c$, выполняется равенство

$$\varphi(\mathrm{succ}\ (x_\alpha^c)) \ = \ \mathrm{succ}(x_\beta^c). \qquad (22)$$

Операция pred доказывается аналогично с помощью $\varphi\,(X_\alpha{}^c{}_{\max}) = X_\beta{}^c{}_{\max}$.

Теорема 2. Любой изоморфизм φ между алгебраическими системами $G_\alpha{}^b$ и $G_\beta{}^b$ является тождественным изоморфизмом:

$$\varphi(X_{\alpha.\ \mathrm{false}}{}^b) = X_{\beta.\ \mathrm{false}}{}^b,$$
$$\varphi(X_{\alpha.\ \mathrm{true}}{}^b) = X_{\beta.\ \mathrm{true}}{}^b. \qquad (23)$$

Доказательство. При отображении $G_\alpha{}^b$ и $G_\beta{}^b$ всегда справедливо $X_{\alpha.\ \mathrm{false}}{}^b < X_{\beta.\ \mathrm{true}}{}^b$. Поэтому для сохранения линейного порядка единственно возможным изоморфизмом является (23).

Теорема 3. Любой изоморфизм между алгебраическими системами с соответствующими числовыми типами является тождественным автоморфизмом.

Доказательство этой теоремы тривиальное и является следствием свойств элементов числовых множеств.

Теорема 4. Пусть G_α^a и G_β^a – алгебраические системы, которые отвечают типам данных массива (a); φ_i и φ_v – изоморфные отображения множеств индексов (i) и значений элементов (Y) массивов, которые сохраняют линейный порядок. Тогда изоморфизм φ между алгебраическими системами целиком определяется изоморфными отображениями:

$$\begin{aligned} &\varphi_i : X_\alpha^a \to X_\beta^a, \\ &\varphi_v : Y(X_\alpha^a) \to Y(X_\beta^a). \end{aligned} \qquad (24)$$

Изоморфизм φ между алгебраическими системами G_α^a и G_β^a определяется отображениями φ_i и φ_v, которые сохраняют линейный порядок и упорядоченность элементов массива.

Теорема 5. Пусть G_α^z и G_β^z – две алгебраические системы, которые отвечают типам данных «запись» и $x_\alpha^z \in X_\alpha^z$, $x_\beta^z \in X_\beta^z$.

Тогда если между последовательностями компонентов записей x_α^z и x_β^z существует взаимно однозначное соответствие, то изоморфизм φ между G_α^z и G_β^z определяется изоморфными отображениями алгебраических систем, которым соответствуют компоненты записи или структуры.

Преобразования между массивами и записями сводятся к преобразованию простых типов данных (ТД) их элементов. Преобразования между действительными типами и другими числовыми значениями предполагают использование эмпирических случаев, так как отсутствует изоморфизм основных множеств этих алгебраических систем.

Операция конструирования *C* массива состоит в формальном приведении в порядок компонентов и определении соответствия между множеством индексов и множеством элементов массива. Аналогично эта операция определяется для записи.

Проблема связи разноязыковых программ обострилась в связи с быстрым изменением архитектуры компьютеров, появлением распределенных, клиент-серверных сред и т.п. Проявилась неоднородность ЯП, как в смысле представления в них ТД, так и платформ компьютеров, на которых реализованы соответствующие системы программирования с ЯП, а также различные способы передачи параметров между объектами в разных средах через операции link, make, config, building, assembling.

1.3.8. Трансформация данных с учетом стандарта ISO / IEC 11404 GDT - 2012

В стандарте GDT 1996 задан подход к решению вопросов интерфейса для всех ЯП с помощью универсального языка LI (Language Independent) для новых типов данных. В этом стандарте определены следующие ТД:

- примитивные ТД (real, integer, char, boolean) и другие ТД,
- сложные ТД (массив, запись, последовательность, портфель, каркас...),
- сгенерированные ТД, которые получаются после генерации нового ТД.

В стандарт 2007 вошли сложные типы данных (например, агрегатные), которые требуют их генерации к фундаментальным ТД и создания новых примитивных функций преобразования неэквивалентных ТД для современных ЯП (C, C++, Python, Ruby, Basic, Java и др.), используемых на современных компьютерных и суперкомпьютерных платформах.

Любые данные, которые специфицированы в готовых ресурсах Интернет, передаются через параметры взаимодействия, требуют преобразования ТД и форматизации данных для разных платформ компьютеров. Для решения проблем преобразования данных при

взаимодействия готовых программных и сервисных ресурсов и ресурсов, предложен подход к генерации общих типов GDT к фундаментальным ТД (FDT).

При генерация GDT<=>FDT создается библиотека функций (процедур) в языке XML, которые обеспечивают:

– преобразование ТД разных $ЯП_1$, …, $ЯП_n$;

– представление фундаментальных ТД к виду специальных функций;

– преобразование общих ТД GDT к виду фундаментальных FDT;

– эквивалентные отображения GDT<=>FDT.

Теория преобразования ТД GDT обеспечивает:

1) отображение примитивных, агрегатных и генерированных ТД к простым и сложным фундаментальным FDT, ориентированных на связь программ в ЯП (C, C+, C#, Java, Python);

2) сохранение функций преобразования ТД для ЯП в хранилищах данных и репозиториях;

3) генерацию интерфейсных посредников (типа Stub, Skeleton) с помощью примитивных функций библиотек и с учетом платформ компьютерных систем;

4) разработку метода генерации неструктурированных типов данных из класса Big Data.

Вариант теории преобразования ТД GDT с Big Data рассмотрены в литературе.

Таким образом, в этом разделе дано описание нового объектно-компонентного метода (ОКМ) моделирования графа прикладной системы, по которому проводится сборка объектов в разные программы P_1-P_5 и приводится теория изоморфного отображения неэквивалентных типов данных (структурных, неструктурных и генерируемых), передаваемых между отдельными объектами программ P_1-P_5.

Метод сборки больших программ и комплексов из модулей проводился на основе графовой структуры. Теория графов обеспечивает связывание элементов графовых модульных структур с помощью математических операций (объединения, соединения, разности и др.) в сложные программные структуры (комплекс, агрегат, система и др.) и доказательство их достижимости. Эта теория начала внедряться в проектах ВПК под руководством Липаева В.В. и при поддержке академика А.П. Ершова в ИПИ СО АН СССР, ученики которого развивали теорию графов для программирования трансляторов и операционных систем. Начиная с 2003 года, теория графов начала применяться для моделирования сложных систем по методу объектного моделирования из объектов (ОМ) и компонентов с применением операций сборки (assembler, config) согласно стандарта ISO/IEC 828 (1996, 2012, Configuration). На основе графового описания программных структур проводится сборка объектов и компонентов и применением аппарата изоморфного отображения неэквивалентных типов данных (структурных, неструктурных и генерируемых), передаваемых между отдельными объектами создаваемых программ. В рамках проекта РФФИ 19-00206-2019 представлена теория моделирования систем по графу, элементами которого являются ресурсы (объекты, компоненты и сервисы). Этот метод графового моделирования представлен в зарубежных и отечественных журналах в 2019 и перспективе будет использоваться в таких прикладных областях, как медицина, генетика, биология и др.

Список литературы к 1.1. – 1.3.7

1. Лаврищева Е.М., Грищенко В.Н. Связь разноязыковых модулей в ОС ЕС.- Москва, 1982.-127с.

2. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование. –К.: Наук. Думка.1991.-136с.

3. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование Основы индустрии программных продуктов. К.: Наук.Думка.-2009.-371с..

4. Глушков В.М., Стогний А.А., Лаврищева Е.М. и др. Система автоматизации производства программ (АПРОП) .-Киев, 1976.-134с.

5. Липаев В.В., Позин Б.А., Штрик А.А. Технология сборочного программирования.-М.: 1992.-284 с.

6. Римский Г.В. Структура и функционирование системы автоматизации модульного программирования.- Искусственный интеллект: применение в химии..-1987.-№5.-с.36-44.

7. Холстед М.Х. Начало науки о программах.- Пер. с анг. –М.: Финансы и статистика.-1981.-201с.

8. Хорн Э., Винклер Ф. Проектирование модульных структур.– Вычислительная техника социалистических стран.-1987.- Вып .21.-с.64-72.

9. Коваль Г.И., Коротун Т.М., Лаврищева Е.М. Об одном подходе к решению проблемы межмодульного и технологического интерфейса// Межотрасл. сборник АН и Мин.Вуза СССР.-1987.

8. Агафонов В.Н. Спецификация программ: понятийные средства и их организация.-Новосибирск.- Наука, 1987.-380с.

10. Котов В. Е., Введение в теорию схем программ, Новосибирск, 1978.

11. Непейвода Н. Н. Логика программ.- Программирование, 1979, № 1, с.15-25;

12. Евстигнеев А.Н. Теория графов в программировании. Москва, Наука. 1985. 351с.

13. Ершов А.П. Введение в теорию программирования.-Москва.-1977.- 287с.

14. Лаврищева Е.М. Теория объектно-компонентного моделирования программных систем. Препринт ИСП РАН, № 29, 2016 г. - М.: 48 с. ISBN 078-5-91474-025-9.13;

15. Лаврищева Е.М. Рыжов А.Г. Применение теория общих типов данных стандарта ISO/IEC 11404 GDT применительно к Big Data.- The conference “Actual problems in science and ways their development”, 27 December 2016, http://euroasia-science.ru/ p.99-110.

16. Лаврищева Е.М., Митулин В.С., Козин С.В., Рыжов А.Г. Создание прикладных и информационных систем из готовых ресурсов Интернет. Труды ИСП РАН.-М.: Том 29. Вып.3.-с

17. Лаврищева Е.М. , А.Г.Рыжов. Подход к моделированию систем и сайтов из готовых ресурсов.- .XX Всероссийская конференция , 17-22 сентября 2018г. г. Новороссийск.-ИПМ им. М.В.Келдыша.-Презентация доклада. Публикация в сборнике.-с. 321-345

18. И.Б.Бурдонов, А.С.Косачев, В.В. Кулямин «Теория соответствия для систем с блокировками и разрушениями.-Москва, 2008.- 411с.

19. Лаврищева Е.М. Software Engineering компьютерных систем. Парадигмы, технологии, CASE- средства- Наук. Думка.- 2014.-284с.

20. Bruno Courcelle, Joost Engelfriet Graph structure and monadic second-order logic. A language-theoretic approach (hal id: hal-oo646514) and Theory graph (wikipedia.ru, Foxford.ru).

21. Лаврищева Е.М., Пакулин Н.В., Рыжов А.Г., Зеленов С.В. Анализ методов оценки надежности оборудования и систем. Практика применения методов надежности. 5 Научно-практическая конференция - OS DAY, Москва, 17-18мая 2018. Презентация доклад и публикация на анг. языке. Труды ИСП РАН, том 5, DOI: 10.15514/ISPRAS-2018-30(3), 2018. (http://0x1.tv/20180517F).

22. Ekaterina M.Lavrischeva. Assemblling Paradigms of Programming in Software Engineering.- 2016, 9, 2016.- p.296-317, http://www.scrip.org/journal/jsea, http://dx.do.org/10.4236/jsea.96021

23. E. M. Lavrischeva.The Scientific basis of software engineering.- International Journal of Applied And Natural Sciences (IJANS) ISSN (P): 2319-4014; ISSN (E): 2319-4022 Vol. 7, Issue 5, Aug - Sep. 2018; p. 15-32.

24. Городняя Л.В. Парадигмы программирования. Анализ состояния и перспективы.-СОРАН, 2018.-282с.

25. Лаврищева Е.М., Рыжов А.Г. Подход к моделированию систем и сайтов из готовых ресурсов /Научный сервис в сети Интернет: труды XX Всероссийской научной конференции

(17-22 сентября 2018 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2018. — С. 321-345. — Doi:10.20948/abrau-2018-50.

26. Лаврищева Е.М. Программная инженерия и технология программирования сложных систем.- Москва-Юрайт, 2019.- 431 (biblio-online.ru).

2. Технология сборки ПС, ПТС с обеспечением качества

Начиная с 2000-х годов, после широкого распространения ООП и языка моделирования UML программных и технических систем повсеместно велись разработки сложных систем из объектов. Одновременно проводились работы по изменению ранее изготовленных Legacy-систем (VAMOS, 2004-2018, Reusesbility, 1987-2018 и др.) и вновь создаваемых. В рамках этих направлений в технологии программирования использовалась графовая теория модульных программных структур, которая после 2002 года начала развиваться для объектных и компонентных структур прикладных систем – ОКМ. Граф прикладной системы определяются с помощью математического логико-алгебраического аппарата на четырех уровнях моделирования (рис.11).

2.1. Способы сборка систем из модульных элементов через сборшик ресурсов Интернет

В 1975году академик В.М.Глушков высказал идею о том, что «программы будут собираться конвейерным способом, как автомобили из готовых деталей на фабрике Форда». При исследовании этой идеи сформировалось понятие интерфейса связывания программных элементов (по типу болтов и гаек деталей автомобилей) методом сборки, который объединял разнородные модули через интерфейс. Межмодульный интерфейс реализован в 1976–1982 г. в системе АПРОП. Межъязыковый и технологический интерфейс обсуждался на конференции «Интерфейс СЭВ-1987 и стал базовым понятием в технологии программирования. Автор получил грамоту Евроконференции Интерфейс СЭВ за определение понятия интерфейса (рис.1).

Метод сборки и библиотека интерфейса (64 примитивов преобразования неэквивалентных данных) реализованы в ОС ЕС или IBM 360, переданы в 52 организации СССР и внедрены в комплексы РУЗА, ПРОМЕТЕЙ, ЯУЗА (В.В. Липаев, МНИИПА, Минрадиопрома СССР) для специализированных ЭВМ ВПК.

Интерфейс модулей в языке MIL (Module Interface Language), а операция сборки модулей задавалась оператором link (M_1, M_2) модулей. Метод сборки с интерфейсными примитивами преобразования разнородных данных ЯП опубликован в книге. Эта система была внедрена в 52 организации страны. Сформировалось сборочное программирование.

Рис.1. Интерфейс в программировании

Рассмотрим существующие виды сборки 1-5 ресурсов в ОС Интернет с использованием стандарта конфигурационной сборки IEEE 824 –Configuration: 1996.

1). Способ сборки make реализован в системах BSD, GNU, Java, как оператор объединения функций в ЯП из библиотек модулей транслятора Microsoft, UNIX и др., интерфейс которых задавался в API (Applic ation Programming Interface) на уровне исполняемых файлов в машинном коде и в ABI (Application Binary Interface) для объединения скомпилированных модулей в промежуточном MSIL байт-коде (C++, Fortran, Go, Perl, PHP, Java) в среде .NET. Процесс сборки make ресурсов в API и ABI ЯП выполняет (рис.2):

– обработку API и ABI интерфейсов из библиотеки .NET;

– генерацию сборочных скриптов GNU Build system, CMake;

– MSBuild (.NET), Apache Ant (Java), Apache Maven (Java, C#, Scala), NAnt (.NET), Scons(C, C++, Java, Fortran,Tex).

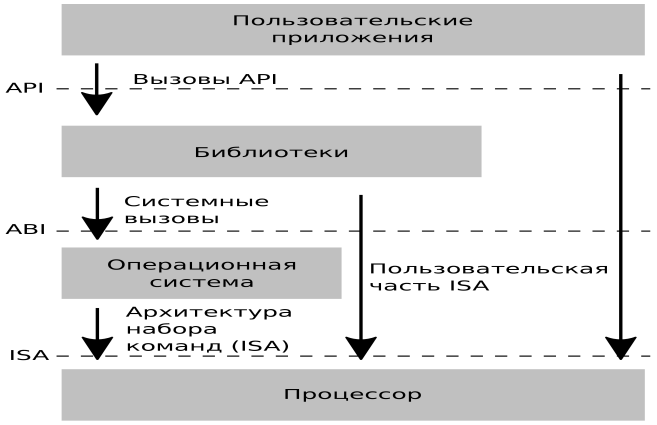


Рис. 2. Схема обработки make

Процесс сборки ресурсов в ЯП с интерфейсом в API и ABI через make выполняет генератор сборочных скриптов GNU Build system в средах MSBuild (.NET), Apache Ant (Java),

Apache Maven трансляторов с Java, C#, Scala; Scons (C, C++, Java, Fortran,Tex) и др. ABI Interface содержит:

– набор инструкций процессора к регистровому файлу, стеку и памяти;

– размер и расположение базовых ТД, с которыми работает процессор компьютера;

– бинарные форматы файлов, библиотек и исполняемых файлов.

Операции связи программ в API и ABI для C++ и Fortran (рис.3):

– add executable (exec_name source1 source2 …);

– создать исполняемый файл из файлов исходного кода source1, source2, и т.д.

main.cpp

```
#include <iostream>

extern "C" void fort(void);
extern "C" void cube(float* x, float* y);

int main() {
  fort();
  std::cout << "C++\n";
  float x = 2.5, y = 0;
  cube(&x, &y);
  std::cout << x << ' ' << y << '\n';
}
```

fort.f90

```
subroutine fort() bind(C)
  implicit none
  write(*,*) "Fortran"
  return
end

subroutine cube(x,y) bind(C)
  implicit none
  real*4 x, y
  y = x * x * x
  return
end
```

Запуск

```
gfortran -c fort.f90 -o fort.o
g++ main.cpp fort.o –lgfortran
./a.out
```

Вывод

```
 Fortran
C++
2.5 15.625
```

Рис.3. Пример сборки модулей в C++ и Fortran

2). Способ сборки через интерфейс Stub-клиент и Skeleton-сервера в IDL (Interface Definition Language) задают описание интерфейсов программ в языках (Java, Cobol, PL/1, Ada-95, C, C++ и др.). Их связь в IDL в разных ЯП объектной модели (ОМ) задает брокер CORBA. В рамках стандарта WWW Web появился язык WSDL (2004) для связывания разнородных программных элементов в ЯП нового поколения (Си, C++, Basic, Java, Cobol, ADA-95, Python и др.) с использованием библиотеки 64 функций в среде Интернет (рис.4)

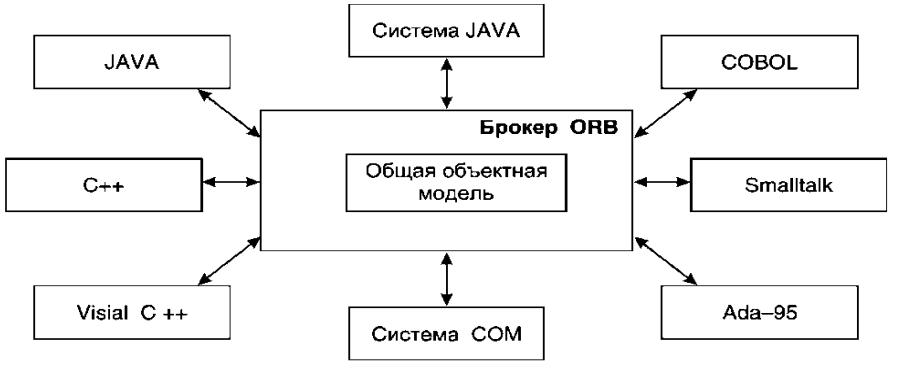


Рис. 4. Функциональная схема брокера ORB

3). Способ сборки в среде .Net основана на обработке передаваемых данных между модулями с помощью системы CTS (Common Type System, рис.5).

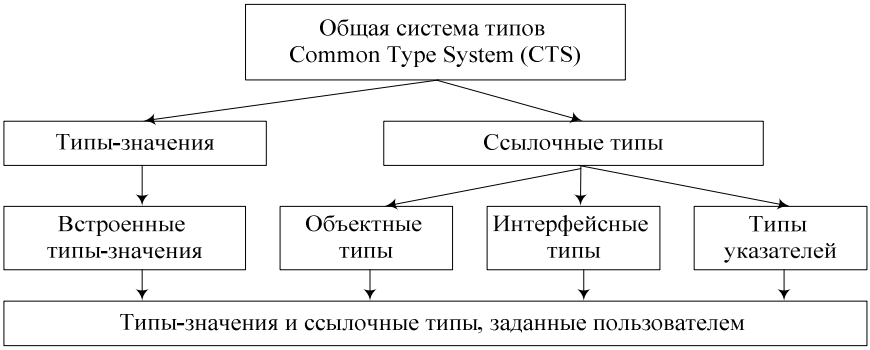


Рис.5. Система обработки общих типов данных в .Net

В задачи сборки входит:

– отображение значений, ссылочных и встроенных ТД в формате данных компьютера;

– описание подпрограмм в языке CLR (Common Language Runtime) и обработка целочисленных данных, с плавающей запятой, строк, битов;

– спецификации классов, интерфейсов, встроенных типов данных, перечислений и др., заданных в CLS (Common Language Specification);

– перевода данных в ЯП к промежуточному MSIL, который преобразуется в код CPU для выполнения на разных архитектурах компьютеров сети.

При сборке объектов на основе интерфейса в языке IDL описание начинается с ключевого слова interface, за которым следует: имя интерфейса, описание типов параметров и операций (op_dcl) вызова объектов:

interface A { ... }

interface B { ... }

interface C: B, A { ... }.

Параметры операций (op_dcl) в задании интерфейсов это:

– тип данных (type_dcl);

– константа (const_dcl);

– название исключительной ситуация (except_dcl), которая может возникнуть в процессе выполнения метода объекта;

– атрибуты параметров (attr_dcl).

Описание типов данных (ТД) начинается ключевым словом typedef, за которым следует базовый или конструируемый тип и его идентификатор. В качестве константы может быть некоторое значение типа данного или выражение, составленное из констант. ТД и константы описываются как фундаментальные типы данных: integer, boolean, string, float, char и др.

Описание операций op_dcl передачи данных включает в себя:

– наименование операции интерфейса;

– список параметров (от нуля и более);

– типы аргументов и результатов, иначе – *void*;

– управляющий параметр или описание исключительной ситуации и др.

Атрибуты передаваемых параметров начинаются служебными словами: in – при отсылке параметра от клиента к серверу; out – при отправке параметров-результатов от сервера к клиенту; inout – при передаче параметров в оба направления (от клиента к серверу и обратно).

Описание интерфейса для одного объекта может наследоваться другим объектом и тогда это описание становится базовым. Пример такого дан ниже:

```
const long l=2
interface A {
void f (in float s [l]); }
interface B {
const long l=3 )
interface C: B, A { }.
```

Интерфейс С использует интерфейс В и А. Это означает, что интерфейс С наследует описание типов данных А и В, которые по отношению к С являются внешними. Но при этом синтаксис и семантика остаются неизменными. Согласно приведенного примера - операция функции *f* в интерфейсе С наследуется из А.

4). Способ сборки в GRID (www.gloubos.org) обеспечивает взаимодействие ресурсов через вызовы RPC/RMI в программах на ЯП с помощью операций assembling, config, make и устанавливают связь ресурсов между собой при работе с Cloud Computing и Big Data.

Система GRID (рис.6) обеспечивает обработку и управление программными, сервисными и др. Web ресурсами глобальной сети.

Сборка разнородных сетевых и системных ресурсов в Веб системы, приложения и пакеты, которые работают с данными разного объема, проводится в системе ETICS GRID (рис.6).

Ресурсы описываются в разных современных ЯП. Базовые сущности систем, пакет и приложений, а также связи описываются в CIM (Common Information Model).

При сборке ресурсов системе ETICS используется стандартизованное описание ТД для главных объектов: *Проект, Подсистема и Компонент. Проект.* Подсистема может содержать только Компоненты. В них используется модель данных CIM, задающая связь между разными объектами и описывать объекты и связи между ними, а также передавать результаты их выполнения по запросам. Сохранение и ведение данных базируется на модели реляционного типа в MySQL.

Описание модели данных основано на следующих базовых положениях:

1) каждый компонент содержит описание сведений (имя, лицензия, URL репозитория и т. п.) и глобального уникального идентификатора – ID (GUID);

2) объект конфигурации содержит информацию о версиях, связи с репозиторием, GUID, платформе фреймворка и связь с конфигурацией системы;

3) каждый компонент проходит проверку (checkout) скомпилированного элемента, тестовых команд и GUIDs, а также связь с конфигурацией;

4) при определении конфигурации и платформы в каждом объекте появляется GUID, его свойства, среда выполнения и зависимости, которые могут объявляться статически или динамически. Статическая зависимость – это взаимоотношение между двумя конфигурациями, динамическая зависимость – взаимоотношение между конфигурацией и модулем.

ETICS функции соответствуют концепции современной фабрики программ. Она базируется на наборах характеристик, услуг и процедур изготовления пакетов.

Пакеты могут объединяться плагинами с услугами для потребителей и поставщиков, обеспечивать управление заданиями с рабочих мест, а также давать доступ к OCAM, архитектуре CPU, компиляторам в ЯП и средствам спецификации зависимостей между разными пакетами и их тестами при сборке программ и их развертывании. Множество функциональных плагинов обеспечивает проверку контрактов, тестов выполнения разных элементов систем, генерацию документации, ведения готовых КПИ в оперативном или статическом репозитории ETICS.

Главная задача системы ETICS состоит в преобразовании некоторых компонентов систем для альтернативной платформы гетерогенной среды компьютеров методом ссылок с 16-, 32-разрядных платформ на 64-разрядную платформу среды Grid.

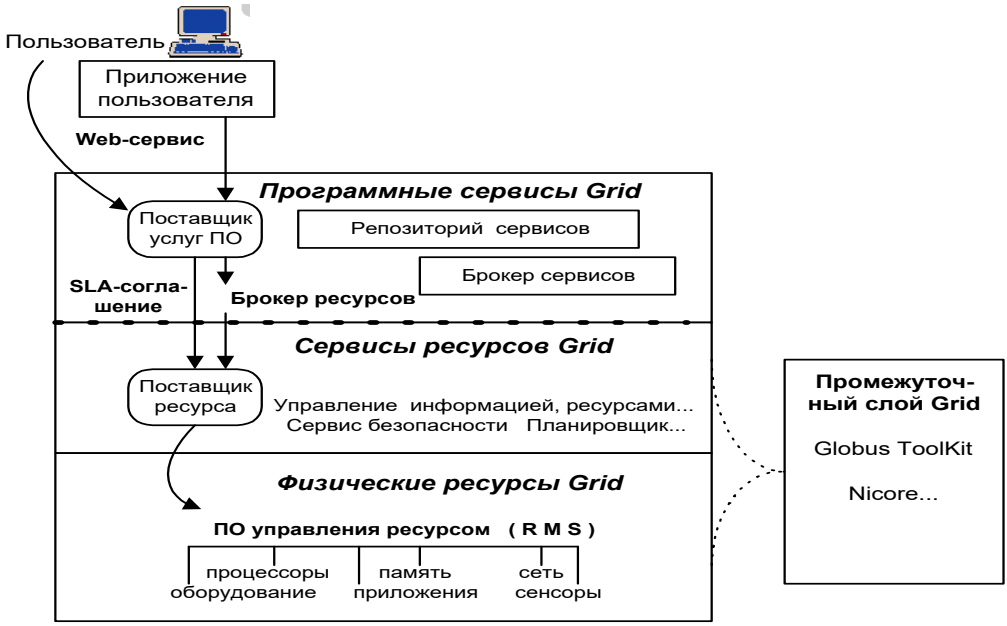


Рис.6. Общая структура функционирования GRID

Вопросами поддержки интерфейса ПО занимается *UNICORE* (UNiform Interface to COmputing REsources) (www.unicore.org). Обеспечение безопасности и защиты данных ресурсов и систем осуществляет инструмент WSRF (Web Service Recourse Framework).

5) Способ сборки по типу фабрик программ в IBM WebSphere**,** Microsoft Biz Talk, BEA WebLogic Oracle 10g, SAP NetWeaver и ИВК «Юпитер». В них содержатся CASE-средства сборки (link) ресурсов (сервисов, компонентов и данных) через брокера обмена данными stub и skeleton системы CORBA, переедаемых данные с помощью протокола Workflow с проверкой безопасности и качества (табл. 1).

Таблица 1. CASE–системы сборки ресурсов

Платформа	**Фирмы**	**Содержание платформы**
IBM WebSphere	Корпорация IBM	Сервер приложений J2EE, брокеры обмена данными, КПИ, портал, workflow/BPM, EII, SCA
Microsoft Biz Talk 2004 и компоненты .Net	Корпорация "Майкрософт"	Сервер приложений COM, брокеры обмена данными, RGB доставка, портал, workflow/BPMN
BEA WebLogic	Корпорация "BEA Systems" (с 2008г. входит в "Oracle")	Сервер приложений J2EE, брокеры обмена данными, ГОР, сервер прикладных приложений, портал, workflow/BPMN
Oracle 10g	Корпорация "Oracle" http://www.oracle.com	Сервер приложений J2EE, брокеры обмена данными, ГОР, портал, workflow/BPMN, средства EII
SAP NetWeaver	Корпорация SAP http://www1.sap.com/www.sap.ru	Сервер приложений J2EE/ABAP, брокеры обмена данными, портал, инструменты BPMN

ИВК "Юпитер	Компания ИВК (Россия) http://www.ivk.ru/	Брокеры обмена данными, КПИ, среда выполнения, сертификация, защита данных

На фабриках программ этих систем модифицируется архитектура в соответствии с требуемыми кодами сервисных ресурсов модели SOA в Visual Studio Industry Partners (VSIP). При этом используются модели Guidance Automation eXtensions (GAX) и Guidance Automation Toolkit (GAT) в Visual Studio. GAX — это среда исполнения в VSIP с использованием пакетов рекомендаций. Существует два варианта служб: веб-служба ASP.NET (ASMX) для Windows Communication Foundation (WCF) в среде .NET Framework 3.0. Версии для веб-службы WCF находятся у разработчиков фабрики ПО GotDotN. В ее состав входят процессы предприятия и пакеты документаций и рекомендаций для приложения типа Global Bank. Это аналогично модели SCA в IBM WebSphere.

6). Способы автоматизированной сборки (интеграции) программ на CASE инструментах: Make, Apache Ant, Apache Maven, Gradle и др.

Make — это кросс-платформенная система автоматизации сборки систем из исходного кода. Она генерирует файлы управления сборкою, например, Makefile в системах Unix для сборки посредством операции make (см. п.4.)

Apache Ant — JAVA-утилита для автоматизации процесса сборки программного продукта. Ant — платформо-независимый аналог UNIX-утилиты Make, но с использованием языка JAVA, приспособленного для JAVA-проектов. Самая важная разница между Ant и Make состоит в том, что Ant использует язык XML для описания процесса сборки и его зависимостей, а Make имеет свой собственный формат файл Makefile. XML-файл (или build.xml) осуществляет сборку исполняемых программ.

Apache Maven — программный инструмент для управления (*management*) JAVA-проектами методом сборки (building) ПО аналогично Apache Ant, но с более простой build-моделью конфигурации, которая базируется на формате XML. Двигатель ядра инструмента динамически загружает плагины из репозитория и обеспечивает доступ ко многим версиям разных JAVA-проектов с открытым кодом Apache.

Gradle — система автоматической сборки, построенная на принципах Apache Ant и Apache Maven, но вместо XML-подобной формы здесь используются языки DSL и Groovy для представления конфигурации создаваемого приложения или проекта.

2.2. Создание Общего сборщика разнородных ресурсов Интернет

В качестве общего вывода по рассмотрению операций сборки можно сделать вывод о том, что приведенные операции сборки (link, make, config, assembling, building, weaver, integration) ресурсов в системных средах (3.2 -3.9), являются базовыми средствами для создания общего «сборщика» готовых ресурсов для прикладных, сервисных, информационных и технических систем в сетевой Глобальной сети Интернет. Основу сборщика ресурсов составляет теория преобразования сложных типов данных стандарта ISO/IEC 11404 GDT к более простым, с которыми могут вычисляться любые прикладные системы. Далее рассматриваются общие положения теории преобразования типов данных стандарта ISO/IEC 11404-2012 GDT при проведении сборки двух обектов.

2.2.1. Теоретический базис преобразования ТД FDT, GDT, GLD

Разнородные КПИ, сервисные компоненты работают с данными, которые включают множество значений, операции над этими значениями и взаимодействие выполнения операций над ними. Первоначально в методе сборки модулей использовалась аксиоматика FDT (Fundamental Data Types) в классе ЯП для ОС ЕС. Эта аксиоматика разработана известными специалистами Э. Дейкстрой, Н. Виртом, В. Турским, В.Н. Агафоновым и др. в

70-годы. Позднее в 1996 г. разработана аксиоматика общих типов данных (General Data Types — GDT) в стандарте ISO/IEC 11404-GDT, 1996, 2007.

К фундаментальным типам данных (FDT) относятся:

– сложные (массивы, таблицы, файлы, записи, множества, деревья и др.).

Эти ТД задают базовые значения данных в программах на ЯП, которые проверяется на этапе компиляции на соответствие типов. На этапе выполнения они реализуются с помощью предикатов полиморфных типов. КПИ обмениваются данными, ТД которых должны соответствовать друг другу. В случае несоответствия ТД (например, передается целое — integer, а требуется для выполнения вызываемого модуля real), проводятся эквивалентные преобразования обмениваемых данных (integer ⇔ real) с помощью примитивных функций библиотеки интерфейсов АПРОП для FDT. К проблемам преобразования ТД в разных ЯП относятся:

а) несоответствие количества (формальных и фактических) параметров или неверное их описание;

б) несогласованность типов передаваемых параметров или их значений для форматов компьютеров;

в) отсутствие прямых и обратных преобразований параметров и др.

Так как FDT не охватывал все виды данных и типов новых ЯП после 1992г., то появился новый стандарт общих типов донных GDT, которые и использовались в информационных и прикладных системах. В стандарт GDT включены FDT и новые сложные типы данных — контейнеры, указатели, множества, списки, последовательности, неструктурные данные и т.п.

Данный стандарт ISO/IEC 11404 GDT — 1996 прошел многолетнюю апробацию и вышел новый вариант в 2007г. В его состав входят такие типы данных: агрегатные, генеративные, расширяемые и др., которые требуют их генерации к фундаментальным ТД для последующего применения в ресурсах Глобальной сети Интернет. Требуется разработка новых примитивных функций преобразования неэквивалентных ТД для новых ЯП (C, C++, Python, Basic, Ruby, JAVA и др.) и других типов (рис.7).

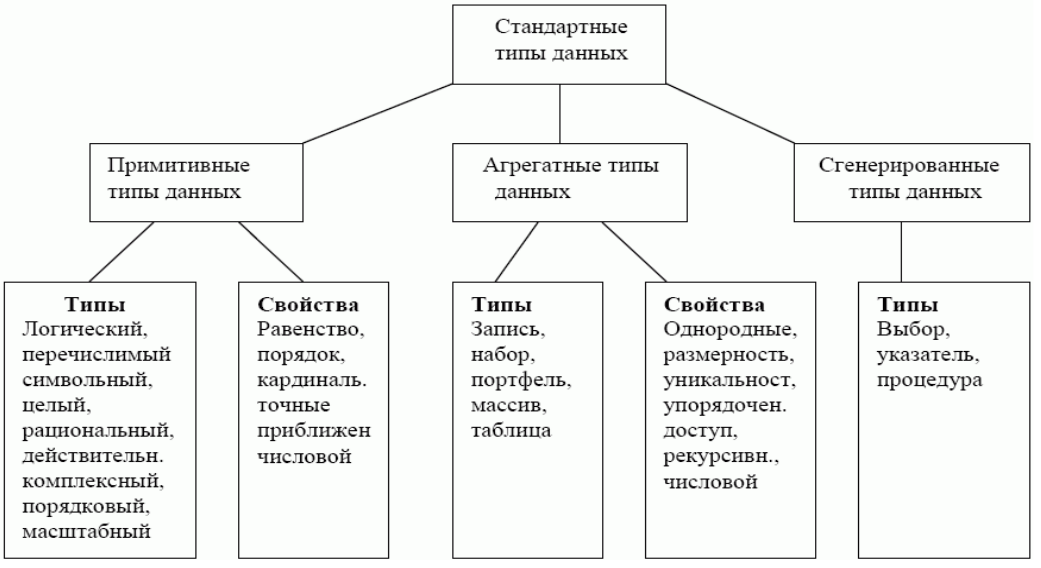


Рис. 7. Типы данных стандарта GDT

Согласно приведенной схемы реализуются следующие задачи

– преобразования данных для поддержки сборщика ресурсов:

– специфицирование внешних ТД в WSDL, сохранения их в БД и в репозиториях;

– преобразование новых ТД в новых $ЯП_1$, …, $ЯП_n$;

– реализацию ТД GDT к виду специальных примитивных функций FDT;

– представление ТД FDT к виду специальных примитивных функций и формату Фреймворка

– эквивалентные отображения и генерацию данных GDT<=>FDT с учетом платформ современных компьютерных и кластерных систем Интернет, где ресурсы будут работать.

Для решения проблем преобразования данных согласно стандарта GDT и FDT при связывании (взаимодействии) КПИ и сервисов, заданных в разных современных ЯП, предложен подход к генерации GDT<=>FDT (рис.8), Основу этого подхода составляет задачи преобразования типов данных в сборщике ресурсов с использованием ранее созданных для FDT .

Полуструктурированные, неструктурированные и расширяемые ТД данного стандарта используются в информационной среде, в связи с исследованиями недр земли, космоса и океана. Потребуется практическая обработка этих новых структур данных, чтобы с ними решались эффективно прикладные задачи в разных прикладных областях, особенно работающих с Big Data.

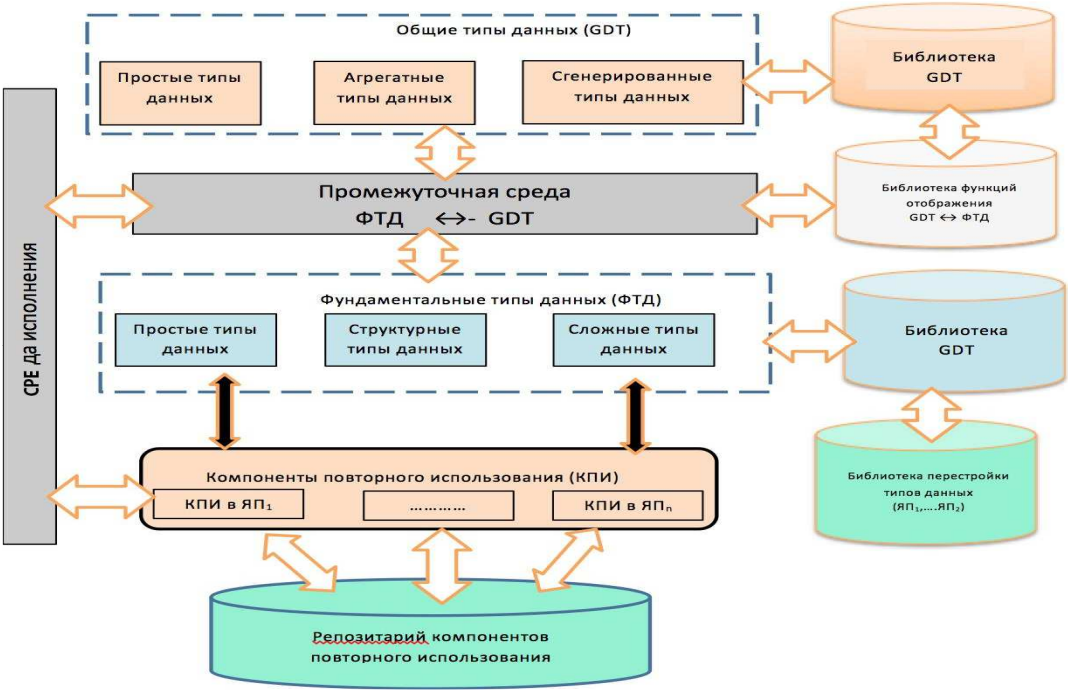


Рис.8. Схема преобразования ТД стандарта GDT

Подходы к обработке неструктурированных данных Big Data

Big Data — набор данных большого объема, а также подходов, средств, инструментов и методов представления неструктурированных огромных объёмов данных для получения данных и эффективного использования их на многочисленных узлах сети Интернет при решении прикладных Intelligence ИС и ИТ систем с использованием СУБД. Для работы с большими объемами данных сформировался метод ETL (Extract Transform Load), с помощью которого производится:

– извлечение данных из внешних источников;

– трансформация и очистка данных с учетом требований;

– загрузка данных в хранилища данных;

– анализ данных и их перенос из одного приложения в другое и др.

К основным свойствам Big Data относятся**:**

– *горизонтальная масштабируемость* обрабатываемых больших данных и большого количества кластеров и серверов;

– *отказоустойчивость* по отношению к сбоям на процессорах кластеров и в узлах сети. Так, инструмент Hadoop-кластер Yahoo имеет более 42000 компьютеров, среди которых часть из них может выходить из строя;

– *локализация данных и о*бработка их на серверах, где практически хранятся большие данные для решения соответствующих задач;

– *изменение числа работающих* на кластере с помощью средств MySQL Cluster. Большие данные могут быть представлены как tensors, которые управляют вычислением (например, полилинейное обучение подпространств Multilinear Subspace Learning и др.).

Системные средства обработки Big Data

NoSQL-БД нереляционные и распределенные данные с открытым кодом и горизонтальной масштабируемостью, эффективно поддерживают случайное чтение, запись и версионность.

MapReduce — модель распределённых вычислений, которая используется при параллельных вычислениях с большими данными в компьютерных кластерах.

Hadoop — свободно распространяемый набор утилит, библиотек и фреймворков для разработки и выполнения распределённых приложений (в том числе, MapReduce-программ), работающих на кластерах с сотнями и тысячами узлов.

СМБ — системы обработки больших данных в рамках Cloud-вычислений.

Big Data анализируются средствами**:** статистических и динамических методов анализа искусственного интеллекта, нейронных сетей, математической лингвистика; A/B Testing, Crowdsourcing Data Fusion; Integration Genetic Algorithms Machine Learning; Natural Language Processing; Signal Processing Simulation and Visualization; Massively Parallel Processing; Search-Based Applications, Data Mining, Multilinear Subspace Learning и др.

2.3. Технологии программирования (ТП) и путь ее развития после 1992

Теория ТП, как математическая дисциплина, сформулирована А.П. Ершовым и зарубежными учеными (E.Dijksra C. Hoar, Z. Manna, D.Gries, D.Buirner, K. Parnas и др.). Эта теория основывается на теории алгоритмов, множества, алгебры, логики исчислении предикатов, комбинаторики и др. В докладе на звание академика СССР (1984) и на Всесоюзной конференции «Технология программирования» (1986) А.П.Ершов определил теорию ТП, которая включает методы синтеза, сборки и конкретизации. *Синтезирующее программирование* базируется на методе доказательного рассуждений о правильности алгоритма программы. *Сборочное программирование* осуществляет объединение совокупности существующих (проверенных на правильность) готовых фрагментов программ (reuses) для задач предметной области и сборку их в сложную структуру. *Конкретизирующее программирование* обеспечивает построение системы по универсальной модели для некоторой предметной области». Основные положения теории ТП он сформулировал, как «конкретный способ организации, создания, распространения и сопровождения программного продукта (ПП).

Технология и методология – это *наука,* а метод входит в них составной частью. Технология начинается тогда, когда она охватывает ЖЦ создания ПП для предметной области.

Математическая наука программирования по мнению Ершова А.П. имеет три перспективных направления развития:

Первое направление (*организационное программирование*) 1975–1985 гг.

Второе направление (*сборочное программирование*) 1985–1995 гг.

Третье направление (*доказательное программирование*) 1995–2005 гг.

К методам ТП Ершов относил функциональное (В.П.Турчин), расслоенное А.Л.Фуксмана, расширяемое программирование М.М.Горбунов-Посадова и параллельное программирование и др. После 1992года многие отечественные методы программирования слабо развивались и в основном применялись зарубежные технологии ООП, UML, VDM, Agile и др.

SEMAT (Software Engineering Methods and Theory (2009). Согласно стандарту SEMAT развитие теории программирования требуют математизации программирования для разных видов прикладных областей. (биология, генетика, медицина и др.) с помощью ЯП (C++, Java, Python, Ruby и др.) с объектной ориентацией и строгое математическое определение разных видов создаваемых прикладных систем с обеспечением качества.

В XXI веке согласно статьи B. Boehm **“**Perspectives and problems of Software Engineering” (IEEE Computer Society, v.41, N3 2008) определены **проблемные задачи** на ближайшие 10 лет:

– обеспечение изменений Legacy-систем (ОС IBM, ОС MS и др.) с применением средств dependability по модели CMM и масштабирование изменений;

– высокие цены на многомиллионное ПО компьютеров, базируется на методе сборки приложений из готовых платных ресурсов для разных областей;

– моделирование физических экспериментов проводить по методу ASC (Advanced Simulation Alliance) в суперкомпьютерных центрах;

– аутентификация серверов и защита каналов между клиентом и сервером (SSL и TSL) с открытыми ключами и паролями в перспективе требует управление ресурсами QOS (quality-of-service) с гарантией качества;

– развитие перспективных технологий средствами Grid-системы (рис.5).- http://www-fp.grids-center.org/news/pdf/HPDC13-Grid3)

Появились онтологические и интеллектуальные средства в Семантик Веб Интернет и проведено конвейерное производство ПП (APP.FAB) - фабрики программ (WebShpereIBM, MS.Net) разного назначения. Определилась концепция моделирования вариантов ОС (Linux, OS IBM, OS MS, Unix и др.) в рамках научного проекта VAMOS (Variability Model OS) и SEPL. Проведены конференции (EuroSys'11, SPLC'12, PlOS’12 and so on) для создания динамических вариантов систем ПО и ОС с помощью модели Kconfig.

Аспекты теории сборки и преобразования данных ресурсов

Сущность теории решения задачи сборки пар разнородных модулей или объектов состоит в построении взаимно однозначного соответствия между множеством формальных и фактических параметров.

$V = \{v^1, v^2, ..., v^\kappa\}$ вызов объекта с множеством формальных параметров;

$F = \{f^1, f^2, ..., f^{\kappa l}\}$ вызов объекта и их отображение с помощью алгебраических систем.

Каждому типу данных T_α^t языка l_α ставится в соответствие алгебраическая система вида:

$$G_\alpha^t = <X_\alpha^t , G_\alpha^t > ,$$

где X_α^t — множество значений рассматриваемого типа данных (ТД), а G_α^t — множество операций над объектами данного типа. Операциям преобразования ТД соответствует изоморфное отображение алгебраических систем G_α^t в G_β^d.

В классе алгебраических систем $\Sigma = \{G_{\alpha}{}^{b}, G_{\alpha}{}^{c}, G_{\alpha}{}^{i}, G_{\alpha}{}^{r}, G_{\alpha}{}^{a}, G_{\alpha}{}^{z}\}$

где тип t – это b — boolean, c — character, i — integer, r — real, a — array, z — record и др. В системе Σ реализованы все виды преобразований для представленных ТД и проведено доказательство изоморфизма алгебраических систем и его отсутствие для множеств неэквивалентных значений $X_{\alpha}{}^{t}$ и $X_{\beta}{}^{q}$. Установлено, что мощности алгебраических систем равны

$|G_{\alpha}{}^{t}| = |G_{\beta}{}^{q}|$.

В парадигме сборки реализованы универсальные формальные основы преобразования типов входных / выходных данных (простых и сложных) FDT к общим типам данных GDT стандарта ISO/IC 11404 (примитивные, агрегатные, генерированные и неструктурированные). Ниже рассматриваются общие вопросы преобразования FDT, GDT и GDT<=>FDT, как очень важные для сборщика программ для среды Интернет.

Подходы к обработке неструктурированных данных Big Data

Big Data — набор данных большого объема, а также подходов, средств, инструментов и методов представления неструктурированных огромных объёмов данных для получения данных и эффективного использования их на многочисленных узлах сети Интернет при решении прикладных Intelligence ИС и ИТ систем с использованием СУБД.

Для работы с большими объемами данных сформировался метод ETL (Extract Transform Load), с помощью которого производится:

извлечение данных из внешних источников;

трансформация и очистка данных с учетом требований;

загрузка данных в хранилища данных;

анализ данных и их перенос из одного приложения в другое и др.

К основным свойствам Big Data относятся**:**

горизонтальная масштабируемость обрабатываемых больших данных и большого количества кластеров и серверов;

отказоустойчивость по отношению к сбоям на процессорах кластеров и в узлах сети. Так, инструмент Hadoop-кластер Yahoo имеет более 42000 компьютеров, среди которых часть из них может выходить из строя;

*локализация данных и о*бработка их на серверах, где практически хранятся большие данные для решения соответствующих задач;

изменение числа работающих на кластере с помощью средств MySQL Cluster. Большие данные могут быть представлены как tensors, которые управляют вычислением (например, полилинейное обучение подпространств Multilinear Subspace Learning и др.).

К системным средствам обработки Big Data относятся:

NoSQL-БД нереляционные и распределенные данные с открытым кодом и горизонтальной масштабируемостью, эффективно поддерживают случайное чтение, запись и версионность.

MapReduce — модель распределённых вычислений, которая используется при параллельных вычислениях с большими данными в компьютерных кластерах.

Hadoop — свободно распространяемый набор утилит, библиотек и фреймворков для разработки и выполнения распределённых приложений (в том числе, MapReduce-программ), работающих на кластерах с сотнями и тысячами узлов.

CMD— системы обработки больших данных в рамках Cloud-вычислений.

Big Data анализируются средствами**:** статистических и динамических методов анализа искусственного интеллекта, нейронных сетей, математической лингвистика; A/B Testing, Crowdsourcing Data Fusion; Integration Genetic Algorithms Machine Learning; Natural Language

Processing; Signal Processing Simulation and Visualization; Massively Parallel Processing; Search-Based Applications, Data Mining, Multilinear Subspace Learning и др.

2.4. Описание задач сборщика интеллектуальных и информационных ресурсов в Интернет

В качестве общего вывода по рассмотрению операций сборки можно сделать вывод о том, что приведенные операции сборки (link, make, config, assembling, building, weaver, integration) ресурсов в системных средах (3.2-3.8), являются базовыми средствами для создания общего «сборщика» готовых ресурсов для прикладных, сервисных, информационных и технических систем в сетевой Глобальной сети Интернет.

Приведенные операции сборки (link, make, config, assembling, weaver) ресурсов в заданных системных средах в п.3.1-3.8 имеют схожие операционные способы сборки разных вариантов ресурсов: модулей в разных ЯП, исходных и выходных текстов компиляторных программ, сервисных и системных ресурсов Веб среды Интернет, технических средств компьютеров имеют похожие способы сборки, которые повторяются в разных общесистемных средах. Как бы они не назывались способы сборки семантика сборки остается одинаковой. Способ сборки зависит от данных, которые используются при обмене данными между связываемыми разнородными ресурсами.

Рассмотренные средства генерации общих типов данных стандарта ISO/IEC 11404 GTB – 2012 требуют создания набора новых примитивных функций для нестандартных типов данных (портфелей, контейнеров, шаблонов, указателей, неструктурных данных и др.) и использоваться в названных способах сборки разнородных ресурсов Интернет.

Для создания общего «сборщика» готовых ресурсов в прикладные и технические (макроконвейерные) системы в Интернет, необходимо на базе конфигурационной сборки сделать следующее:

Определить формальный вариант задания операции сборки ресурсов

Создать библиотеку примитивных функций для всех систем Интернет и дорабатывать примитивные функции для новые ТД (контейнеров, шаблонов и др.) согласно стандарта GDT.

Создать анализатор (агент) операции сборщика и взаимодействия (компиляторных, системных, прикладных, технических) с учетом всех типов ресурсов.

Проводить анализ соответствия типа ресурса и взаимодействия с другим ресурсом через интерфейс и осуществлять обращение к соответствующим программам преобразования обмениваемых данных для генерации соответствующего преобразования по теории преобразования типов данных стандарта GDT.

Управлять конфигурационной сборкой выполняя все процессы, определенные в стандарте IEEE 828 Configuration.

Выдавать сведения о результатах сборки и завершения работы сборщика.

Обеспечить размещение ресурсов в библиотеках и хранилищах Интернет из разных предметных областей знаний (медицины, биологии, генетики, математики и др.).

Техническим результатом сборки общего сборщика является:

простота поиска ресурсов в хранилищах Интернет и снижение затрат на разработку за счет готовых правильных многоразовых ресурсов и настройку их на конкретные условия применения;

повышение качества и производительности создаваемых из ресурсов Веб-приложений и систем за счет системных операций замены отдельных более правильных ресурсов и изменения конфигурации (архитектуры) варианта конфигурационного файла с получением качественных решений функциональных задач приложений в разных предметных областях знаний.

2.5. Конфигурационная сборка сервисных ресурсов Web-систем

Под *конфигурацией системы* понимается структура некоторой ее версии, включающая функции, объединенные между собой операциями связи с параметрами, задающими режимы функционирования системы.

Версия или конфигурация системы согласно IEEE Standard 828-2012 (Configuration) включает:

– базис конфигурации — BC (Configuration Baseline);

– элементы конфигурации (Configuration Item);

– компоненты, ГОР, входящие в описание моделей M_{sys}, M_{wsys};

Управление конфигурацией (Configuration Management) заключается в наблюдении за модификацией параметров конфигурации и компонентов системы, а также в проведении систематического контроля, учета и аудита внесенных изменений, целостности и работоспособности системы на процессах:

– Идентификация конфигурации (Configuration Identification).

– Контроль конфигурации (Configuration Control).

– Учет статуса конфигурации (Configuration Status Accounting).

– Аудит конфигурации (Configuration Audit).

– Трассировка конфигурации при сопровождении и эксплуатации системы;

– Верификация компонентных сервисов по моделям систем и веб-систем.

При конфигурационной сборке сервисных ресурсов используется модели систем и модель характеристик MF (Model Feature). КПИ хранятся в репозиториях или библиотеках системы. Они выбираются их библиотек, адаптируются и интегрируются в систему. Основную роль в этих процессах выполняет конфигуратор, например в (http://7dragons.ru/ru) рис.9.

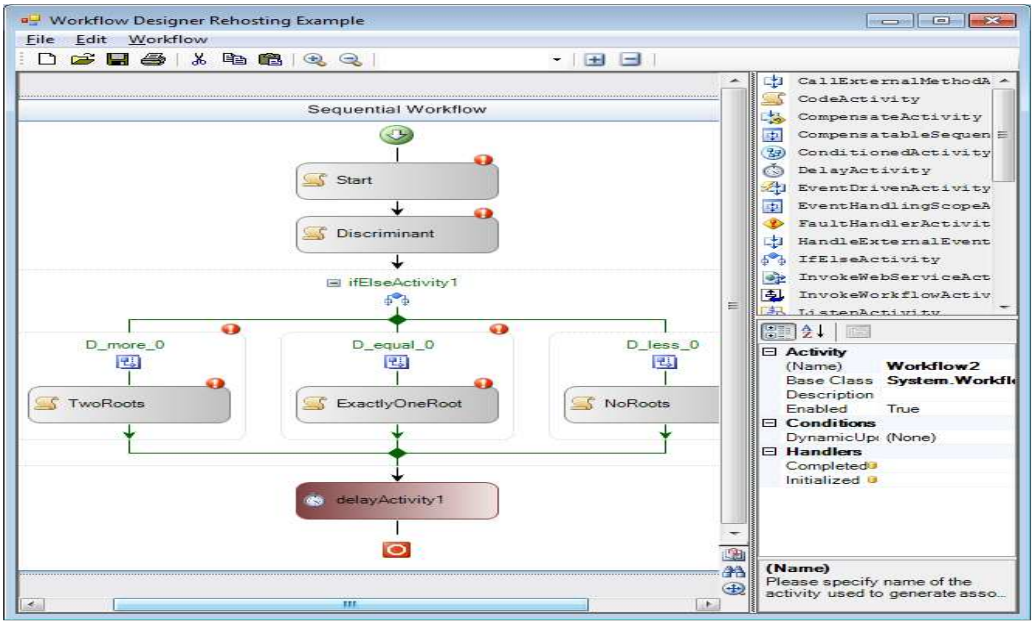


Рис.9. Модель конфигуратора

Конфигуратор управляет созданием варианта готового продукта и сохраняет его в репозитории.

М*одель среды конфигуратора* включает:

- графовую структуру системы из ресурсов;

- модель вариантов системы;

- аудит конфигурации системы;

- верификация моделей и ресурсов;

- оценку качества КПИ и систем.

Конфигуратор с помощью операции config собирает веб-систему (http://7dragons.ru/ru) с учетом модели M_{sys}, M_{MF} и заданных ресурсов предметной области и их интерфейсов. Проводит оценивание сконфигурированного файла на качество и безопасность.

2.6. Задачи сборка экспериментального варианта ядра OS Linux

OS Linux содержит более 10 000 переменных и большое множество функциональных системных компонентов, обеспечивающих обработку разного рода заданий по функционированию любых прикладных систем под управлением OS Linux представлен в отчете 2019 года и в схеме (рис.9):

На верхнем уровне находится пользовательское пространство, где находятся приложения пользователей и библиотека GNU C (glibc) с интерфейсами системных вызовов для связи с ядром. Ядро OS Linux содержит более 11000 программных элементов и является общим для всех процессорных архитектур, поддерживаемых ОС Linux. Следующий уровень - архитектурно-зависимый код ядра BSP (Board Support Package) определяет конкретную архитектуру процессора и платформы. OS Linux можно откомпилировать для огромного количества разных процессоров и платформ, имеющих разные архитектурные ограничения и потребности. OS Linux может работать на процессоре как с блоком управления памятью (MMU), так и без MMU.

К основным компонентам ядра OS Linux относятся следующие:

- интерфейс системных вызовов функций ядра;

- управление процессами или потоками с помощью интерфейса программирования приложений (API) через SCI для создания нового процесса и алгоритма планировщика, время работы которого не зависит от числа потоков;

- управление памятью с участием аппаратных средств, устанавливающих соответствие между физической и виртуальной памятью;

- определение виртуальной файловой системой (VFS), которая предоставляет общую абстракцию интерфейса к файловым системам и служит для коммутации между SCI и файловыми системами ядра;

- сетевой стек имеет многоуровневую структуру самих протоколов.

Internet Protocol (IP) - это базовый протокол сетевого уровня, располагающийся ниже транспортного протокола Transmission Control Protocol, TCP). Выше TCP находится уровень сокетов, вызываемый через SCI. Уровень сокетов представляет собой стандартный API к сетевой подсистеме. Он предоставляет пользовательский интерфейс к различным сетевым протоколам;

- драйверы устройств обеспечивают возможность работы с конкретными аппаратными устройствами.

В OS Linux разработан гипервизор, с помощью которого можно строить вариант ОС для других систем. На основе ядра ОС выбраны необходимые компоненты и создана модель MF с базовыми характеристическими компонентов ОС и модели системы *Msys,* включая множество функциональных M_{fi}, интерфейсных M_{io} и M_d работы с данными базового ядра OS Linux.

2.7. Парадигмы программирования задач предметных областей знаний

Парадигма (от греч. παράδειγμα, «пример, модель, образец») – это совокупность фундаментальных научных установок, представлений и терминов, принимаемая и разделяемая научным сообществом.

Парадигма обеспечивает преемственность развития науки и научного творчества в некоторой предметной области. Томас Кун называл парадигмами устоявшиеся системы научных взглядов, в рамках которых ведется разработка формальных математических механизмов.

Парадигма программирования – это совокупность идей, понятий, теорий и методов, определяющих стиль написания компьютерных программ. Этот термин Р.М. Флойд определил в своей работе «The Paradigms of Programming» (Communications of the ACM. 1969. V. 22(8). P. 455–460), Э. Дейкстра в книге «Дисциплина программирования» (М.: Мир, 1976) и Д. Грис в книге «Наука программирования». Они назвали парадигму программирования способом концептуализации и формального определения программ и систем для последующего проведения вычислений на компьютере. Таким образом, *парадигма программирования* включает теорию, метод и средства программирования компьютерных систем.

Приведем классификацию парадигм программирования:

- прикладное, декларативное программирование (функциональное, логическое, автоматное, алгебраическое, агентное и др.);

- теоретическое программирование (модульное, объектное, компонентное, аспектное, сервисное и др.);

- системное *программирование (параллельное, распределенное и др.) для высокопроизводительных компьютеров, кластеров и др.*

Рассмотрим некоторые парадигмы теоретического плана.

Парадигма объектного программирования. Эта парадигма основана на графовом представлении структуры моделируемой прикладной системы. В вершинах графа располагаются функциональные объекты и интерфейсные объекты.

Разработана алгебра операций взаимодействия функциональных и интерфейсных объектов между собою и объектной средой, применяемая при графовом представлении систем.

На основе графа G можно задать несколько подсистем предметной области знаний (M_{sysi}) обработки функций F_i:

$M_{sys1} = <F_1 (f_{O2} ((Io_5, f_{O5}), (Io_6, f_{O5})); f_{O3}; f_{O4} ((Io_7, f_{O7}), (Io_8, f_{O8}))>,$

$M_{sys2} = <F_2 (f_{O2} ((Io_5, f_{O5}), (Io_6, f_{O5})>;$

$M_{sys3} = <F_3 (f_{O3})>;$

$M_{sys4} = <F_4 ((Io_7, f_{O7}), (Io_8, f_{O8}))>.$

С помощью треугольника Фреге объектам задаются уникальные имена:

знак — идентификатор, который задает сущность объектной реальности;

денотат — сущность, смысл;

концепт — совокупность свойств или предикатов из унарных предикатов «иметь свойство» с помощью логических связок.

Эти данные определяют сущность объекта описания для его использования в конфигурационной сборке варианта функций F_i или соответствующей программы ее реализации в среде систем VS.MS, IBMSphere, Java, Linux, Intel.

Сервисно-компонентное программирование

Основано на системных и сервисных и компонентных ресурсах Интернет. Сервис Интернета представляет собой ресурс, который реализует некоторые общие функции в службе Интернет (сервисы транзакций, именования, безопасности, защиты и др.).

Веб-сервис имеет URL-адрес, интерфейс и механизм взаимодействия с другим сервисом через протоколы Интернет или связи с другими программными элементами БД и деловыми операциями. Обмен данными между веб-сервисом и КПИ осуществляется с помощью XML-документов, оформленных в виде сообщений. Веб-сервисы обеспечивают решение задачи *интеграции (сборки) приложений* разной природы, являясь при этом инструментом построения распределенных систем. К основным средствам описания и разработки новых систем средствами веб-сервисов относятся:

язык XML для описания и построения SOA-архитектуры;

язык WSDL (Web Services Description Language) для описания веб-сервисов и их интерфейсов в XML, а также типов данных, сообщений и протоколов связи сервисов;

SOAP для определения форматов запросов к веб-сервисам;

SCA для создания более сложной системы на основе компонентов и сервисов;

UDDI для описания и интеграции сервисов, их хранения в библиотеках.

Для сборки сервисов имеется системный инструмент — ***Jopera for Eclipse*** (http://www.jopera.ethz.ch/). В нем есть набор Eclipse-плагинов для связи различных программных элементов и реализации итеративной композиции сервисов (через маршрутизаторы SOAP и RESTful Web-сервис, Grid-сервисы, Java snipets и др.) и моделирования процессов в сети. Для поиска сервисов по их семантическим описаниям используются *Feta Client* и *Feta Engine,* где *Feta Client* — это GUI-плагин системы Интернет Taverna для описания сервиса, а *Feta Engine* для задания Web-сервиса.

Клиент серверная архитектура Интернет для работы с ресурсами

Архитектура Интернет Браузера - трехуровневая: клиент, сервер приложений и сервер Баз Данных (БД).

На уровень *клиента* выносятся ресурсы веб-систем (приложений): интерфейс, операции с данными, алгоритмы шифрования и взаимодействия с сервером приложений и т.п. Клиент посылает запрос (request) в языке XML на сервер с помощью протоколов (HTTP, SMTP).

Сервер приложений обеспечивает горизонтальное масштабирование производительности веб-систем без внесения изменений в код выполняемой системы. Сервер принимает запрос от клиента, обрабатывает его и формирует ответ (response) клиенту.

Сервер БД запускается с сервера приложений и выполняет обслуживание БД с обеспечением целостности, сохранности данных и доступа клиента к информации БД. Для работы с Big Data на сервере решаются задачи управления и анализа данными большого объема, а также манипулирование данными с большой нагрузкой. Клиентская часть приложения Front-end отвечает за интерфейс к программно-аппаратной части веб-приложения. Front-end сервера обрабатывают приложения Интернет, которые занимают достаточное место в памяти. Back-end сервера запускает серверное приложение на ЯП (C, C++, Python, Ruby, Java, Basic и т.п.), организует их вычисление и обработки аварийных ситуаций.

2.8. Обеспечение надежности и качества создаваемых систем

Согласно стандарта ISO/IEC 12207 ЖЦ ПО регламентируется планирование, управление качеством и оценку затрат на создание системы. На этих процессах ЖЦ проводится анализ достижения качества; верификация и валидация (V&V) ресурсов и оценивание степени достижения отдельных показателей качества; тестирование готовой

системы; сбор данных об отказах, дефектах и др. ошибках; оценивание надежности по соответствующим моделям надежности с учетом результатов тестирования.

Модель качества стандарта ISO/IEC 9000 (1-4) Quality задает шесть показателей (характеристик) q_1 — q_6 (q — quality) качества:

q_1 — функциональность (functionality),

q_2 — надежность (reliability),

q_3 — удобство (usability),

q_4 — эффективность (efficiency),

q_5 — сопровождаемость (maintainability),

q_6 — переносимость (portability).

Каждая характеристика q_i рассчитывается по специальным формулам и метрикам стандарта. Надежность оценивается согласно полученных на процессе тестирования ошибок, дефектов и отказов в ПО и по соответствующим моделям надежности (оценочным, измерительным и др.).

Данные по всем показателям качества q_1 — q_6 оцениваются по формуле:

$$q_1 = \sum_{j=1}^{6} a_{1j} m_{1j} w_{1j}$$

где a_i — атрибуты каждого показателя качества (i = 1..6); m_i — метрики каждого атрибута качества; w_i — вес каждого атрибута показателя качества системы. Полученные значения по показателям модели качества входят в сертификат качества продукта. Варианты оценки показателей качества сконфигурированных систем приведены на сайте ИТК.

3. Средства Интернет для представления знаний

3.1. Средства создания моделей знаний

Процесс создания и документирования *моделей знаний* задается методологическими системами, основанными на знаниях (Knowledge-based systems, KBS), CommonKADS. Их основу составляет идея построения библиотек, содержащих элементы решения задач в проблемных областях, являющиеся повторно использованными (reuses) для других областей. Данные KBS и CommonKADS расширены агентной методологией для проведения мультиагентного анализа и проектирования систем, основанных на знаниях. В агентной методологии определяется множество моделей следующего вида:

- *модель агентов* для спецификации характеристик, определяющих способность агента к рассуждению, сенсорным /эффекторам (sensor/effectors) и сервисам; - *модель задач*, которые могут выполняться агентами для достижения поставленных целей;

- *организационная модель*, которая задает социальную структуру сообщества агентов;

- *координационная модель,* задающая переговоры (общение) между агентами;

- *коммуникационная модель*, детализирующая взаимодействие агентов, людей и программных агентов;

- *модель проекта* системы задает типичные действия по проектированию, связанному с определением сети агентов, выбора наиболее подходящей архитектуры агентов и платформы для разработки новых агентов.

Единая *платформа знаний* для домена (Knowledge Domain) формируется инструментом ***GRACE*** (http://www.grace-ist.org/). В нем задаются ресурсы информации, онтологии и сообщества. Домен знаний представляется в виде концептуального фрейма, содержащего документы описаний, образующих единую структуру представления домена. Этот инструмент обеспечивает предварительную индексацию документов из многих источников контента с помощью *инженерии онтологии*, описываемой ниже.

Семантик Веб для представления знаний о предметных областях знаний

Semantic-WEB – эта глобальная информационная среда, которая содержит семантические ресурсы, языки и инструменты разработки онтологий, систем и бизнес-процессов с использованием накопленных в этой среде знаний.

Semantic Web предлагает мощный способ создания интеллектуальных программных приложений, использующих информационные и сервисные ресурсы, существующие в глобальной сети и внутри любого предприятия. Семантическое веб охватывает:

• Семантическую веб-архитектуру, инструменты и лучшие практики.

• Пути представления знаний и интеграции приложений.

• Методы объединения, выравнивания и вывода данных и информации в разных форматах.

• Взгляды в будущее семантической сети, включая расширенную интеграцию, распространение, расширенные рассуждения, визуализацию и др.

• Подробное рассмотрение рекомендаций OWL 2 W3C и определение их влияния на программные архитектуры и их создание.

• Существующие приложения в Semantic Web обеспечивают получение данных из многих источников, включая Facebook™, MySQL®, Jabber, а также выравнивание и унификацию информации, и их экспортирование в различные форматы.

Семантик Веб предоставляет научный сервис для представления и автоматизированной обработки научных задач, больших данных в разных форматах, интеграцию данных из коллажей (Mash-Ups), поиск и композирование веб-сервисов, управление интеллектуальными агентами в обильных приложениях и др. Веб обеспечивает решение задач с использованием многочисленных сервисов и научных достижений в области предоставления бизнес-услуг. Дает новые подходы к интеллектуализации больших объемов информации, основанных на новых словарях и концептуальных моделях, методах онтологизации новых научных задач и приложений с помощью предметно-ориентированных языков (OWL, DSL и др.) и инструментов (FODA, Protégé, DSLTool MS.Net и др.). Отдельные языки описания онтологий и инструментов Семантик Веб нами использовались для представления научных доменов «Вычислительная геометрия» и жизненного цикла (ЖЦ) систем ПО, выполненных с участием студентов МФТИ и КНУ в период 2011-2015.

3.2. Общее представление знаний онтология прикладных областей

Онтология, как инструмент представления знаний о прикладных областях, задает общие инженерные аспекты описания любой прикладной области, выполняя:

1. Задание основного набора запросов, которые систематически передаются множеству источников контента прикладной области для выбора подходящих документов и обеспечения поиска и обновления документов, созданных средствами GRACE;

2. Отбор документов для анализа содержания и связи с различными понятиями в концептуальной модели онтологии прикладной области;

3. Просмотр и выполнение запросов с индексами, которым соответствуют документы источника контента, ранее сформированного.

Концептуальная модель онтологии может использоваться системой Grid (в частности, Data Grid) Интернет. Публикация и распространение знаний поддерживаются *системами управления документами* (document management systems), которые используют языки разметки контента и индексирования, поиска и предоставления понятий контента. Некоторые из них используют персонификацию (customization or personalization) контента средствами AKT (Advanced Knowledge Technologies, www.aktors.org), позволяющим персонифицировать

контент и представлять его в соответствии с интересами пользователей данной онтологии (http://eldora.open.ac.uk/my-planet/).

Для публикации знаний используется инструмент ***ePrints*** (www.eprints.org), который предоставляет набор сервисов для архивирования, публикации и сопровождения сервисов с расширенными метаданными.

В этом инструменте создаются разнообразные сервисные знания, основанные на методах контентного поиска (content-based retrieval).

В развитие инструментов, поддерживающих создание электронных библиотек, и Grid-технологий e-science, в проекте *TextGrid* разработаны ***текстовые решетки Grid***. С их помощью предоставляются средства обработки, анализа, индексирования, аннотирования, редактирования и опубликования текстовых данных в рамках академических исследований.

Для решения проблемы упорядочения текстовой информации используется *система управления терминологией* (System Quirk – SQ, http://www.computing.surrey.ac.uk/ SystemQ), которая поддерживает создание и ведение терминов в терминологической базе данных, а также организацию коллекций текстов на компьютере. Система включает в себя целый набор разнообразных инструментов: (*Virtual Corpus*, *KonText*, *Ferret и др.).*

В функции инфраструктуры знаний входит *поддержка коммуникаций* при совместной (коллективной) работе сообщества ученых. Эти функции реализуются набором средств организации *Web-конференций* и *виртуальных распределенных рабочих пространств* (virtual shared workspace), которые в значительной мере облегчают диалог и коммуникации между отдельными лицами и группами Интернет, как, например, в среде CAVE.

Один из базовых компонентов, которыми должны оснащаться региональные центры UK (e-Science Regional Centres), это ***сети доступа*** (access grids), которые поддерживают распределенные заседания специалистов, сессии коллективного творчества, семинары, лектории и тренинги. Сеть доступа основывается на коллекции ресурсов и технологий, которые обеспечивают аудио- и видео-сотрудничество в территориально распределенных сообществах Интернет. Архитектура сети доступа включает широкоформатные мультимедийные дисплеи, презентации и интерактивные среды, а также интерфейсы с программным обеспечением промежуточного слоя сети Grid и сред визуализации.

Инструментальную поддержку функционирования сети доступа осуществляет ***Access Grid®***, текущая версия которого – *Access Grid* 3.1 *beta*1 – базируется на стандартизованных Интернет-технологиях и протоколах. При этом все сетевые соединения защищены и сертифицированы.

3.3. Базовые ресурсы Семантик Веб

Семантический Веб включает множество ресурсов и инструментов, основные из которых приведены ниже.

Языки семантического Веба включают набор ключевых слов, которые позволяют описывать компоненты и утверждения. Это языки описания онтологий, бизнес процессов и данных (OWL, RDF, BPMN, DSL и др.), а также языки задания уникальных идентификаторов ресурсов – Uniform Resource Identifier (URI), Uniform Resource Locator (URL) и др. Уникальные имена элементов Веба образуют пространстве имен. Доступ к различным ресурсам осуществляется с помощью уникальных имен ресурсов URN (Uniform Resource Name).

Онтология Семантик Веб определяет понятия, отношения и ограничения объектов в концептуальной модели предметной области. Многие онтологии могут быть включены в любое приложение путем адаптации к специфическим потребностям этой области. Онтология поддерживает коммуникации между приложениями и является способом описания доменов таких областей, как финансы, медицина, экономика, бизнес приложения и др.

Инструменты могут быть четырех типов: конструирования и развития приложений семантического веба, справочные инструменты для изучения веба, инструменты резонеры, добавляющие механизмы вывода для семантического веба, а также машины правил для расширения семантического веба.

Инструменты конструирования позволяют конструировать или интегрировать семантический веб путем создания или импорта компонентов для онтологии экземпляров. Некоторые GUI-инструменты позволяют просмотр и исследование данных веба, образуя полезный редактор семантического веба.

Справочные инструменты обеспечивают навигацию по семантическому вебу в поисках ответа на вопрос. Существуют различные справочные методы, начиная от простой навигации по графу при поиске и до полного применения языка запросов.

Механизмы резонеры добавляют новые понятия к семантическому вебу для пользователей. Компоненты создают логические дополнения путем классификации. Классификация заполняет структуру класса, позволяя должным образом соотносить понятия и отношения с другими. Имеется несколько типов резонеров, предлагающих различные уровни рассуждений. Резонеры включаются в другие инструменты и каркасы. Они служат рычагами для создания логически правильных вспомогательных утверждений.

Машины на правилах. Это машина вывода на правилах дескрипторной логики. Они добавляют измерение конструкций знаний. Правила позволяют слияние онтологий и других логических задач, в том числе поиск с подсчетом и по строке (count and string searches). Машины на правилах могут рассматриваться как часть общего представления знаний. Каждая машина правил придерживается заданным языком описания правил.

Семантический каркас объединяет перечисленные выше инструменты и позволяет им работать как единое целое. Утверждения, URI, языки, онтологии, и данные экземпляра составляют семантически связанную информацию в семантическом вебе, которыми он манипулирует и создает новые инструменты.

Веб данные отражают смысловое значение данных и интеграции данных для совместного использования путем доступа и обмена богатыми информационными ресурсами WWW, включая использование многих существующих источников данных.

Динамические данные веба позволяют динамичное, во время выполнения, изменение структуры и содержания информации.

Фреймворки Семантического Веба — это набор программных средств, сред, библиотек для создания, манипулирования и обогащения семантического веба. Они помогают создавать выражения, онтологии, веб-приложения и публиковать их в Вебе. Для примера, среда программирования Eclipse IDE, язык программирования Java, набор библиотек для работы с Apache Jena и средств создания онтологии Protege.

RDF язык был утвержден как стандарт W3C (2004). Это система описания сетевых ресурсов, понятых компьютеру. Формат RDF предназначен для сохранения метаданных (данные о данных), описания семантических ресурсов, т.е. служит каркасом для создания отдельных компонентов семантической паутины. Документы в RDF обрабатываются комп'ютером автоматически. RDFS (англ. *RDF Schema*) — это надстройка над RDF, которая позволяет создавать классы и свойства объектов.

OWL (Web Ontology Language) с 2004 г. построен на форматах RDF и RDFS, предназначен для обработки информации в сети. Язык OWL имеет 3 степени детализации, легко масштабируется и согласовывается с современными сетевыми стандартами. В 2008 году был принят новый стандарт OWL 2, включающий описание логики.

SPARQL (Protocol And RDF Query Language) — новый язык запросов для быстрого доступа к данным RDF. Используя обычный протокол и язык SPARQL, программы могут анализировать RDF-описания ресурсов и получать из сети необходимую информацию.

Теория онтологии для представления концептуальных знаний

Концептуальная модель онтологии определяется в виде:

$$O = \langle X, R, F \rangle,$$

где X – конечное множество понятий предметной области или домена;

R – конечное множество отношений между понятиями;

F – конечное множество функций интерпретации.

Такая модель служит связью между людьми, компьютерными системами и средами. Она сохраняет информацию и может подавать необходимые знания о готовых объектах и программах разного назначения, которые находятся в современных библиотеках и репозиториях, реализующих объекты онтологии. Концептуальная модель может фиксировать знания, необходимые для аккредитации соответствующих учебных программ в вузах и сертификации специалистов в масштабе государства и международного сообщества.

Для отображения знаний об определенной предметной области используются концепты. Источниками знаний могут быть терминологические или толковые словари в близких проблемных областях, требования на разработку ПС и другие документы.

Начальным шагом построения онтологии является анализ существенных объектов предметной области, установление отношений между ними и предоставление им уникальных наименований. Для классов объектов выбираются имена, уникальные в границах данного домена.

Источником для поиска объектов ПрО может быть выявление тех понятий в проблемной области, которым соответствуют отдельные задачи по достижению определенных профессиональных целей, которые компъютеризированы и могут быть реализованы при обращении к системе. Таким образом, происходит последовательная декомпозиция сложности проблемы, которую можно выявить в домене предметной или прикладной области:

1) сложная проблема трансформируется в совокупность целей для ее достижения;

2) каждая из целей трансформируется в совокупность возможных примеров использования системы, отраженных в сценарии;

3) сценарии трансформируются в процессе их анализа в совокупность взаимодействующих объектов.

Определенная таким образом цепочка *проблема–цели* или *работы–сценарии – объекты* отображает степень концептуализации для достижения понимания проблемы, последовательного снижения сложности ее частей и повышения уровня формализации моделей последних. Для объектов устанавливаются атрибуты, связи и состояния.

Анализ предметной области для моделирования ее характеристик – первая задача, которая решается при построении семейств систем, предназначенных для функционирования в данной предметной области. Традиционным подходом к ее решению является построение характеристических диаграмм (feature diagram) и DSL языка описания прикладной области. Онтологический подход обуславливает проведение исследований в разных отраслях науки и моделирование концептуальной модели с помощью подхода FDD для его преобразования к языку XML.

Одной из новых средств описания моделей ПрО является язык DSL (Domain Specific Language) и трансформация этих моделей в программный код.

Проектирование в языкt DSL включает анализ ПрО, состоящий в:

1) идентификации прикладной области;

2) сборе знаний о прикладной области;

3) кластеризации полученных знаний в терминах семантических понятий и операций над ними;

4) проектировании проблемного языка типа DSL для описания понятий ПрО и их представления в концептуальной модели.

Следующий шаг – это реализация ПрО следующими процессами:

1) создание библиотек для отображения семантических понятий в модели;

2) создание компиляторов для реализации формального описания программы в языке DSL;

3) обработка на компиляторе описания задач ПрО в DSL и подготовка для их выполнения.

Систематическое моделирование ПрО получило название доменной инженерией (domain engineering) в рамках ProductLine ISE USA.

Для моделирования используются такие методологии: ODM (Organizational Domain Modeling, FODA (Feature-Oriented Domain Analysis, DSSA (Domain Specific Software Architectures). Кроме того, сформировались другие подходы к разработке семейств продуктов: Lucent, Family–Oriented Abstraction, Specification and Translation (FAST) и др. Семейства программ отождествляются с линиями программных продуктов (Product Line/Product Family).

Разработка некоторой онтологии предусматривает глубокий структурный анализ области знаний и включает в себя следующие действия:

1) выделение концептов – базовых понятий данной ПрО;

2) определение "высоты дерева онтологии" – количество уровней абстракции;

3) распределение концептов по уровням;

4) построение связей между концептами;

5) консультации с разными специалистами для исключения противоречий и неточностей.

Процесс построения онтологии является итеративным и допускает выбор целей и задач прикладной области для составления онтологии с помощью таких действий:

1) фиксация знаний о ПрО, определение основных понятий и их отношений м выявления непротиворечивых определений для каждого базового понятия;

2) выбор или разработку специального языка для представления онтологии;

3) задание фиксированной концептуализации на выбранном языке представления знаний;

4) обеспечение возможности использования знаний ПрО;

5) отделение знаний о прикладной области и оперативных знаниях.

Характеристика языков описания онтологии

К языкам спецификации онтологии относятся традиционный Ontolingua, CycL, языки, основанные на дескриптивной логике - LOOM, и языки, основанные на фреймах – OKBC, OCML, Flogic.

Более современными языками, основанными на веб-стандартах, являются XOL, SHOE (или UPML), RDF(S), DAML, OIL, OWL.

В целом, различие между традиционными и Веб-языками описания онтологий заключается в выразительных возможностях описания моделей ПрО и в некоторых механизмах логического вывода понятий.

Язык XML (Extensible Markup Language) фактически стал стандартом разметки данных для их сохранения и обмена информацией между разными пользователями. XML активно используется в самых разных областях научной и коммерческой деятельности. Однако XML имеет низкий уровень описания ресурсов и необходимые средства автоматической трансформации концептуальных моделей ПрО и XML-схем, пригодных для разработки приложений.

Основным языком моделирования ПрО является UML и на его основе можно сгенерировать описание классов и заготовок программного кода для выбранного ЯП и платформ его реализации.

Альтернативным подходом UML является онтологический подход *Ontology-Driven Software Development,* который позволяет описывать классы, отображающие базовые понятия ПрО в модели и генерация кодов программ с "выполняемыми" артефактами.

Средства описания онтологий

К общим средствам описания онтологий любых предметных областей относятся: классы, аксиомы, *слоты*, как свойства классов и фасеты, описывающие конкретные типы и возможные диапазоны значений.

В аксиомах определяются дополнительные правила определения базовых понятий. Абстрактные классы являются контейнерами конкретных классов и могут содержать *абстрактные* атрибуты. Атрибуты понятия прикладных областей называются *слотами*.

Конкретные классы содержат слоты, с помощью которых могут быть определены значения (экземпляры атрибутов).

Слоты **(**например, в Protege) описывают свойства классов и экземпляров. Согласно фреймовой модели представления знаний, слот – это фрейм. Слоты определяются независимо от любого класса, и один и тот же слот может принадлежать разным классам.

Фасеты позволяют вводить ограничение на типы и диапазоны значений экземпляров (значений атрибутов), подобные соответствующим понятиям XML-схемы. Кардинальность слота определяется количеством значений слота, ограничениями для типов значений (например, целое, символьное и т.п.) экземпляров класса и граничных значений (min, max). Фасеты определяют ограничение на присоединение слота к фрейму класса.

Слоты-образцы **(**template slot) *и собственные* (own) *слоты*. *Слот* можно присоединить к фрейму (класса) одним из двух способов: как *слот-образец* или как собственный слот. Собственный слот, присоединенный к фрейму, описывает свойства объекта, представленного фреймом. Классы также могут иметь собственные слоты. Например, документация класса является собственным слотом, присоединенным к классу, поскольку описывает сам класс, а не экземпляры класса.

На сегодня функционирует в Интернете ряд онтологий: Sensus с общими знаниями и понятиями естественного английского языка (более 70000 терминов и их дефиниций); онтология понятий электронной коммерции; глобальная онтология продуктов и услуг (OOH); коммерческие онтологии SCTG, RosettaNet транспортных потоков товаров и т.п. Кроме того действует медицинские онтологии (Galen для определения клинического состояния; UMLS Национальной медицинской библиотеки США; ON9 для аттестации медицинских систем за общими параметрами, а также инженерные, химические онтологии, и т.п. К средствам представления знаний относится также общероссийский Веб–портал математических ресурсов, всемирная информационная математическая библиотека MathLab, Ret и др.

Таким образом, в основе представления любой системы знаний в виде онтологии лежит понятийная база, совокупность концептов (понятий) и отношений между ними. Следующим шагом нормализации знаний является зафиксированная классификация понятий в виде тезауруса предметной области. Процесс построения модели ПрО, ориентированной на ее понимание человеком, называют *концептуальным моделированиям.*

Каждая ПрО или *домен* имеет присущую ему систему понятий, систему "умалчивания" того, что должен считаться "общеизвестным" в рамках своего домена, собственные характерные свойства (атрибуты), отношения и правила поведения. Фактически концептуальная модель как посредник между заказчиком и разработчиком. Степень ее формализации должна быть достаточной, чтоб обеспечивать точность и однозначность толкования разными носителями интересов в разработке и доступности для понимания.

По форме представления онтологии домена используется концептуальная модель, которая отображает систему понятий с характерными свойствами (атрибутами), отношениями и правилами поведения. Такая модель служит коммуникации (между людьми, между компьютерными системами) и сохранению информации в компьютерной среде

Онтология задает соглашение об общем использовании понятий, которое включает средства представления предметных знаний и договоренностей о методах рассуждений.

Для *описания онтологии* используются язык OWL (Web Ontology Language) со спектром семантических языков разметки и словарю RDF для доступа и обмена онтологическими знаниями в Интернете. Описание онтологии в языке OWL – это последовательность аксиом и фактов, информация о классах, свойства и ресурсов с идентификатором (ID*) и веб- документы, которые импортируются через* URI и могут ссылаться на XML–схему в форме: <DatatypeID>:: = < URI ссылка >.

Здесь факты задают информацию в форме классов и свойств со значением ресурса, а аксиомы используются для сопоставления класса и свойств ID с частичными или полными спецификациями характеристик, давая логическую информацию о классах и свойствах. Каждая аксиома класса содержит совокупность более общих классов и совокупность локальных ограничений свойства. Класс является эквивалентом, подмножеством или пересечением более общих классов с ограничениями.

В OWL аксиома класса содержит совокупность описаний, которые могут иметь вид обобщенных классов, ограничений, наборов ресурсов, булевых комбинаций описаний. При этом компания OMG предложила для управления, взаимодействия и семантики доменов метамодель ODM (Ontology Definition Metamodel).

Понятийная база домена должна определять не только терминологию в процессе анализа требований, но и отношения между понятиями и их интерпретацией. При этом семантика понятий может быть охарактеризована совокупностью общих существенных признаков тех явлений и предметов, которые входят в это понятие (образовывая их *объем)*.

Отношения между понятиями задают:

1) обобщение существенных признаков понятия, давая расширение круга охваченных понятиями объектов и объема;

2) *конкретизацию* понятий путем добавления существенных признаков, благодаря чему содержание понятия расширяется, а объем понятия сужается;

3) *агрегацию,* т.е. объединение ряда понятий в новое понятие, существенные признаки которого при этом могут образовывать сумму признаков;

4) *ассоциацию* как наиболее общее отношение, которое утверждает наличие связи между понятиями, не уточняя зависимости от содержания и объема.

Для отдельных прикладных областей могут использоваться специфические для них отношения и онтология позволяет пользователю находится в пространстве предопределенных понятий, содержание которых зафиксировано.

Онтология – инструмент стандартизации мировой системы знаний

В области онтологии появились международные профильные стандарты терминов и определений в ряде международных органов, ответственных за их ведение и существование - ISO, WWW, IEEE и др.

В основе представления любой системы знаний лежит понятийная база, совокупность концептов (понятий) и отношений между ними, классификация понятий и их таксономия в виде тезаурусов, а также методы создания профильных онтологий.

В мировой практике уже имеется ряд действующих профессиональных онтологий:

Sensus базируется на понятиях естественного (английского) языка и содержит более чем 70 000 терминов и их дефиниций;

онтология электронной коммерции;

глобальная онтология продуктов и услуг (стандарт организации Объединенных наций);

коммерческая онтология SCTG транспортных потоков товаров компаний;

онтология e–class поддерживает обмен данными и материалами между продавцами и пользователями крупных компаний Германии;

онтология товаров RosettaNet поддерживают 400 коммерческих компаний.

Существует и широко используется круг медицинских онтологий (Galen для определения клинической картины; UMLS для Национальной медицинской библиотеки Соединенных Штатов; ON9 для аттестации известных медицинских систем по определенным параметрам), а также инженерные онтологии; деятельности предприятий; химические, биологические онтологии и др.

К основным направлениям создания онтологий в Интернет относятся:

– разработка онтологических концептуальных моделей, методов и средств доступа к мировым информационным ресурсам;

– создание моделей описания готовых ресурсов разработки как повторно используемых знаний;

– создание библиотек онтологии с профилями знаний;

– разработка методов и средств создания комплекса инструментов ведения онтологии;

– разработка требований к системе для реализации свойств домена и формирования базы знанидля решения задач прикладной области;

– разработка методических и учебных материалов по применению Семантического Веба для описания профессиональных знаний.

Жизненный цикл онтологии предметной области проводится следующими этапами:

– первичное накопление общеизвестных понятий предметной области в толковом словаре последней;

– установление отношений между выделенными понятиями;

– накопление разработок отдельных программных систем или других КПИ для выбранной предметной области в репозитории или библиотеке;

– выявление общности накопленных разработок, согласование их с толковым словарем и создание концептуальной модели домена;

–анализ требований к новой разработке под управлением онтологии домена;

– модификация онтологии и концептуальной модели домена недостающими программными элементами для новой разработки системы или ее варианта;

– верификация и валидация концептуальной модели и готовых КПИ;

– сборка (конфигурация) готовых ресурсов, КПИ повторного использования;

– оценка надежности и качества собранного варианта продукта.

3.4. Характеристика сервисных и семантических ресурсов Интнернет

В 1992 году при правительстве России был создан Научно-технический Совет «Информатизация России» из 30 ведущих специалистов (Липаев В.В., Иванников В.П, Левин В.К. и др.) институтов РАН и других научно-исследовательских организаций России. В нем определены новые фундаментальные и технические основы информатизации России («Научно-технические основы информатизации России», Москва, 1992.-151с):

1. Экономические и социальные задачи информатизации.

2. Технико-экономическое развитие рыночной экономики информатизации.

3. Научно-техническое обеспечение информатизации России:

– фундаментальные и методологические исследования и разработки ИТ и систем;

– технология и программная инженерия разработки систем;

– обеспечение защиты информации;

– стандартизация программно-технических средств создания систем;

– концепция отбора конкурсных проектов (РФФИ);

– поддержка информатизации системой высшего образования.

4. Развитие инфраструктуры информатизации России.

5. Системный проект информатизации России.

К основным фундаментальным, прикладным и методологическим основам технологии программных систем (В.П.Иванникова, Липаева В.В.) отнесены:

– алгебраические и логические теории формальных систем;

– анализ параллельных и распределенных систем, языки, логика, модели;

– формальные спецификации и верификация сложных систем;

– технология программирования;

– парадигмы программирования;

– формализмы представления знаний;

– модели человеко-машинных интерфейсов;

– визуализация в динамических системах и виртуальной реальности.

Системное программирование, должно обеспечивать интерфейс с аппаратурой вычислительных средств и включать ЯП, компиляторы, ОС, системы управления данными и знаниями, технологические системы поддержки разработки больших программных комплексов, открытые системы, которые будут базироваться на международных стандартах и обеспечивать совместную работу с другими компонентами.

К задачам технология программирования отнесены:

– определение основ технологии и инженерии, как самостоятельной и прибыльной сферы деятельности по созданию прикладных областей;

– методы повышения производительности систем на основе сборки готовых ресурсов и повторных компонентов КПИ;

– методы обеспечения качества создаваемых систем на основе ЖЦ процессов разработки и стандартов гарантии качества, аттестации и сертификации программных продуктов;

– перспективные технологии на базе искусственного интеллекта и методов автоматической генерации машинных программ с ЯП.

Эти фундаментальные задачи системного программирования положены в основу образования при РАН Института Системного программирования (ИСП) 21.02 1994 г. Иванниковым В.П. Созданный ИСП в течение 25 лет ИСП выполнял многие поставленные задачи по линии РАН и Проектов РФФИ.

В рамках проекта РФФИ № 16-01-00352 «Моделирование программных и операционных систем» (за подписью директора Иванникова В.П.) исследованы и разработаны ряд фундаментальных задач, включая моделирование сложных систем, парадигмы программирования и обеспечения безопасности, надежности, защиты и качества, а также задачи интеллектуализации и онтологизации предметных разных областей знаний (биология, химия, физика, математика и др.).

3.5. Моделирования технических и программных систем в рамках проекта РФФИ

Результаты работ по **проекту РФФИ 2016-2018:**

1). Анализ отечественных теоретических основ ПО программ и систем с целью поднятия теоретического уровня подходов к разработке и моделированию сложных систем в нашей стране. Определен современный подход к моделированию прикладных систем из объектов с помощью логико-математического аппарата с использованием графового

представления структур систем и задания вариабельных моделей MF (Feature Model) для формирования вариантов изменяемых систем методом сборки модулей и компонентов повторного использования (КПИ) в классе программных, информационных и ОС для обеспечения взаимодействия КПИ и приложений в современных средах.

2). Отработаны процессы верификации моделей MF и создаваемых Msys систем, конфигурирования КПИ и интерфейсов, тестирования отдельных системных ресурсов, КПИ и сбора сведений об ошибках, дефектах и отказах для обеспечения гарантии работоспособности и оценки качества моделируемых систем модели MF.

3). Разработан подход к преобразованию типов данных компонентов, передаваемых друг другу через интерфейсы для обеспечения взаимодействия отдельных КПИ, систем и семейств систем в современных средах.

4). Отработан компонентный подход к решению поставленной в данном проекте задачи создания Веб-систем из сервисов и сервисных компонентов SOA, SCA, SCM Интернет с использованием новых языков (C++, C#, Basic, Python, Java и др.) W3C и обеспечения безопасности и качества прикладных систем.

5). Отработан метод сборки компонентов, идентично методу сборки разноязыковых модулей, реализованный в системе АПРОП и описанный выше в п. 2.4. Метод сборки с интерфейсными примитивами преобразования разнородных данных ЯП опубликован. Эта система была внедрена в 52 организации страны. По словам А.П.Ершова реализовано «сборочное программирование, которое обеспечивает построение уже существующих (проверенных на правильность) готовых отдельных фрагментов программ (типа reuses) в сложную структуру».

6). Разработан вариант ОС Linux путем извлечения (Variability Mining) системных и функциональных модулей из ядра, построения моделей MF, M_{OS} Linux и генерации по ним экспериментального вариантов ядра ОС для возможного применения в биологии, экономике, медицины и др. Эти модели верифицированы, а КПИ и их интерфейсы протестированы. Проведена конфигурационная сборка (Kconfig), как развитие сборки Link.

7). Появление новых ЯП после после 1992. Это C, C++, Basic, Java, Ruby, Python и др. Они используются для спецификации программ и систем, а интерфейсов в языках IDL, API, WSDL и др. Разработанный ранее метод сборки продолжал использоваться для новых ЯП с помощью операций сборки link в системах IBM, make в BSD (1996) и Kconfig, в Grid, SPAROL (2000) и др. Оператор make ориентирован на ПО систем трансляторов и собирает исполняемые модули из библиотеки Filemake, задавая порядок связей модулей в среде ОС Linux, Unix и др. (см. рис. 3-5). Разработан стандарт ISO/IEC 828 Configuration - 2012 для сборки программных элементов в новых ЯП с помощью оператор Kconfig Configuration. Этот оператор прошел апробацию в подсистеме ETICS Европейского проекта Grid (http://www.semanticgrid.org/documents/semgrid2004/).

Модели вариабельности. В период информатизации быстрыми темпами развивалась *индустрия* ПО. Появились продуктовые линии - ProductLine/ProductFamily (K.Pochl), 2004, технология мультисборки К.Чернетски и У.Эзенакера (2005), использующие новые модели систем *Msys* и MF. Эти модели и метод сборки стали основой реализации изменяемых систем и основой индустрии разных вариантных продуктов.

В русле концепции моделирования систем и семейств систем проблемных областей с использованием моделей *Msys* и MF проводились исследования и обсуждение на Международных конференциях (VAMOS, SPRL, Reusebility, ETAPS и др.). На основе этих моделей и ООП создан оригинальный объектно-компонентный графовый метод моделирования систем из функциональных, системных, сервисных и интерфейсных элементов и отработан в:

http://0x1.tv/20181122AF

http://0x1.tv/20191206AB

http://0x1.tv/20181122AF

https://videonauka.ru/stati/30-metodika-prepodavaniya-tekhnicheskikh-distsiplin/237-informatika-i-evm-70-analiz-i-aspekty-razvitiya

https://videonauka.ru/stati/30-metodika-prepodavaniya-tekhnicheskikh-distsiplin/238-analiz-metodov-otsenki-nadezhnosti-oborudovaniya-i-sistem-i-praktika-primeneniya-etikh-metodov

E.M. Lavrischeva, A.K. Petrenko. Technology of Assembly of intellectual and information resources Semantic Web Internet, http://keldysh.ru/abrau/2019/theses/93.pdf doi:10.20948/abrau-2019-93.

E. M. Lavrischeva. The theory graph modeling systems from quality modules of the application, p.235-247, http://ceur-ws.org/Vol-2514/

E. M. Lavrischeva. The Theory Graph Modeling and Programming Paradigms of Systems from Modules to the Application Areas, Russia Journal. Computer and Information Sciences – vol.12, №4 2019 (ISSN: 1913-8989, E-ISSN: 1913-8997).

E.M. Lavrischeva. The Theory Graph Modeling and Programming Paradigm Systems from Modules to the Application Areas. DOI: https://doi.org/10.14738/tmlai.74.6782, Vol 12 No 6 (2019): Transactions on Machine Learning and Artificial Intelligence

3.6 Интеллектуальные и информационные ресурсы Интернет

Технология сборки интеллектуальных и информационных ресурсов – это процессы сборки ресурсов (модульных элементов в разных ЯП), структур данных, процедур, классов, компонент и сервисов.

Ресурсы разрабатываются для разных прикладных областей знаний (математика, медицина, биология, генетика и др.). Они накоплены в библиотеках, хранилищах Интернет и в общесистемных средах (IBM, MS.Net, Unux, Intel и др.).

К хранилищам относятся библиотеки, репозитории, базы данных (БД). Библиотеки процедур, Базы знаний, Reusability Libraries и др. Функциональные элементы типа reuses создают высокоинтеллектуальные специалисты в области информатики, математики, биологии и др. (например, MatLab, Demral и др.). Каждый готовый reuse при размещении в библиотеки общего использования проверяется на правильность и качество реализации функций области знаний.

Интеллектуальные ресурсы (ИНР) – это компоненты повторного использования (КПИ и reuses), отображающие знания специалистов в разных областях знаний. Любая функция ПрО описывается в ЯП в виде модуля (объекта, компонента, сервиса и др.) и может взаимодействовать через оператор сборки link и операции вызова CALL/RPC/RMI в текстах программ ЯП, в параметрах которых задаются данные в IDL, API, ABI, WSDL и др.

КПИ (компоненты повторного использования) обрабатывают данные, которые относятся к фундаментальным (FDT) и общим типам данных (GDT) стандарта ISO/IEC 11404, а также могут оперировать с большими объемами данных (Big Data). ИНР взаимодействуют с модулями, объектами или сервис-компонентами Интернет через интерфейс, задаваемыйи в языках IDL, API, ABI, WSDL и др.

Reuse — это готовый интеллектуальный программный объект, компонент, сервис, реализующий алгоритм функции в некоторой предметной области знаний. Он специфицируется в стандартном языке WSDL и может многократно использоваться, если удовлетворяет функциональным требованиям отдельных предметных областей. КПИ как Reuses имеет код (implementation) с интерфейсом (interface) и схему развертки (deployment) для их выполнения. КПИ, Reuses могут иметь общие переменные, которые образуют **классы** или **множества**. Внешние общие переменные и методы класса задаются в интерфейсе экземпляров класса.

Информационные ресурсы (ИР, IR) – это данные разного объема, в том числе и Big Data, представлены в Базах Данных, файлах, каталог**ах,** таблицах, документах и т.п. ИР

обрабатываются сервисными, системными, клиентскими и серверными службами Интернет и помещаются в Хранилищах данных, БД малых и больших объемов со структурированными и неструктурированными данными.

Интерфейс — это спецификатор информационной части ИНР — КПИ, компонента, reuses для обращения к другим ресурсам и обмена данными между ними. Интерфейс содержит вызов метода или функции (RPC/RMI) с параметрами, которые управляют внешними переменными экземпляра класса (выборка значения *get-метод,* присвоение значения *set-метод*, *Home-интерфейс* в языке JAVA и др.) при их взаимодействии. Интерфейс описывается в языке WSDL и в общем случае содержит:

- *название* функции (метода) и *ID* — идентификатор ресурса;
- *описание (спецификацию)* функции средствами ЯП;
- *параметры* (входные и выходные) для передачи данных другим КПИ;
- *язык* описания КПИ (C, C++, Java, Python, Ruby и др.);
- *необязательные атрибуты* (дата, состояние, версия, право доступа, автор, срок использования и т.п.).

Среда программирования (Eclipse, Protege и др.) позволяет автоматически создавать описания ресурсов на основе классов языка JAVA. Определены следующие основные типы данных:

1) строки (xsd:string);
2) целые числа (xsd:int, xsd:long, xsd:short, xsd:integer, xsd:decimal), числа с плавающей запятой (xsd:float, xsd:double);
3) логический тип (xsd:boolean);
4) последовательность байт (xsd:base64Binary, xsd:hexBinary);
5) дата и время (xsd:time, xsd:date, xsd:g);
6) тип объекта (xsd:anySimpleType).

В качестве переменных могут использоваться множества, последовательности, включающие фиксированное количество переменных простых типов.

Типичный WSDL-файл имеет следующую структуру.

```
<wsdl:definitions [.]>
<!-- Декларация типов, которые используются в сервисе -->
<wsdl:types>
<element name="someMethod">
<complexType>
<sequence>
<element name="arg0" type="xsd:double"/>
<element name="arg1" type="xsd:boolean"/>
</sequence>
</complexType>
</element>
<element name="someMethodResponse">
<complexType>
</wsdl: types> …
```

Данное WSDL-описание задает веб-сервис MyService с единственным методом String someMethod (double arg0, boolean arg1). На его основе можно сгенерировать два типа данных, которые соответствуют входным и выходным параметрам метода. Эти типы применяются в описаниях someMethodRequest и someMethodResponse — входного и выходного сообщений для операции someMethod.

Операции декларируются в описании интерфейса сервиса (декларация wsdl:portType) и затем дается описание привязки сервиса к протоколу SOAP (Simple Object Access Protocol) через декларацию wsdl:binding. При этом устанавливается вызов <wsdlsoap:body use="literal"/> с параметрами, заданными в методе класса. В конце WSDL-файла приводится

декларация веб-сервиса (<wsdl:service>), в которой содержится информация о расположении <параметр location>.

3.7. Сервисные и информационные ресурсы при сборке Веб-систем

Web-сервисы основаны на открытых стандартах Интернет, которые широко поддерживаются на платформах Unix, Intel, Windows, IBM, Linux и др. и задаются PHP, ASP, JSP-скрипт, JavaBeans и др.

Формат запросов к Web-сервисам определяет протокол SOAP, который задается в XML и отправляется через HTPP к Web-серверу. IBM, Microsoft и Universal Description, Discovery and Integration (UDDI) способствуют созданию общего каталога Web-сервисов. SOAP, XML и UDDI обеспечивают также надежность функционирования приложений из Web-сервисов и сервис- компонентов.

К средствам создания прикладных систем в Интернет относятся сервис-ориентированная архитектура SOA (Service Oriented Architecture) и сервисно-компонентная архитектура SCA (Service-Component Architecture).

SOA задает функциональность (Functions) и качество сервисов (Quality service) на уровнях:

транспортный (transport layer) для обмена внешними данными на коммуникационном уровне (service communication layer);

описания сервиса и интерфейса (service description layer) с обеспечением безопасности и защиты (авторизации, аутентификации);

операций публикации, поиска и вызова сервисов через реестр сервисов.

В Интернет имеется реестр сервисов, который содержит описание сервисов в языках Семантик Веб (SOA, SCA, SMC и др.) и обеспечивает регистрацию, поиск и вызов сервиса по запросам поставщиков и потребителей сервисов.

Рассмотрим сервисы SOA и SCA. Операции над сервисами SOA такие:

1) публикация сервиса через вызов сервиса и его интерфейс;
2) поиск сервиса с помощью протокола SOAP;
3) связь UDDI с моделями CORBA, DBMS, JNET и т. п.;
4) запрос к провайдеру за опубликованными интерфейсами сервиса.

Сервис SCA дает доступ к сервис-компонентам, которые упаковываются в модуль сервиса в среде WebSphere, эквивалентного EAR-файлу J2EE. Сервисы SCA интегрируются через интерфейс JAVA и реализуются в виде классов JAVA™. В модели SCA объекты данных представлены в JAVA common.sdo.DataObject, который включает в себя метод определения свойств данных. WebSphere Integration Developer платформы Eclipse 3.0 позволяет композировать (собирать) сервисы SCA и сервисные интерфейсы.

Для вызова внешнего сервиса используется JMS, Enterprise JavaBeans или готовые сервисы. Доступ к БД системы проводится через аппарат ссылок. Веб-система собирается из сервисных компонентов, описанных в языках Python, JAVA, BPEL, OWL и интерфейсов с помощью операции конфигурирование Kconfig SAP [18, 19]. Библиотека SCA содержит набор reuses композитных сетевых сервис-компонентов и гетерогенных данных (http://www.ibm.com/developerworks/websphere/techjournal).

Эти сервисы могут создаваться в среде Java Enterprise Edition, которая позволяет проводить:

- динамическую генерацию серверных страниц (Java Server Pages);
- определение КПИ в виде Enterprise Java Beans;
- преобразование параметров КПИ к формату представления других сред;

– обмен сообщениями (Java Message Queue) со службами JMS (Java Message Service).

В SCA могут создаваться новые сервисные компоненты для проблемных областей знаний, а также объекты данных (Service Data Objects — SDO) и интерфейсы, обеспечивающие передачу данных другим КПИ. Для вызова внешнего сервиса используется JMS, Enterprise JavaBeans или готовые сервисы. Доступ к БД из системы (предприятия/фирмы) проводится через аппарат ссылок. Создаваемая Веб-система собирается из сервисных компонентов, описанных в языках Python, Java, BPEL, OWL и интерфейсов с помощью операцией конфигурирования Kconfig SAP.

При работе веб-сервисов с даннями из класса Big Data используются такие средства Интернет: IBM WSDK+WebSphere; Apach Axis+Tomcat; Apach Axis+ Classfish; Microsoft.Net+IIS и др.

К основным **системным ресурсам** Интернет относится клиент-серверная архитектура сети Интернет, которая является трехуровневой: клиент, сервер приложений (систем) и сервер БД.

На уровень *клиента* выносятся простые сервисные ресурсы веб-систем: интерфейс, операции с данными, алгоритмы шифрования, взаимодействия и безопасности для передачи серверу приложений и т.п. Клиентская часть отвечает за интерфейс и обработку приложений Интернет, которые записаны на ЯП (C, C++, Python, Ruby, Java, Basic и др.) и выполняют прикладные функции, обработку возникающих аварийных ситуаций, контроль безопасности и качества и др.

Сервер БД запускается с сервера приложений и выполняет обслуживание БД с обеспечением целостности, сохранности данных при доступе клиента к информации БД. Для работы с Big Data выполняются задачи анализа данных большого объема, а также манипулирования данными с малой и большой нагрузкой.

Клиенту соответствует Интернет-браузер (Chrome, Firefox, MS Internet Explorer, Safari, Opera и др.). Он посылает сообщения серверу через интерфейс следующего вида: $WebAppInterface = \{Request^p\}$, где $Request^p$ — р-й запрос.

3.8 Обеспечение безопасности, надежности и качества ПТС

ИСП РАН провел конференция OS Day. – Надежность, 17-18 мая 2018. На ней сделан доклад Лаврищевой Е.М. и Пакулиным Н.В. по теме «Анализ методов оценки надежности оборудования и систем» (http://0x1.tv/20180517F). Доклад доработан до статьи с участием Рыжова А.Г., Зеленова С.В. и опубликован. В нем главное внимание уделяется вопросам обеспечения работоспособности таких средств, особенно если они используются в авиации, космосе и др. критических областях. Основой работоспособности системы является безопасность.

Термин безопасность относится ко многим сферам деятельности человека, включая массовую информацию, государственное управление, создание разных видов продуктов, включая программные продукты и среды их функционирования.

В книге «Функциональная безопасность программных средств» В.В.Липаев рассмотрел применение и использование ЭВМ в жизни общества, а также программные и информационные средства. Под *функциональной безопасностью* продукции, процессов и систем понимается их работоспособность при обработке информации и управляющих воздействий в соответствии с требованиями к ним. Аварии, катастрофы и чрезвычайные ситуации, которые возникают в авиации, на атомных станциях, транспорте являются следствием недостаточной безопасности функционирования программных средств. Это связано с некоторыми недоработками отдельных элементов таких систем, неадекватностью воздействия окружающей среды или действиями злоумышленников. В результате приноситься большой финансовый ущерб, а иногда и жертвы.

Исследования показывают, что нестабильность функционирования систем определяется следующими факторами:

– техническими отказами внешней аппаратуры и искаженной информации от внешней среды;

– случайными сбоями и разрушениями отдельных компонентов систем;

– дефектами и ошибками программ обработки информации и данных;

– недостатками в средствах обнаружения отказов и оперативного восстановления работоспособности программ и данных.

Понятия и характеристики безопасности систем связаны с понятием надежности. В показателях надежности учитываются опасные отказы, которые могут использоваться при оценке характеристик надежности и качества систем. Оценка надежности и качества проводиться согласно стандарта ISO 9126. В этом стандарте определены характеристики качества систем:

– внешние характеристики задаются в требованиях к системе и задаются метриками реализации продукта в заданной среде;

– внутренние характеристики задают принцип реализации продукта на этаках жизненного цикла системы;

– эксплуатационные характеристики качества отражают результат достижения заданного требования к системе с учетом затрат.

Оценка надежности систем определяется согласно внешних, внутренних и эксплуатационных характеристик, которые сравниваются с требованиями заказчика на заданную систему и оцениваются с учетом стандартов жизненного цикла ISO 12207 и ISO 15288 -2006.

Надежность систем сформировалась как самостоятельная теоретическая и прикладная наука, способствующая определению одного из показателей качества систем (функциональность, надежность, завершенность и др.). Методы оценки надежности систем позволяют прогнозировать, измерять и оценивать качество продукта, доводя ошибки, дефекты и интенсивность отказов в компонентах ПТС к минимуму на процессах ЖЦ. Главными источниками информации об ошибочных ситуациях для оценки надежности являются процессы ЖЦ (верификация, тестирование, Конфигурация, эксплуатация и испытание). К таким стандартам ЖЦ (ISO/IEC 15846-1998, 15939:2002 и др.).

Возникновение ошибок и отказов, как правило, является случайным процессом и зависит от времени появления или частоты их появления. В связи с этим модели надежности основываются на нахождении случайной величины в системе, числом и интенсивностью возникновения отказов в системе. Одним из подходов к исследованию надежности на основе отказов является классическая теория вероятностей, согласно которой отказы в системе считаются случайными и зависят от дефектов, внесенных при разработке системы. На процессах ЖЦ набирается статистика отказов, их интенсивность, проявляются дефекты в компонентах. Они оцениваются при испытании системы и влияют на получение коэффициента надежности и соответственно качество системы.

В безопасности учитываются только те отказы, которые могут привести к катастрофическим последствиям и ущербам (например, пожар, взрыв, разрушение здания и др.). Оценка надежности и безопасности функционирования ПТС измеряется по метрикам (внешние, внутренние, эксплуатационные). Они сравниваются с требованиями заказчика на систему и используются при сертификации готовой системы. Для оценки надежности и функциональной безопасности может использоваться стандарт ISO 15288 -2006.

Задачи обеспечения надежности ПТС

В рамках данного проекта РФФИ рассматриваются задачи обеспечения надежности, безопасности и качества ПТС.

К наиболее распространенным методам обеспечения безотказной работы относятся модели Джелинского-Моранды, Нельсона, Мусы, Вейса и др. Сформировалась классификация моделей надежности: прогнозирующие, измерительные и аналитические.

Прогнозирующие (статические) модели Холстеда, Маккейба и др. основываются на методе поиска ошибок и дефектов при анализе текстов исходных программ и измерении технических характеристик программы.

Измерительные модели предназначены для измерения надежности ПО, работающего с заданной внешней средой. Эти модели учитывают интервалы между отказами, которые распределяются по экспоненциальному закону, а интенсивность отказов пропорциональна числу обнаруженных ошибок.

Оценочные модели основываются на серии тестовых прогонов на этапах тестирования и верификации программных средств. В тестовой среде определяется вероятность отказа программ в процессе тестировании или выполнения. Эти типы моделей могут применяться на процессах ЖЦ стандарта ISO/IC 12207-2007, используя количество и интенсивность отказов.

Приведенные модели основываются на результатах верификации, тестирования компонентов и систем, включая вероятность безотказной работы в течение заданного времени и числа оставшихся ошибок и дефектов.

3.8.1. Процессы ЖЦ для разработки ПТС

Многие современные системы, работающие в реальном времени (авио, космические, и др.) требуют высокой надежности (недопустимость ошибок, отказов, аварий и др.), которая в значительной степени зависит от оставшихся и не устраненных ошибках в процессе разработки ПТС.

На надежность ПО также угрозы, приводящие к неблагоприятным последствиям, риску нарушения безопасности и способности компонентов системы сохранять устойчивость в процессе ее эксплуатации. При этом риск уменьшает надежность и безопасность системы, если проявляются внешние угрозы.

В связи со сказанным особую важность приобретает задача применения ЖЦ при создании качественных компонентов, включая процессы:

– системный анализ и разработка требований к системе и характеристик качества функционирования;

– проектирование архитектуры (модели) системы и отдельных компонентов;

– реализация компонентов и их конфигурация в систему;

– тестирование компонентов и системы из компонентов;

– испытание системы;

– сопровождение системы.

На процессе анализа и разработки требований определяются функции системы и спецификация основных требований к системе с заданием метрик для оценки надежности, в терминах интенсивности отказов или вероятности безотказного функционирования.

Разработчики системы формируют:

– приоритеты функций системы по критерию важности их реализации;

– параметры среды функционирования и интенсивности использования функций и их отказов;

– входные и выходные данные для каждого функционального компонента системы;

– категории отказов и их интенсивность при выполнении функций в заданное время.

На процессе проектирования определяются:

– размеры информационной и алгоритмической сложности всех типов проектируемых компонентов;

– виды дефектов, свойственные всем типам компонентов системы;

– стратегии функционального тестирования компонентов по принципу «черного ящика» с помощью тестов для выявления дефектов и интеграционного тестирования.

Для достижения надежного продукта проводится анализ:

– вариантов архитектуры системы на соответствие требованиям к надежности;

– анализ рисков, отказов и деревьев ошибок для критических компонентов с целью обеспечения отказоустойчивости и восстанавливаемости системы;

– прогнозирование показателей размера системы, чувствительности к ошибкам, степени тестируемости, оценки риска и сложности системы;

– прогнозирование количества и плотности дефектов для модели измерения надежности.

На процессах реализации и тестирования системы проектные спецификации функций компонентов системы переводятся в коды и подготавливаются наборы тестов для автономного и комплексного тестирования компонентов и систему. При проведении автономного тестирования обеспечение надежности состоит в предупреждении появления дефектов в компонентах и создании эффективных методов защиты от них. Все последующие процессы разработки (например, верификация) не могут обеспечить надежность систем, а лишь способствуют повышению уровня надежности за счет обнаружения ошибок с помощью тестов различных категорий и их исправления

На процессе испытаний проводится системное тестирование для установления соответствия внешних спецификаций функций целям системы. Испытание проводится в реальной среде функционирования или на испытательном стенде с помощью наборов данных для имитации функций компонентов. При подготовке к испытаниям изучается "история" тестирования (таблица сведений об ошибках и отказах) на процессах ЖЦ, использование ранее разработанных тестов для составления специальных тестов испытаний с целью:

– управления ростом надежности при неоднократном исправлении и регрессионном тестировании ПТС;

– принятия решения о степени готовности системы и возможности ее передачи в эксплуатацию;

– оценка надежности по результатам системного тестирования и испытаний по соответствующим моделям надежности, подходящих для заданных целей системы.

На процессе сопровождения оценка надежности ПС проводится путем:

– протоколирования отказов в ходе работы системы, измерения надежности функционирования и использования результатов измерений для определения потерь (снижения) надежности в период времени эксплуатации;

– анализ частоты и серьезности отказов для определения порядка их устранения;

– оценка влияния функционирования системы на надежность в условиях усовершенствования технологии и применения новых инструментов разработки и тестирования.

3.8.2. Модели и методы качества ПТС и их оценивание по стандартам

Под качеством систем понимается совокупность свойств, обуславливающих ее пригодность к использованию по назначению при заданных требованиях и условиях функционирования.

Качество систем характеризует его пригодность к использованию по назначению. Для разных типов ПТС в классе задач прикладной области предварительно разрабатывается набор показателей и устанавливаются их базовые значения, которые должны быть достигнуты при разработке отдельных прикладных программ и компонентов системы.

Модель качества $M_{кач}$ в стандарте (ISO/IEC 9000 (1-4) определяет формализованное представление характеристик качества, их свойств и способы оценки на процессах ЖЦ, представленных выше.

Формально модель качества имеет такой вид:

$M_{кач} = \{Q, A, M, W\}$,

где $Q = \{q_1, q_2, \ldots, q_i\}_{i=1,\ldots,6}$, – множество характеристик качества *(Quality – Q)*;

$A = \{a_1, a_2, \ldots, a_j\}_{j=1,\ldots,J}$, – множество атрибутов (*Attributes – A*), каждый из которых фиксирует отдельное свойство q_i – характеристики качества;

$M = \{m_1, m_2, \ldots, m_k\}_{k=1,\ldots,K}$, – множество метрик (*Metrics – M*) для каждого a_j атрибута, используемого после измерения этого атрибута;

$W = \{w_1, w_2, \ldots, w_n\}_{n=1,\ldots,N}$, – множество весовых коэффициентов (*Weights – W)* для нивелирования метрик атрибутов.

Базовые показатели качества ПО систем обозначают:

q_1: функциональность (functionality),

q_2: надежность (realibility),

q_3: удобство применения (usability),

q_4: эффективность (efficiency),

q_5: сопровождаемость (maitainnability),

q_6: переносимость (portability).

Данная модель качества - четырехуровневая.

На *первом уровне модели* находятся характеристики (показатели) качества *Q*, отображающие свойства, которыми должна обладать созданная система (табл.1). Каждый из показателей q_i обладает свойствами, определяющими разные значения, которые необходимы для соответствующего вида систем.

Таблица 1. Показатели качества ТПС

Характеристика (показатель)	**Атрибут показателя**	**Метрический анализ**	**Оценка Продукта**
Функциональность q1 – functionality	Точность, наличие функций, корректность, защищенность, интероперабельность	Экспертиза структуры системы, полноты функций, непротиворечивость, защита, совместимость со средой	Оценка сложности 1-функция реализована 0 – нет 0,9 | 1 $0,75 \leq К_З \leq 1$
Надежность q2 - reliability	Безотказность, отказоустойчивость, завершенность	Экспертиза безошибочности, функционирования восстанавливаемости, помехоустойчивости	Оценочные методы надежности $0,75 \leq Н \leq 1$
Эффективность q3 - efficiency)	Реактивность, рациональность использования ресурсов	Экспертиза критериев эффективности, работоспособность системы	0 или 1

Эргономичность, удобство применения q4 –usability	Понимаемость алгоритмов, обучаемость, адаптивность	Экспертиза простоты обучения, работа независимости от среды, возможности обучения	Оценкапо формулам 0 или 1
Сопровождаемость q5 – maintainability	Стабильность документируемость, изменяемость, тестируемость	Экспертиза спецификаций, освоения, внесение изменений, Верификация	0 или 1
Переносность q6 – portability	Разработка Внедрение, сопровождение стандартизация, унификация	Экспертиза структуры, интерфейсов, оценка трудоемкости внедрения, сопровождение КПИ (reuse), унификация	0 или 1 Оценка по формулам (гл.8) 0 или 1 0 или 1

Показатели *второго уровня* – это атрибуты показателей *A* (корректность, безотказность и др.), необходимые для разрабатываемой ПТС, чтобы достигнуть заданные характеристики качества первого уровня. Каждый атрибут представляется набором единичных свойств, уточняющих показатели качества первого уровня, т. е. каждый показатель определяется набором отдельных атрибутов, которые необходимо достигнуть в процессе разработки системы на процессах ЖЦ и использовать их при комплексной оценке качества системы.

На *третьем уровне* находятся метрики *M*. (ISO/IEC 9126-1-4:2002. Метрики ПО) **Метрика** – это совокупность оценочных элементов, позволяющих определить степень достижения группы атрибутов по каждому показателю качества в отдельности. Иными словами, метрика, характеризует показатель качества, который может иметь один или несколько оценочных элементов.

На *четвертом уровне* находятся оценочные элементы *W*, являющиеся элементарными характеристиками отдельных свойств создаваемого ПС. Набор оценочных элементов группируется и распределяеся по соответствующим процессамтехнологии разработки ПТС.

Текущая оценка отдельных элементов качества проводится на всех ЖЦ. Контрольные точки проверки – это специальные технологические операции, выполняемые службой инспекции результатов.

На процессах ЖЦ проводится также метрический анализ степени достижения соответствующего атрибута и соответственно показателя качества путем:

– экспертизы состояний компонентов (структура, правильность оформления документации и др.) и заполнения протокола экспертизы;

– аналитических вычислений с помощью информации, формируемой в процессе регистрации ошибок о тестировании объекта на соответствующих процессах ЖЦ;

– оценки надежности соответствующими математическими моделями и методами;

– комплексной оценки качества созданной системы с использованием отдельных промежуточных значений атрибутов оцененных элементов на ЖЦ.

Модель качества (стандарт ISO/IEC 9000 (1-4) или ISO/IEC 9126-2002)

Стандарт содержит шесть показателей качества (q-quality) q_1-q_6, которые представлены в табл.1. Каждый показатель обладает свойством и признаками, которые могут оцениваться экспертным и аналитическим путем. Определение *количественных* показателей

качества начинается с ранних этапов ЖЦ стандарта ISO/IEC 12207: 2007 Процессы ЖЦ ПО. В этом стандарте определен процесс «*управления качеством»,* включающий верификацию, тестирование и обеспечение гарантий качества ПО.

Приведем показатели модели качества $M_{кач}$ и формулы для их оценки.

Функциональность включает свойства, определяющие способность системы предоставлять требуемое множество функций для решения задач с учетом заданных требований. Этот показатель задается атрибутами $q_1 = \{a_{11}, a_{12}, a_{13}, a_{14}, a_{15}, a_{16}\}$, обозначающими:

a_{11}: функциональная полнота определяет степень достаточности функций для решения задач в соответствии с его назначением. Данный атрибут представляется в виде отношения всех реализованных функций F^c в компонентной системе (c), к функциям F^m (m – требование):

$$a_{11} = \sum_{i=1}^{N} F^c / \sum_{j=1}^{K} F^m.$$

a_{12}: корректность, означающий степень достижения правильности каждой функции F^m, заданной в требовании, и функции F^c, реализованной в системе. Система корректна, если $F^m = F,^c$ и частично корректна, если $F^m \subset F^c$. Для большинства ПС достаточно частичной корректности. Степень корректности – это степень функциональной корректности: $\surd = 1 – (card\ (F^m / F^c)\ /card\ F^m$.

a_{13}: точность определяет получение результатов с необходимой степенью точности и оценивается отношением ∇ разности значений функций $F^c{}_i\ (D_i)$ и $F^m{}_i\ (D_i)$ на D_i входном наборе к значению функции в соответствии с выражением:

$$\nabla = \sum_{i=1}^{card\ F^m} ((F^c{}_i\ (D_i) – F^m{}_i\ (D_i))/ F^m{}_i\ (D_i))/card\ F^m.$$

a_{14}:: интероперабельность – свойство, обеспечивающее взаимодействие систем в другой операционной среде;

a_{15}: защищенность – атрибут, который показывает возможность элемента системы фиксировать дефекты, а также ошибки, связанные с данными. Оценку степени защищенности можно провести с помощью выражения: $a_{15} = fal^z / fal$,

где fal^z – количество дефектов, от которых элемент защищен; fal – общее количество дефектов системы;

a_{16}: согласованность – атрибут, который показывает степень соблюдения стандартов, правил и других соглашений.

Надежность ПО – определяется как вероятность того, что компоненты системы или сама система функционируют безотказно в течение фиксированного периода времени в заданных условиях операционного окружения/среды. В модели качества надежность задается на множестве атрибутов $q_2 = \{a_{21}, a_{22}, a_{23}, a_{24}\}$, которые определяют способность системы преобразовывать исходные данные в результаты при условиях, зависящих от периода жизни системы (износ и старение не учитываются). Снижение надежности компонентов происходит из-за ошибок проектирования. Отказы и ошибки в программных компонентах могут появляться на заданном промежутке времени функционирования компонента/системы.

К атрибутам надежности относятся.

a_{21}: безотказность – свойство системы функционировать без отказов (программ или оборудования). Если компонент содержит дефект, то во множестве $D=\{De/e \in L\}$ всех дефектов, можно выделить подмножество $E \subseteq D$, для которых результаты не соответствуют функции F^m, заданной в требованиях на разработку. Вероятность p безотказного выполнения компонента на De, случайно выбранном из D среди равновероятных, равна:

$p = 1– card\ \{E\}/ card\ \{D\}$.

Отказ (failure) показывает отклонение поведения системы от предписанного выполнения предписанных ей функций. Появление отказа может быть причиной ошибки (fault/ error), вызывающей его. Если ошибка сделана человеком, то используется термин mistake. Когда различие между fault и failure не критично используется термин defect, означающий либо fault (причина), либо failure (действие). Связь между этими понятиями такая: fault → error → failure.

Существует большое разнообразие видов отказов ПО, типичные из них: внезапные, постепенные, перемещающиеся (сбои). Причины отказов могут быть физические, структурные, отказы взаимодействия и др. Они могут возникать естественным путём, вноситься человеком или внешней операционной средой в период создания или эксплуатации системы, а также быть постоянными или носить временный характер.

Наработка на отказ определяет среднее время между появлением угроз, нарушающих безопасность, и обеспечивает трудно измеримую оценку ущерба, которая наносится соответствующими угрозами.

Вычисление среднего времени *T* наработки на отказ реализуется формулой:

$$T = \sum_{I=1}^{De} \nabla t_i^E \ / N,$$

где ∇t_i^E – интервал времени безотказной работы компонента *i* –го отказа; *N* – количество отказов в системе.

a_{22}: устойчивость к ошибкам показывает на способность системы выполнять функции при аномальных условиях (сбоях аппаратуры, ошибках в данных и интерфейсах, нарушениях в действиях операторов и др.). Оценку устойчивости можно получить по формуле: $\Upsilon = N^v/N,$ где N^v – количество разных типов отказов, для которых предусмотрены средства восстановления; *N* – общее количество всех отказов в системе.

a_{23}: восстанавливаемость – свойство, показывающее способность возобновлять функционирование системы после отказов для повторного исполнения. Среднее время восстановления компонента можно определить по формуле

$$T = \sum_{I=1}^{De} \nabla t_i^b \ / D,.$$

где ∇t_i^b – время восстановления работоспособности компонента после i – отказа; *De* – количество дефектов и отказов в системе.

a_{24}: согласованность – атрибут, который показывает степень соблюдения стандартов, правил и других соглашений.

Обнаруженные ошибки устраняются, а надежность системы возрастает. Чем интенсивнее проводится эксплуатация, тем интенсивнее выявляются ошибки, и обеспечивается рост надежности. Фактором, влияющим на оценку надежности ПО, относится – угроза, приводящая к снижению безопасности системы и ущербности всей системы.

В проблеме надежности ПО важное место занимает понятие устойчивости системы к отказам ПО и возможности восстанавливаться самопроизвольно после отказа. Количественная оценка надежности системы имеет вид

$$q_2 = \sum_{j=1}^{4} a_{2j}\, m_{2j\, W\, 2j.}$$

Удобство применения **–** это множество свойств ПС, обеспечивающих условия использования ее пользователями для получения необходимых результатов. Эта характеристика определяется на множестве таких эргономичных атрибутов:

a_{31}: свойство понимания выражается усилием, затрачиваемым на распознавание логических концепций и условий применения системы;

a_{32}: свойство возможности изучения означает усилия отдельного пользователя понять, как применяется ПО, как проводиться контроль (например, ввода, вывода данных), а также процедур и документации;

a_{33}: оперативность – реакция системы при выполнении разных операций и их контроля;

a_{34}: согласованность – соответствие системы требованиям стандартов, соглашений, правил, законов и предписаний.

Все атрибуты оцениваются экспертами, которые в зависимости от их уровня знаний дают соответствующие качественные заключения. Данная характеристика имеет вид:

$$q_3 = \sum_{j=1}^{4} a_{3j}\, m_{3j\, W\, 3j}.$$

Эффективность – свойства системы, которые показывают взаимосвязь между уровнем ее выполнения, количеством используемых ресурсов (аппаратуры, расходных материалов и др.), услуг штатного обслуживающего персонала и др.

К свойствам относятся:

a_{41}: реактивность – время отклика, обработки и выполнения функций компонентов/системы;

a_{42}: эффективность – количество используемых ресурсов при выполнении функций ПО и продолжительность их вычислений;

a_{43}: согласованность – соответствие данного атрибута заданным стандартам, правилам и предписаниям.

Количественная оценка данной характеристики имеет вид

$$q_4 = \sum_{j=1}^{3} a_{4j}\, m_{4j\, W\, 4j}.$$

Сопровождаемость – множество свойств, которые отражают усилия, которые затрачиваются на проведение модификаций, включая корректировку, усовершенствование и адаптацию системы для новой среды выполнения, а также требований или функциональных спецификаций. Данная характеристика в модели качества состоит из следующих атрибутов:

a_{51}: анализируемость – свойство, необходимое для проведения диагностики отказов в ПО и идентификации частей системы с целью модификации;

a_{52}: изменяемость – свойство, которое определяет возможность проведения модификации компонента или системы с удалением ошибок при внесении изменений, а также дополнения новых возможностей в систему или среду функционирования;

a_{53}: стабильность – способность системы работать без ошибок и выполнять необходимые функции без риска проведения ее модификации;

a_{54}: тестируемость – свойство, обеспечивающее возможность верификации в целях проверки правильности и возможного обнаружения разного рода ошибок, а также валидации требований на соответствие их правильной реализации в системе;

a_{55}: согласованность – соответствие данного атрибута определенным стандартам, соглашениям, правилам и предписаниям.

Количественная оценка данной характеристики проводится экспертным и аналитическим способом по формуле

$$q_5 = \sum_{j=1}^{5} a_{5j}\, m_{5j\, W\, 5j}.$$

Переносимость **–** множество атрибутов, указывающих на возможность системы приспосабливаться к работе в новых условиях среды (организационной, аппаратной и программной). Перенос на другую платформу или среду связан с совокупностью действий по обеспечению возможности функционирования в новой среде, отличной от той, в которой система создавалось. К атрибутам данной характеристики относятся:

$a_{61:}$ адаптивность – способность системы адаптироваться к различным операционным средам, она оценивается по формуле: $a_{61} = Za / Zd,$ где

Za – затраты на адаптацию к новой операционной среде;

Zd – затраты на разработку новой системы для новой операционной среды;

a_{62}: настраиваемость – атрибут, определяющий необходимые затраты на запуск или инсталляцию программного продукта в другой среде;

a_{63}: сосуществование – возможность использования специального ПО или компонентов среды работать вместе в одной среде;

a_{64}: заменяемость – возможность заменять отдельные компоненты другими с условием обеспечения их взаимодействия (интероперабельности) с другими программами и не совместной работы в заданных условиях среды;

a_{65}: согласованность – соответствие стандартам или соглашениям и правилам переноса программной системы в другую среду.

Количественная оценка данной характеристики с учетом соответствующих метрик и их весов имеет вид:

$$q_6 = \sum_{j=1}^{5} a_{6j}\, m_{6j}\, W_{6j}.$$

Приведенные формулы для оценки показателей качества с использованием метрик $a_i \subset A$ и весовых коэффициентов $w_i \subset W$ для каждого атрибута могут использоваться отдельно для каждого компонента системы. Используя полученные оценки q_j характеристик качества применительно к отдельному компоненту (com), получаем интегральную оценку качества одного компонента в виде:

$$Q_{com} = \sum_{j=1}^{6} q_j .$$

Если ПС система содержит N–компонентов и для них проведена оценка, то комплексная оценка качества системы (sys) имеет вид:

$$Q_{sys} = \sum_{l=1}^{N} Q^{l}_{com}.$$

Заметим, имеются и другие подходы к аналитической оценки качества ПО.

Процесс измерения и контроля показателей качества проводится членами службы качества и включает борьбу с угрозами качества и обнаруженными дефектами/ошибками. На процессах ЖЦ проводится *выявление и устранение дефектов* (что повышает качество), но и могут вноситься некоторые *дефекты* в ПТС (что снижает качество). Для определения уровня завершенности компонентов ПТС на процессах ЖЦ используется теория прогнозирования.

На основе полученных данных о надежности и других показателях качества (надежность, эффективность и др.) рассчитывается целевое значение завершенности и полезности системы, адекватных потребностям заказчика. При этом мера эксплуатационного качества системы определяется функцией полезности вида:

$$Q_{nc} = \sum_{i=1}^{k} a_i \cdot R_i$$

где a_i – мера важности i-й функции системы для процесса, R_i – надежность выполнения функций в заданном периоде t эксплуатации системы. Полученные данные по всем показателям качества используются при формировании сертификата качества.

Применение моделей надежности для оценки ПТС

Применение моделей надежности для небольших, средних и больших проектов показывает, что наиболее перспективными являются модели оценочного типа, которые базируются на пуассоновских процессах (модели Мусы, Гоэла–Окомото, S-образные и др.). По этим моделям надежность стремиться к 1. Одним из недостатков является форма кривой интенсивности выявленных отказов или неисправностей (экспоненциальная) и строго спускается при t>0. Это свидетельствует о том, что при тестировании проведено недостаточно экспериментов или мало найдено ошибок, когда интенсивность отказов была близка 0. В системе могут оставаться ошибки и их поиск требует больше времени.

Что касается S-образной модели, функция интенсивности λ(t) выявления ошибок в зависимости от времени работы имеет вид:

$$\lambda(t) = a\beta^2 t \exp(-\beta t),$$

где a – общее количество дефектов, обнаруженных от начала и до конца тестирования;

β – скорость изменения функции интенсивности при выявлении отказов.

Введение в формулу параметра в степени 1 модели Мусы и Гоэла–Окомото дает изменение формы кривой так, что она сначала растет, а потом спадает. Практика применения этих моделей в ПТС привела к уточнению функции интенсивности при введении дополнительного параметра n:

$$\lambda(t) = a\beta^{n+1} t^n \exp(-\beta t),$$

где n отражает сложность и размер проекта системы. Это позволяет более точно определить форму кривой интенсивности с учетом получаемых практических результатов. В таблице 2 представлены практические значения функций интенсивности отказов λ (t) и количество отказов μ(t) для базовых и общих моделей. В них значения a и β находятся в следующих соотношениях: N= a, β = a, b = β, β1 = β, a0 = aβ.

Таблица 2. Характеристики моделей надежности Пуассоновского типа для вычисления интенсивности и количества отказов.

Название модели	Функции интенсивности отказов $\lambda(t)$	Функциz количества отказов $\mu(t)$
Модель Гоэла-Окумото	$\lambda(t) = Nb \exp(-bt)$	$\mu(t) = N(1 - \exp(-bt))$
Модель Мусы	$\lambda(t) = \beta_0 \beta_1 \exp(-\beta_1 t)$	$\mu(t) = \beta_0 (1 - \exp(-\beta_1 t))$
S –подобная модель	$\lambda(t) = a\beta^2 t \exp(-\beta t)$	$\lambda(t) = a\{1-(1+\beta t) \exp(-\beta t)$
Модель Шнайдевинда	$\lambda(t) = a_0 \exp(-\beta t)$	$\mu(t) = a_0/\beta (1 - \exp(-\beta t))$
Общая модель пуассоновского процесса	$\lambda(t) = a\beta^{n+1} t^n \exp(-\beta t)$	$\mu(t) = a (n^1 - \sum n\beta^{n-1}/(n-1)^1 t^n \exp(-\beta t)))$

Здесь параметр n зависит от процесса тестирования и его значений:

n=0 для небольшого проекта, в котором разработчик является также тестером (модели Мусы, Гоэло-Окомото и др.);

n=1 для среднего проекта, в котором тестирование и проектирование ПО исполняются несколькими разработчиками из одной рабочей группы (S- образная модель);

n=2 для большого проекта, в котором группы разработчиков работают параллельно;

n=3 для очень большого проекта, в котором группы тестирования и разработки работают независимо друг от друга.

На основе проведенных экспериментальных данных получены функции о количестве отказов μ(t) и интенсивности отказов λ (t) на выходных данных и значениях параметра n, который показывает вид функций μ(t) при разных значений n = 0, 1, 2, 3.

Наибольшее приближение достигается при n=3, а наименьшее при n = 0 (модель Мусы, Гоэло-Окомото и др.). Это подтверждается соответствующими статистическими данными (табл.3), которые задают разницу между выходными данными (t_2) и соответствующими значениями функции μ(t) при значениях n = 0, 1, 2, 3. На основе экспериментальных данных a, β, n, приведены значения функций μ(t) и λ (t) при n = 3, 2, 1, полученные при использовании методов оценки надежности Мусы, Мусы-Окомото и Шнайдевинда.

Таблица 3. Статистические данные для μ(t) при n=3, 2, 1 и данных t_2

Статистические показатели отклонений	**Разница функцій *t_2 - μ_3***	**Разница функций *t_2 - μ_2***	**Разница функций *t_2 - μ_1***	**Разница функций *t_2 - μ***
Среднее	16.13522	16.22889	19.88387	58.93807
Медианное	15.27700	14.11600	16.0000	60.89700
Максимум	33.58100	54.23600	49.10800	88.80200
Минимум	4. 848000	-1.280000	4175000	15.96200
Среднеквадрати-ческое отклонение	8.374089	17.37143	14.07056	23.63765

3.8.3. Марковский анализ безопасности и надежности ТС (Зеленов С.)

Одним из методов оценки безопасности программно-аппаратных систем является марковский анализ.

Вероятностный или стохастический автомат — это конечный автомат, в котором нет входных и выходных символов, а переходы между состояниями осуществляются с заданной вероятностью.

Исследуемая на предмет безопасности система может быть описана в виде вероятностного автомата, содержащего особые состояния, соответствующие отказам. Вероятность перехода в такое состояние — это вероятность возникновения отказа.

Во многих случаях можно считать, что будущее технической системы определяется ее текущим состоянием, так что построенный вероятностный автомат является *марковской цепью* и к нему применимы все результаты теории марковских цепей.

Построение марковской цепи

Для данной модели, состоящей из компонентов $c_1, \ldots, c_N$, рассмотрим следующий вероятностный автомат. Состояниями этого автомата являются кортежи состояний всех компонентов $s_1 \times \ldots \times s_N$.

Переход автомата в другое состояние происходит при возникновении некоторого набора внутренних сбоев в компонентах. Для данного состояния s автомата и данного набора внутренних сбоев $\{e_{i,j}\}$ итеративно анализируется распространение сбоев между компонентами и переходы компонентов в новые состояния до тех пор, пока для модели не будет достигнуто такое состояние s', в котором модель стабилизируется. Таким образом в вероятностном автомате имеется переход $s \to s'$ с вероятностью $\prod_{i,j} p(e_{i,j})$.

Отметим, что если в модели имеется N компонентов и L различных внутренних сбоев, причем каждый компонент может находиться не менее чем в двух состояниях («рабочее», «нерабочее», а также, возможно, несколько промежуточных состояний), то вероятностный автомат будет содержать не менее чем 2^N состояний, и из каждого состояния будет выходить 2^L переходов.

Однако, на практике не все состояния вероятностного автомата являются достижимыми из некоторого начального состояния. Для заданного начального состояния модели вероятностный автомат строится постепенно, по мере построения переходов из уже достигнутых состояний. В результате количество полученных состояний оказывается существенно меньше экспоненты.

Кроме того, в реальных системах отдельные сбои имеют довольно низкую вероятность, порядка 10^{-4} .. 10^{-12}. Соответственно, вероятность одновременного возникновения нескольких сбоев является исчезающее малой величиной. Таким образом, на практике наборы сбоев $\{e_{i,j}\}$ можно ограничить парами или тройками. В результате количество переходов из каждого состояния вероятностного автомата будет ограничено полиномом степени 2 или 3.

Анализ видов последствий и критичности отказов

Одним из методов оценки безопасности программно-аппаратных систем является анализ видов и последствий отказов (ГОСТ 27.310-95).

Анализ видов и последствий отказов является анализом “снизу вверх”, когда анализ стартует с некоторого сбоя в отдельном компоненте и определяет, к каким последствиям может привести этот сбой. При проведении этого анализа:

выявляют возможные сбои компонентов и системы в целом, изучают их причины, механизмы и условия возникновения и развития;

определяют возможные неблагоприятные последствия возникновения выявленных отказов, проводят качественный анализ тяжести последствий отказов и/или количественную оценку их критичности.

Критичность отказов оценивают с использованием показателей, учитывающих для каждого анализируемого отказа:

- вероятность его возникновения за время эксплуатации;
- условные вероятности наступления всех возможных неблагоприятных последствий отказа;
- размер возможного ущерба в результате наступления каждого из ожидаемых последствий отказов.

Инструментальная поддержка анализа рисков

В рамках проекта реализованы следующие методы анализа надежности технических систем:

– Анализ дерева неисправностей методами логико-вероятностного анализа.

– Марковский анализ методами теории Марковских процессов.

– Анализ видов последствий и критичности отказов.

– Анализ надежности технической системы производится на основе формального описания системы на языке AADL.

В рамках ***анализа дерева неисправностей:***

- для заданного отказа целевого компонента автоматически строится дерево неисправностей;
- автоматически вычисляются минимальные сечения;
- автоматически вычисляется вероятность возникновения заданного отказа;

- атомарные отказы подкомпонентов автоматически ранжируются по мере значимости относительно величины вклада в вероятность возникновения заданного отказа.

В рамках ***Марковского анализа***:

- автоматически строится Марковская цепь;
- автоматически составляются уравнения Колмогорова-Чепмена и формулируется задача Коши;
- автоматически численными методами находится решение задачи Коши;
- автоматически вычисляется вероятность отказа целевого компонента в заданный момент времени.

Решение полученной задачи Коши и нахождение вероятностей отказов данной (под)системы в заданные моменты времени осуществляется численно методом Рунге-Кутты.

В рамках ***анализа видов последствий и критичности отказов***:

- для заданной комбинации отказов целевых компонентов автоматически вычисляются ее возможные причины и их вероятности;
- для заданной комбинации отказов целевых компонентов автоматически вычисляются ее последствия;
- для заданной комбинации отказов целевых компонентов автоматически вычисляется ее критичность для системы в целом.

Поддерживаются две разновидности анализа видов и последствий отказов:

исходными сбоями считаются внутренние ошибки компонентов;

исходными сбоями считаются нерабочие состояния компонентов.

В *первом случае* (когда исходными сбоями считаются внутренние ошибки компонентов) таблица видов и последствий отказов соодержит следующие данные:

- Item: компонент, в котором произошел исходный сбой;
- Initial failure mode: исходный сбой (событие) в данном компоненте;
- End effect: набор финальных последствий данного исходного сбоя;
- Sev: тяжесть данного последствия для системы (по 10-балльной шкале);
- P: вероятность достижения данного последствия в результате возникновения данного исходного сбоя.

Во *втором случае* (когда исходными сбоями считаются нерабочие состояния компонентов) таблица видов и последствий отказов соодержит следующие данные:

- Item: компонент, в котором произошел исходный сбой;
- Initial failure mode: исходный сбой (нерабочее состояние) в данном компоненте;
- Potential cause: возможные причины данного сбоя:
- Cause: описание непосредственной причины сбоя;
- P: абсолютная вероятность возникновения данного сбоя;
- Occ: рейтинг вероятности возникновения данного сбоя (по 10-балльной шкале);
- End effect: набор финальных последствий данного исходного сбоя;
- Sev: тяжесть данного последствия для системы (по 10-балльной шкале);
- P (cond): условная вероятность достижения данного последствия в результате возникновения данного исходного сбоя.

3.8.4. Прогнозирование и распределение надежности по компонентам

Прогнозирование надежности компонентов

Метод прогнозирования значения надежности выполняется по следующей модели надежности:

$$R_i = \exp[\ -D_i\ I_i \cdot (1 - \exp(\ \frac{\rho_i \cdot K}{I_i \cdot \varphi_i} \cdot t))] \ ,$$

где ρ_i – параметр среды эксплуатации *i*-го компонента, φ_i – характеристика среды разработки, I_i – оцененный размер начального кода, а D_i – прогнозируемая плотность дефектов в системе. Коэффициент дефектов K – константа, полученная для всех объектов ПТС, а значения ρ_i и φ_i – взяты при первоначальном прогнозировании надежности, которые не изменяются во время разработки компонентов системы.

Оценка надежности системы выполняется согласно классификации дефектов (Orthogonal Defect Classification), в соответствии с которой каждый выявленный дефект использует параметры: тип дефекта, триггер дефекта, влияние дефекта. Эти параметры используются одной или двумя подходящими моделями надежности, из выше приведенных, в целях проведения оценки прогнозного значения надежности отдельных компонентов и системы в целом. Результаты оценки сравниваются и выбирается наиболее правдоподобная модель надежности.

Метод распределения надежности по компонентам системы путем парного их сравнения и построения квадратной матрицы A ($n \times n$) из элементов вида:

$$a_{11} = a_{22} = ... = a_{nn} = 1,\ a_{ij} = \frac{1}{a_{ji}}\ ,\ u, j = 1,...,n;\ i \neq j;\ n=k,\ l,\ m,$$

где n – количество сравниваемых компонентов, *k, l, m* – количество функций и модулей соответственно. Матрица включает относительный вес w_i-го компонента, который вычислялся по формулам: $w_i = \frac{\sum_{j=1}^{n} a_{ij}}{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}},\ \sum_{i=1}^{n} w_i = 1$

В случае больших размеров матрицы в целях получаются более точные оценки компонентов иерархии, вычисляется собственный вектор и собственные значения матрицы. В них используются следующие данные: λ_{max} – максимальное собственное значение матрицы A *n*–порядка, w_i – коэффициент относительного веса элементов матрицы A, W = ($w_1, w_2, ... w_n$) – собственный вектор, которому соответствует λ_{max}. Общность решения задачи сравнения устанавливается соотношением $\alpha = \sum_{i=1}^{n} w_i$ и значением $\sum_{i=1}^{n} w_i = 1$. Если матрица A имеет *n-1* собственных значений λ, равных нулю и $\lambda_{max} = n$, то она является согласованной. При этом индекс согласованности *CI* и коэффициент согласованности *CR* вычисляется по формулам:

$$CI = \frac{\lambda_{\max} - n}{n - 1},\ CR = \frac{CI}{E\ (CI\ )},$$

где *E(CI)* – математическое ожидание матрицы парных сравнений $A(n \times n)$.

Критерий приемлемости парного сравнения элементов в матрицах размером $n \geq 3$ получен такой: $CR \leq 0.05$ и $CR < 0.1$ для $n>5$. По результатам сравнения формируется квадратная матрица $F(k \times k)$. Аналогично проводится сравнение компонентов ПТС. В

результате сравнения получается k матриц. Возможный порядок каждой матрицы – i, а максимальный из них – m.

Инструментом для сравнения является ExpertChoice входной матрицы A, которая автоматически вычисляет собственный вектор W, собственное значение λ_{max} и коэффициент согласованности CR. Для вычисления λ_{max} и W используются соответствующие функции пакета MATLAB 6.5.

Результаты сравнений заносятся в форму, содержащую перечень весовых коэффициентов программ, критерии, индексы и коэффициенты согласованности. Они предоставляются в виде готовых результатов обработки матриц. Полученные весовые коэффициенты синтезируются с помощью пакета MATLAB 6.5. Результаты отображаются в виде отчета о распределении надежности по объектам системы.

Прогнозирование плотности дефектов

Прогнозирование плотности дефектов по модели RLM (Rome Laboratory Model) выполняется следующими действиями:

1). Анализ значений параметров модели прогнозирования, включая остаток дефектов от предыдущего этапа работ с ПО системы, и используется для целевого распределения значения надежности.

2). Сравнение прогнозируемого значения надежности с распределенным значением.

3). Корректировки переменных параметров и учет текущего состояния системы.

4). Оценка параметров модели прогнозирования надежности.

5). Прогнозирование плотности дефектов.

6). Определение значений (допусков) для оценок результатов прогнозирования и анализа альтернатив.

7). Расчет прогнозного значения надежности системы.

Расчет плотности дефектов делается с помощью модели RLM. Вначале выполняется прогнозирование плотности дефектов по формуле: $D_0 = \prod_{i=1}^{9} K_i$,

где K_i – модификатор плотности дефектов D_0, с учетом пороговых значений данных о плотности дефектов.

Для каждой системы результаты сравниваются с полученными по модели RLM. При проверке оказалось, что для ПО объемом 10 - 25 KSLOC погрешность прогнозирования плотности дефектов примерно составила 30-35%. Это объясняется некоторыми ограничениями системы Hugin Lite 6.5. Эти результаты используются при прогнозировании надежности компонентов.

На следующий год ставятся задачи оценки качества систем в классе программной продукции для транспортной или строительной отрасли промышленности.

3.8.5. Заключение по проекту РФФИ 206 -2019

В проекте РФФИ № 19-01-00206 «Модели, методы и средства надежности технических и программных систем с обеспечением качества и безопасности» 2019 г. проведен анализ и исследование технологии сборки сложных систем из готовых и разрабатываемых программных элементов (модулей, объектов, компонентов, сервисов и др.), начиная с истории появления ЭВМ и информатики (70-летие), проведенное по предыдущему проекту РФФИ. Рассмотрена теория графового моделирования ТПС с использованием разных ресурсов Интернет и программных элементов из разных предметных областей и подходов к обеспечению качества функционирования ТПС.

Готовые ресурсы описывались ЯП (C, C++, Basic, Java, Python и др.), а их интерфейсы задавались в новом языке WSDL. Эти элементы трансформировались к выходному коду и могут обрабатывать передаваемые данные (простые, структурные и неструктурированные) типы данных в распределенной среде Интернет.

Рассматривались ресурсы Интернет: интеллектуальные, сервисные, сервисно-компонентные и информационные. Для работы с разнотипными ресурсами использовался формальный аппарат отображения (mapping) неструктурных типов GDT стандарта ISO/IEC 11404–2007) к фундаментальным типам данных ФДТ для современных общесистемных сред VS.Net, IBMWebSphere, BSD, JAVAEE, OS Linux и др. Исследованы методы обработки неструктурированных данных и больших данных для Web-applications, которые проводились разными методами, в том числе с помощью методов извлечения данных Web Content Mining и ЯП разных сред W3C Интернет.

Используемые программные элементы отрабатывались на процессах ЖЦ: верификации, тестирования, конфигурации и сбора данных для проведения оценки надежности программ, работающих с большими данными и неструктурированными данными, используемыми в Cloud Computing. На процессе конфигурационной сборки ресурсов и компонентов проводился поиск ошибок, отказов и дефектов при функционировании некоторого варианта ТСП. Приведена стандартная модель качества ISO 9126 и формулы вычисления шести показателей надежности. Приведено экспериментальное измерения надежности по оценочным моделям и представлены в таблице. Описан метод оценки надежности технических средств по Марковскому методу с учетом возможного появления угроз, ошибок и дефектов, а также распределения надежности по компонентам для измерения показателей качества ТПС. В заключении отчета приведены доклады на конференции ИСП РАН-2019 http://0x1.tv/20180517F, в статьях по этому докладу 2019.

Список литературы к под разделу РФФИ 206-2019

1. Лаврищева Е.М., Пакулин Н.В. Доклад «Модели и методы надежности технических и программных систем». Конф. ИСП РАН-18 OS DAY- Надежность.- 17-18 мая 2018.
2. Лаврищева Е.М., Пакулин Н.В., Рыжов А.Г., Зеленов С.В. Анализ методов оценки надежности оборудования и систем. Практика применения методов. Труды ИСП РАН, том29, N6. DOI: 10.15514/ISPRAS-2018-30(3), 2018..- p.93-112.
3. Андон Ф.И., Коваль Г.И. и др. Основы инженерии качества программных систем. К.: Наук. думка, 2007, 670 с.
4. Липаев В.В. Надежность и функциональная безопасность комплексов программ реального времени.- Москва, ИСП. РАН. 2013.-190 с.
5. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование. Основы индустрии программных продуктов. К.: Наук. думка, 2009, 372 с.
6. Lavrischeva K.: Formal Fundamentals of Component Interoperability in Programming. In: Cybernetics and Systems Analysis, vol. 46, no. 4, pp. 639–652. Springer, Heidelberg (2010).
7. Липаев В.В. Методы обеспечения качества крупномасштабных программных средств. – М.: СИНТЕГ.- 2006.–520 с.
8. С.А Зеленова, С.В. Зеленов. Моделирование программно--аппаратных систем их безопасности. Труды ИСП РАН, Том29, выпуск5.- с.257-282.
9. Musa J.D. Okumoto K. A. Logarithmic Poisson Time Model for Software Reliability Measurement. In Proc. of the 7th International Conference on Software Engineering, 1984, pp. 230–238.
10. Shanthikumar J.G.Software realibility models^ A Review. Microelectronics Reliability, v.23, N5. 1983
11. Yamada S., Ohba M., Osaki S. S–shaped software reliability grows modeling for software error detection. *IEEE Transactions* on *Reliability, vol*. R–32, № 5, pp. 475–478.

12. Chulani S. Constructive quality modeling for defect density prediction: COQUALMO. In Proc. of the International Symposium on Software Reliability Engineering (ISSRE'99), 1999.

13. Duval P., Matyas R., Grover A. Continuous integration improving Software quality and reducing risk. Addison Wesley, 2009, 691 p.

14. Горбенко А.В., Засуха С.А., Рубан В.И., Тарасюк О.М., Харченко В.С. Безопасность ракетно-космической техники и надежность компьютерных систем: 2000-е годы. Авиационно-космическая техника и технология, №1(78), 2011, с. 9-20.

15. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, 2004, pp. 11-33.

16. Липаев В.В. Программная инженерия сложных заказных программных продуктов. Учебное Пособие, 2014.- 308с. ISBN)78=5-317-04750-4.

17. И.С. Захаров, М.У. Мандрыкин, В.С.Мутилин, Е.М. Новиков, А.К. Петренко, А.В. Хорошилов. Конфигурируемая система статической верификации модулей ядра операционных систем. Программирование, №1, 2015, стр. 44-67.

18. Кулямин В.В., Лаврищева Е.М., Мутилин В.С., Петренко А.К. Верификация и анализ вариабельных операционных систем// Труды ИСП РАН.-М.: 2017, Том 28. Вып.3.- с.189-209.

19. E.M. Lavrischeva, V.S.Mutilin, A.G.Ryzhov. Designing variability models for Software operating systems and their families. Proceeding of the System Programming of the AS. Vol.28, issue 5. P.93-110.

20. Kozin S.V., Mutilin V.S. Static Verification of Linux Kernel Configurations. Trudy ISP RAS. Proc. ISP RAS, vol. 29, issue 4, 2017, pp. 217-230. DOI: 10.15514/ISPRAS-2017-29(4)-14.

21. Козин. С.В. Конфигурационная сборка варианта ядра Linux для прикладных систем. Труды ИСП РАН, 2018,- Том 29. Вып.3.-с.171-186.

22. Лаврищева Е.М., Рыжов А.Г. Подход к созданию систем и сайтов из готовых ресурсов.- Научный сервис в сети Интернет: Труды XX Всероссийской научной конференции (17-22 сентября 2018.- Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2018. —321-346с. ISBN 978-5-98354-046-0.

23. E. M. Lavrischeva, The Scientific basis of software engineering.- International Journal of Applied and Natural Sciences (IJANS), ISSN(P): 2319-4014; ISSN(E): 2319-4022 Vol. 7, Issue 5, Aug. 2018.- p.13-32

Доклады и статьи по тематике проекта РФФИ №19-01-00206 -2019

1. Моделирование систем для прикладных областей знаний. Пути развития системного программирования (Екатерина Лаврищева, ISPRASOPEN-2019. http://0x1.tv/20191206AB)

Аннотация. Дана характеристика направлений развития системного программирования ПО (ОС, трансляторов, отладчиков, верификаторов и др.), с помощью которого создаются системы для разных предметных областей знаний (математика, физика, биология, медицина и др.). Рассмотрены базовые задачи первых ОС и трансляторов у нас в стране на первых ЭВМ, аналогичные системной инженерии Computer Science. Представлена история формирования системного программирования и его применение при решении задач в космосе, авиации и др. Описываются новые теоретические направления моделирования систем (UML, VDM, RAISE, Z и др.), как развитие ООП. Представлены основные научные результаты, полученные по проекту РФФИ No16-01-00352 «Моделирование программных и операционных систем». Описываются базовые операции создания систем (link, make, weaver, config) из модульных элементов в ЯП (Algol, Cobol, Smalltalk, C++, Python, Java и др.) в общесистемной (IBM, WebSthereIBM, MS, .NET, Intel и др.) и в компиляторной (BSD, Java,

Linux, Grid, VAMOS и др.) средах. Определены парадигмы программирования предметных областей и перспективы из их развития.

2. The theory graph modeling systems from quality modules of the application areas (E. M. Lavrischeva, ASPSSE-2019. http://ceur-ws.org/Vol-2514/.- p.235-247)

Abstract: The graph modeling of applied systems (AS) from ready resources (modules) are presented. The graph is represented by an adjacency and reach ability matrix. A new program structures are modeling by mathematical operations (unions, connections, etc.). The assemble of programs structures (complex, system, packets, AS, OS, IS, etc.) from modules in Language Programming, as resources are integrating by such operations (link, make, weaver, config, etc.) and controlled on the quality every recourses and the making systems from them.

3. Технология сборки интеллектуальных и информационных ресурсов Интернет (Е.М. Лаврищева, А.К. Петренко. XXI Всероссийская научная конференция (23-28 сентября 2019 г., г. Новороссийск. — М.: ИПМ им. М.В.Келдыша, 2019. — С. 469-488. — http://keldysh.ru/abrau/2019/theses/93.pdf, doi:10.20948/abrau-2019-93.) Некоторые аспекты технологии сборки отражены и в презентации доклада на конференции ИСП РАН-2019 (http://0x1.tv/20191206AB).

Аннотация. Предлагается технология конфигурационной сборки интеллектуальных (Reuses) и информационных (data) ресурсов, которые разрабатываются для ТПС и накапливаются в библиотеках и хранилищах Интернет, в прикладные web-системы разного назначения. Представлены средства описания ресурсов, их верификация, тестирование, сборка и обеспечение безопасности и качества их взаимодействия. Обмениваемые между ресурсами данные трансформируются к требуемым форматам данных серверной среды Интернет с учетом стандарта ISO/IEC 11404 GDT и в соответствии с исходными данными компонентов повторного использования (КПИ), готовых сервисных и системных ресурсов Интернет. Приведена технология проверки правильности и оценивания показателей качества вариантов Веб-систем.

Ключевые слова: компонент, сервис, ресурс, система, веб-система, брокер сервисов, надежность, качество.

4. Technology of Assembly of intellectual and information resources Semantic Web Internet (E.M. Lavrischeva, A.K. Petrenko. http://keldysh.ru/abrau/2019/theses/93.pdf. doi:10.20948/abrau-2019-93)

Annotation. Technology offers intelligent configuration of the Assembly (Reuses) and information (data) resources developed and accumulated in the libraries and depositories of the Internet, applied web-based systems. Means of description, verification, testing, Assembly and security of interaction of resources are presented. The transmitted data is generated to the required data formats of the server environment taking into account the ISO/IEC 11404 GDT standard and converted to the source data of the client. Proposed configuration Assembly variation of the system of CPI and assessment of the required indicators of the quality of the resulting web system.

Keywords: component, service, resource, system, web-system, service broker, reliability, quality, Internet.

5. Technology of Assembly of intellectual and information resources Internet. **(E.M. Lavrischeva, A.K. Petrenko, and B.A.Pozin, SEUR-19)**

Annotation. The technology of Assembly of intellectual (Reuses) and information (data) resources which are developed and saved up in libraries and storages of the Internet, in applied web-systems of different function is offered. The intellectual tasks of subject areas of knowledge in the Semantic Web in mind service, software and system resources used in creating Web systems are presented. Methods of assembling resources (link, make, assemble, weaver, kconfig) are considered for modern environments. The problem of data exchange and generation to the formats of the global

network environment according to ISO/IEC 11404 GDT is defined. The concept of a network resource collector is proposed as a generalization of ways to assemble resources with functional security, resource protection and quality assessment of Web-systems. Perspective paradigms of programming of different areas of knowledge are defined.

Keywords: resource, service, web system, configuration, functionality, security, reliability, quality.

6. Теория графового моделирования сложных систем из модульных элементов для прикладных областей, Лаврищева Е.М., Доктор физ.мат.-наук, почетный профессор МФТИ, гнс ИСП РАН Научный журнал «Austrua Science", №28/2019, с.12-29.

Аннотация. Представлены математические основы графового моделирования прикладных систем, в вершинах которого находятся функциональные элементы (модули) систем, а дуги задают связи между ними. Граф представляется матрицей смежности и достижимости. Показан ряд графов программных структур и их представление математическими операциями (объединения, соединения, разности и др.). Дана характеристика графовых структур - программ, комплекса, агрегата и систем, создаваемых из модулей графа. Описан метод сборки ресурсов по графу разноязыковых модулей в первых языках программирования – ЯП для ОС ЕС IBM (1976-1987). Определяется научная теория связывания объектов по графу с использованием интерфейсов, содержащих операции обмена данными между объектами графа. Показан формальный переход к компонентам, сервисам сложных программных структур по графу с операциями сборки, интеграции (make, config, assembling, weaver и др.), функциями преобразования обмениваемых данных по стандарту ISO/IEC 11404-2007 в классе прикладных систем, работающих в среде Big Data, Cloud Computing Internet и обеспечения качества систем.

Ключевые слова: теория графов; матрица смежности, достижимости; математические операции объединение, проекция, следование; сборка, качество, преобразование данных.

7. Graph theory modeling complex systems module elements for the application areas, E. M. Lavrischeva, Scientific Journal «Austrua Science" info, №28/2019, p.12-29. http://austrua science/info.

Annotation. The basics of graph modeling of applied systems are presented, in the vertices of which the functional elements of the systems are located, and the arcs define the connections between them. The graph is represented as an adjacency and reach ability matrix. A number of graphs of program structures and their representation by mathematical operations (unions, connections, differences, etc.) are shown. Given the characteristics of graph structures, complexes, units, and systems created from the modules of the graph. The method of assembling the system on the graph of multilingual modules in the first programming languages – LP (Algol-60, Fortran, Cobol, PL/1, Smalltalk, etc.) for the IBM OS (1976-1985) is given. A scientific theory is determined by the OCM associate objects graph using objects of the interface containing operations for data exchange among objects of the graph. Transition to components, services and modeling of complex program structures on the graph with the advanced operations of assembly, association (make, config, assembling, weaver, etc.), functions of transformation of the exchanged data according to the ISO/IEC 11404-2007 GDT standard in the class of the applied systems working in the environment of Big Data, Cloud Computing Internet and quality assurance of this systems are shown.

Keywords: graph theory; adjacency matrix, reach ability; mathematical operations; assembling; quality; data transformation.

8. Методы оценки надежности программных и технических систем» с исполнителями данного проекта. Е.М. Лаврищева, С.В. Зеленов, Н.В. Пакулин, А.Г.

Рыжов, С.В.Морозов, О.А. Тарлапан. Труды ИСП РАН, том 31, вып. 5, 2019 г., стр. 95-108. DOI: 10.15514/ISPRAS-2019-31(5)-7
https://videonauka.ru/stati/30-metodika-prepodavaniya-tekhnicheskikh-distsiplin/238-analiz-metodov-otsenki-nadezhnosti-oborudovaniya-i-sistem-i-praktika-primeneniya-etikh-metodov

Аннотация. Определяются основные методы обеспечения и оценки надежности и безопасности программно-технических систем (ПТС) на процессах жизненного цикла (ЖЦ) и сбором сведений о возникающих в них ошибках, дефектах и отказах для последующих изменений. Рассмотрена стандартная модель надежности и дана характеристика базовых показателей, среди которых показатель надежности, функциональность и безопасность составляют основу измерения надежности. Приведена классификация моделей надежности, дана характеристика моделей оценочного типов, используемых при проверке показателей надежности компонентов ПТС. Показаны экспериментальные результаты применения оценочных моделей надежности к разным размерам программных компонентов ПТС и представлена оценка результатов измерения показателя надежности на этих компонентах с учетом плотности дефектов, интенсивности отказов и восстанавливаемости. Отмечается важность обеспечения надежности и безопасности систем в рамках новых стандартов (dependability and safety) в рамках интеллектуальных систем и Интернет вещей.

* Выполняется по проекту РФФИ 19-01-00206.

Ключевые слова: надежность, ошибка, дефект, отказ, дефект, безопасность, тестирование, надежность, риск, умные компьютеры.

9. Methods for assessing the reliability of software and hardware systems*. Lavrischeva E.M., Zelenov S.V., Pakulin N.V., A.G., Morozov S.V., Tarlapan O.A. Proceeding of the ISP RAS, Volume 31, ussue 5, 2019, pp. 95-108. DOI: 10.15514/ISPRAS-2019-31(5)-7

Annotation. The basic methods of ensuring and assessing the reliability and safety of software and hardware systems (PTS) on the life cycle processes (LC) and collecting information about errors, defects and failures arising in them for subsequent changes are determined. The standard model of reliability is considered and the characteristic of basic indicators among which the indicator of reliability, functionality and safety make a basis of measurement of reliability is given. Classification of models of reliability is given, the characteristic of models of the estimated types used at check of indicators of reliability of components of PTS is given. The experimental results of the application of reliability evaluation models to different sizes of software components of PTS are shown and the evaluation of the measurement results of the reliability index on these components taking into account the density of defects, failure rate and recoverability is presented. The importance of ensuring the reliability and safety of systems within the new standards (dependability and safety) in intellectual systems and Internet Thinks.

* Performed by the RFBR project 19-01-00206.

Keywords: reliability, error, defect, failure, defect, security, fault, completeness, testing, reliability, risks, smart computers.

10. The Theory Graph Modeling and Programming Paradigm Systems from Modules to the Application Areas – E.M. Lavrischeva Institute for System Programming of the Russian Academy of Sciences (ISP RAS, MIPT), Russia, DOI: https://doi.org/10.14738/tmlai.74.6782, Transactions on Machine Learning and Artificial Intelligence . Vol 12 No 6 (2019).

Abstract. The mathematical basics of graph modeling and paradigm programming of applied systems (AS) are presented. The vertices of graph are been the functional elements of the systems and the arcs define the connections between them. The graph is represented by an adjacency and reach ability matrix. A number of graph of program structures and their representation by mathematical operations (unions, connections, differences, etc.) are shown. Given the characteristics of graph structures, complexes, units, and systems created from the modules of the graph. The

method of modelling the system on the graph of modules, which describe in the programming languages – LP (Algol-60, Fortran, Cobol, PL/1, Smalltalk, etc.) and the advanced operations of association (assembling, make, weaver, config etc.). The standard of configuration (2012) Assembly of heterogeneous software elements in AS of different fields of knowledge is made. A brief description of modern and future programming paradigms for formal theoretical creation of systems from intelligent and cloud service elements of the Internet is given.. There is a new direction of nanotechnology in the near future.

Keywords: graph theory: adjacency matrix, reach ability; mathematical operations; assembling; projection; transformation.

11. The Theory Graph Modeling and Programming Paradigms of Systems from Modules to the Application Areas"- E. M. Lavrischeva, Doctor of phys.-мат.Sci., Professor of MIPT, General Sci. Specialist ISPRAS, Russia .Journal. Computer and Information Sciences – vol.12, №4 2019 (ISSN: 1913-8989, E-ISSN: 1913-8997)

Abstract. The mathematical basics of graph modeling and paradigm programming of applied systems (AS) are presented. The vertices of graph are been the functional elements of the systems and the arcs define the connections between them. The graph is represented by an adjacency and reach ability matrix. A number of graph of program structures and their representation by mathematical operations (unions, connections, differences, etc.) are shown. Given the characteristics of graph structures, complexes, units, and systems created from the modules of the graph. The method of modeling the system on the graph of modules, which describe in the programming languages (LP) and calling them with operations (link, assembling, building, etc.). The standard of configuration (2012) Assembly of heterogeneous software elements in AS of different fields of knowledge is made. Brief descriptions of modern and future programming paradigms for formal theoretical creation of systems from service-components for Internet in the near future are given.

Keywords: graph theory, adjacency matrix, reach ability, mathematical operations, configuration, Assembling, paradigm programming, future technologies

12. Лаврищева Екатерина Михайловна. Теория графового моделирования сложных систем из модулей для прикладных областей, Журнал Высшая школа (рус.) июль (14)2019, ISSN 2409-1677.-с.24-44.

Аннотация. Представлены математические основы графового моделирования прикладных систем. В вершинах графа располагаются функциональные элементы (модули) систем, а дуги задают связи между ними. На основе графа из модулей определяются сложные структуры (программа, комплекс, агрегат и др.) с помощью математических операций (объединения, соединения, разности и др.) и они представляются матрицей смежности и достижимости. Приводятся формальное связывание, сборка (link) модулей и их интерфейсов с операциями обмена данными между ними модулями и преобразования неэквивалентных данных в среде ОС ЕС IBM (1976-1987). Описан формальный переход от модулей к объектам и компонентам согласно парадигмы ОКМ, а также появления стандартной операции config (IEEE 828-2012, Configuration Managment) для получения выходного конфигурационного файла прикладной системы. Предложен теоретический аппарат формального преобразования неэквивалентных данных из класса общих типов данных – генерированных, неструктурных и агрегатных стандарта ISO/IEC 11404-2007, GDT к более простым фундаментальным. Сделан вывод о перспективном развитии теории графов и механизмов сборки применительно к прикладным областям (медицина, биология, генетика и др.).

Ключевые слова: теория графов; матрица смежности, достижимости; математические операции; объединение, проекция, следование; конфигурационная сборка; преобразование данных.

13. Graph theory modeling complex systems modules for the application areas

Annotation. The basics of graph modeling of applied systems are presented. In the vertices of graph are located the functional elements (modules) of the systems, and the arcs define the connections between them. On the basis of graph from modules are represented the program structures by mathematical operations (unions, connections, differences, etc.) and mathematical matrix of the adjacency and reach ability. Given the characteristics of graph structures, complexes, units, and systems created from the modules of the graph. The method of assembling the system on the graph of multilingual modules in the first programming languages – LP (Algol-60, Fortran, Cobol, PL/1, Smalltalk, etc.) for the IBM OS (1976-1985) is given. A scientific theory is determined by the OKM associate objects graph using objects of the interface containing operations for data exchange among objects of the graph. Transition to components, services and modeling of complex program structures on the graph with the advanced operations of assembly, association (make, config, assembling, weaver, etc.), functions of transformation of the exchanged data according to the ISO/IEC 11404-2007 GDT standard in the class of the applied systems working in the environment of Big Data, Cloud Computing Internet and quality assurance of this systems are shown.

Keywords: graph theory; adjacency matrix, reach ability; mathematical operations; assembling; configuration; nostructured big data; transformation.

Вывод по отчету проекта РФФИ №19-01-00206 -2019

Изложена теория и практика моделировании сложных программных, операционных и Веб-систем из готовых ресурсов - ГОР (reuses, модулей, объектов, компонентов, сервисов и др.) с оценкой качества отдельных программных элементов и ТПС из них. Описан подход к моделированию ТПС из интеллектуальных и информационных ресурсов Интернет, который может использоваться для создания систем в разных предметных областях. Основная операция сборки link/make/config функционирующая в системах IBMSphere, .Net, BSD, Grid определена как основная для отдельных ресурсов и является подготовительной для проведения конфигурационной сборки (ISO/IEC 826-92-2012 -configuration) для формирования отдельных вариантов ТПС. Сборка определена с общих позиций и предлагается как заявка на патент. После сборки проводятся процессы верификации моделей и тестирования отдельных программных элементов, а также вариантов ТПС. Результат полученного программного продукта оценивается на надежность и качество ТПС согласно описанной методики в п.5 отчета. Многие результаты исследований и реализаций по данного проекту опубликованы и доступны для применения в конкретной деятельности по проектированию программных и информационных систем.

Новые публикации 2019 года:

1. Доклад Лаврищева Е.М.Моделирование сложных систем - «Научный сервис - 2019 http://0x1.tv/20191206AB); Актуальные проблемы систем и программного обеспечения» - ASPSSE-2019 http://ceur-ws.org/Vol-2514/ ; -p.235-247. ISPRAS OPEN -2019 http://0x1.tv/20191206AB); http://0x1.tv/20180517F; http://0x1.tv/20181122AF.

– XXI Всероссийской научной конференции (23-28 сентября 2019 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2019. — С. 469-488. — URL: http://keldysh.ru/abrau/2019/theses/93.pdf doi:10.20948/abrau-2019-93. Размещена презентация к докладу;

– Научный журнал «Austrua Science", №28/2019, с.12-29 - Теории графового моделирования систем в Международных журналах «Computer and Information Sciences» – vol.12. №4 2019 и TMLAI Vol 12 No 6 Dec 2019 – "The Theory Graph Modeling and Programming Paradigms of Systems from Modules to the Application Areas".

2. Доклады и публикации являются перспективными в области теории программирования систем и ТПС**.** Они вызывают интерес у специалистов разных стран мирового сообщества по видео в Интернете.

http://0x1.tv/20191206AB,
http://0x1.tv/20180517F;
http://0x1.tv/20181122AF.
https://videonauka.ru/stati/30-metodika-prepodavaniya-tekhnicheskikh-distsiplin/238-analiz-metodov-otsenki-nadezhnosti-oborudovaniya-i-sistem-i-praktika-primeneniya-etikh-metodov.
http://keldysh.ru/abrau/2019/theses/93.pdf doi:10.20948/abrau-2019-93
http://keldysh.ru/abrau/2019/theses/93.pdf doi:10.20948/abrau-2019-93

3. Лаврищева Е.М., Петренко А.К. Технология сборки интеллектуальных и информационных ресурсов Интернет // Научный сервис в сети Интернет: труды XXI Всероссийской научной конференции (23-28 сентября 2019 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2019. —С. 469-488. — URL: http://keldysh.ru/abrau/2019/theses/93.pdf doi:10.20948/abrau-2019-93

Размещена презентация к докладу
Соглашения: https://drive.google.com/open?id=1Sind_wVjT5v5lP4HiR65oViDAqc0I9GG
Тексты статей: https://drive.google.com/open?id=1YF506xO2Zp05phnYbzxIxM_iyDXg1AGg.

Раздел 5. Форма 505. Проект РФФИ 19-01-00206-2020. Методы оценки надежности и безопасности технических и программных систем

Е.М. Лаврищева, В.С. Мутилин, Зеленов С.И., Козин С.В.

В рамках проекта №19-01-00206-2020 членами проекта проводились исследования и разработка методов обеспечения надежности, качества и безопасности систем, создаваемых по методу моделирования сложных систем. Ранее эти методы отрабатывались на технических и программных средствах в проекте ВПК (Липаев В.В.) и по программе 1.1.6 ГКНТ ВТ «Технология программирования СЭВ» 1984-1987. В результате создаваемые средства тестировались и оценивались на качество и надежность их в устройствах и средствах военного назначения. Под руководством Липаева В.В. был создан проект стандарта качества в рамках СЭВ и одобрен на конференции «Интерфейс-СЭВ 1987». Но договор со странами СЭВ прекратил свое сушествовани в 1989г. и стандарта качества СССР-СЭВ не был сделано.

В 2018 на конференции ИСП РАН OS DAY-18 Лаврищевой Е.М. (Пакулин Н.А) сделан доклад «Методы и средства обеспечения надежности технических ипрограммных средств». Эти методы были отработаны Пакулиным Н.А. в системе реального времени и в трудах ИСП РАН были опубликованы статьи на эту тему:

1. http://0x1.tv/20181122AF ИИ, http://0x1.tv/20180517F. - качество

Пакулин Н.В., Лаврищева Е.М. Анализ методов оценки надежности оборудования и систем. Практика применения методов. 5 Научно-практическая конференция - OS DAY, Москва, 17-18мая 2018. Опубликована статья по докладу с участием Рыжова А.Г., Зеленова С. на анг. языке.-Труды ИСП РАН, том5 DOI: 10.15514 ISPRAS-2018-30(3), 2018.-р. 99-120.

Lavrischeva E. M., Pakulin N.V., Ryjov A.G., Zelenov S. V. (2018, 17-18 October). Analysis of methods of Assessment reliability of equipment and systems. Practice of application of methods of reliability.- Scientific- practical conference - OS DAY, Moscow,. The proceedings of ISP RAS, Tom 5. DOI: 10.15514/ISPRAS-2018-30(3), (http://0x1.tv/20180517F).

2. E.M. Lavrischeva, V.S. Mutilin, A.G. Ryzhov. Designing variability models for software, operating systems and their families, 2017, Сборник ИСП РАН, 2017, issue 5, 2017, pp.93-109.- DOI: 10.15514/ISPRAS- 2017(5).

Теоретики, изучая природу ошибок в системах, разработали более 100 математических моделей надежности, основанных на учете различных ситуаций, возникающих в технических и программных системах на первых и последующих поколениях ЭВМ. Методы надежности обеспечивают повышение надежности систем путем исправления разного рода обнаруженных ошибок в процессе разработки и их эксплуатации.

Надежность систем сформировалась как самостоятельная теоретическая и прикладная наука, способствующая определению качественных показателей систем (функциональность, точность, отказоустойчивость, гарантоспособность, завершенность и др.).

Методы оценки надежности систем позволяют прогнозировать, измерять и оценивать качество продукта с учетом возникающих ошибок, количества и интенсивности отказов, а также процессов разработки отдельных компонентов систем в жизненном цикле (ЖЦ).

В работе рассматриваются все аспекты обеспечения надежности, безопасности и качества технических и программных систем.

5.1. Методы надежности оборудования и систем

Методы оценки надежности технических систем (аппаратуры, устройства, оборудование и др.) были разработаны значительно ранее компьютерных систем и основывались на вероятностных Марковских процессах с множеством числа состояний по теории массового. Эти методы обеспечивали проверку надежности функционирования техники, приборов и устройств в различных областях (машиностроение, энергетика, космос, медицина и др.). На их работоспособность влияли неисправности и разные недоработки в конструкции, приводящие к разрушительным последствиям и к ущербу системы в целом.

На оценку надежности компьютерных систем и ПО существенным образом влияют следующие особенности.

1. Большое количество кода в ПО, зачастую превышающее емкость физических элементов ЭВМ, и способов взаимодействия отдельных элементов ПО между собой;
2. Нематериальный характер элементов ПО, которые не деградируют, но стареют во времени, и в их процессах, программах и конструкциях могут случаться разного рода непредвиденные ситуации;
3. Возникновение ошибочных ситуаций, разных дефектов и отказов как в задании формальной спецификации отдельных элементов, так и в их выходном коде;
4. Элементы ПО трудно поддаются визуализации, обнаружению и коррекции найденных ошибок, поэтому измерение надежности ПО требует анализа и проверки данных об ошибках в большей степени, чем для аппаратуры;
5. Системы ПО могут изменяться при функционировании и выходить из рабочего состояния от разных ситуаций внешней среды (вирусов, атак и др.), которые не предусмотрены в соответствующих моделях надёжности и обеспечиваются методами безопасности информации и систем.

Надежность технических систем зависит от следующих факторов:

– качества отдельных технических конструктивных элементов системы;

– отсутствия неисправностей в конструктивных элементах и их способность работать надежно и качественно;

– безопасность, отказоустойчивость и защита обрабатываемой информации.

Надежность программных систем зависит от этих же факторов и от случайных изменений данных и маршрутов исполнения программ, которые могут привести к неверным результатам или отказам.

Между надежностью аппаратуры и ПО систем имеется сходство, состоящее в возникновении случайных явлений в процессах и системах, которые должны анализироваться методами теории вероятности, надежности и безопасности.

5.2. Определение понятия надежности и безопасности систем

Теории надежности появилась в 1945 году после окончания 2- мировой войны. Военное ведомство США начало интенсивно финансировать работы, связанные с качеством вооружений самого разного назначения. В СССР первые работы по надежности были инициированы ВПК (военно-промышленным комплексом) страны, связанные с обеспечением качества военной техники в авиационной, космической и морской областях.

В 1959 году была создана Международная федерация по обработке информации (International Federation for Information Processing (IFIP). Руководящим органом IFIP является Генеральная ассамблея. Через каждые три года IFIP созывает конгрессы. В качестве вице-председателя Программного комитета и руководителя одной из десяти секций («Программное обеспечение») был представитель бывшего СССР А.П. Ершов, который принимал активное участие в подготовке и работе конгрессов IFIP (1971, 1974, 1980 гг. и др.), проходивших в разных странах мира.

В классической теории надежности определен термин надежность (dependability, 1986)* - свойство объекта сохранять все параметры, обеспечивающие техническое обслуживание, хранение и транспортирование. Данный термин обозначает отказоустойчивость, готовность, живучесть, обслуживаемость, достоверность, информационную безопасность (целостность, конфиденциальность). Техническое средство (ТС) – это отказоустойчивая, высоконадежная, безопасная и живучая система с гарантированными достоверными вычислениями. Для сервис-ориентированных КС Интернет гарантоспособность состоит в предоставлении услуг для вычисления требуемых функций со свойствами безотказности, готовности, долговечности, ремонтопригодности и сохраняемости.

Примером создания таких ТС в СССР были авиация, космос, энергетика, военная техника, межконтинентальные баллистические ракеты, боевые самолеты, вертолеты и т.п. При этом вопросы отказоустойчивости, 1950г. безопасности, живучести (надежности), защиты - являются актуальными и очень важными.

Президентская директива США PDD-63 от 22.05.1998 г. «Программы гарантирования всесторонней защиты инфраструктур» (Defense – wide Information Assurance Program - DIAP) активно поддерживается оборонными ведомствами ФБР и НАТО. К ней присоединились и страны Северной Америки, Евросоюза, Японии и др.

В IFIP в 1970 г. создан Технический комитет по отказоустойчивым вычислениям и Рабочей группы (WG 10.4) «Гарантоспособные вычисления и отказоустойчивость». «Гарантоспособность. Основные определения и терминология» (1992 г.) изложил J.C. Laprie. В Брюсселе в 1991 году была опубликованы «Критерии оценки безопасности информационных технологий (ITSEC)», принятые Германией, Францией, Великобританией, Италией и США. Эти критерии постоянно совершенствуются и согласовываются ведущими научно-исследовательскими центрами правительственных и международных структур.

Центр надежных и высокоэффективных вычислений Иллинойского университета (Center for Reliable and High - Performance Computing University of Illinois) модифицировал программный продукт Chameleon, который обеспечивает необходимый уровень функциональной надежности КС с сетевой структурой. С помощью администратора отказоустойчивости (Fault – Tolerance Manager) обеспечивается адаптация составляющих системы с разными уровнями готовности в однородной, гетерогенной и кластеризованной вычислительных средах. Департамент компьютерных наук университета в Теннесси (Department of Computer Science University of Tennessee) разработал продукт для отказоустойчивых сетевых вычислений NetSolve в клиент-серверной архитектуре. Модульность продукта дает возможность вести гарантоспособные вычисления на двух уровнях: внутреннем и межсерверном. Ведущими учреждениями в области гарантоспособности являются: Лаборатория реактивного движения (JPL) Калифорнийского технологического института (США, Корпорация электронных систем коммутации Bell System (США, 1953 г.) AT&T, компания IBM с исследовательским центром им. Дж. Уотсона (США), Лаборатория им. Ч. Дрейпера Массачусетского технологического института (США), Международный Стэнфордский научно-исследовательский институт (США), корпорация Boing (США), Лаборатория автоматики и системного анализа (LAAS) Тулузского научно-исследовательского центра (Франция), компания Schneider Electric (Франция), компании SRC и CSR (Англия), компания Siemens (ФРГ) и другие.

В работах наиболее известных зарубежных авторов Avizienis A., Laprie J.-C., Randell B., Landwehr C., Dobson I.E и др. приводятся обобщенные результаты исследований в области гарантоспособных КС. Однако коммерческие тайны учреждений, в которых они работают, и специфика применений их разработок в закрытых тематиках, финансируемых военными ведомствами, не дают полных и конкретных сведений об особенностях использования методов и средств, которые можно было бы напрямую имплантировать в проектирование конкретных гарантоспособных систем.

Под надежностью систем понимается способность системы сохранять свои свойства (безотказность, восстанавливаемость, защищенность и др.) на заданном уровне в течение фиксированного промежутка времени при определенных условиях эксплуатации. Термин надежность (reliability) обозначает способность системы обладать свойствами, обеспечивающими выполнение функций системы в соответствии с заданными требованиями.

С 2004 году ассоциация IEEE издает журнал Dependable and Security Computing. В нем обсуждаются бизнес-критические приложения, электронная коммерция, банковские технологии и др. С точки зрения гарантоспособности надежность является целевой функцией реализации системы. К ней предъявляются высокие требования (недопустимость ошибок, отказов, дефектов и других аварийных ситуаций). Надежность систем зависит от числа оставшихся и не устраненных ошибок в отдельных программах и компонентах системы.

Гарантоспособная КС (ГКС) – это система, обладающая полным или частичным набором первичных свойств (атрибутов), составляющих гарантоспособность. Иными словами, ГКС – это отказоустойчивая, высоконадежная, безопасная и живучая система с гарантированно достоверными вычислениями. Для сервис-ориентированных КС можно сформулировать гарантоспособность как способность КС предоставлять требуемые услуги, которым можно оправданно доверять. Гарантоспособность означает способность системы гарантировать выполнение услуг, для которых она предназначена, и состоит в:

– безотказности выполнения;

– готовности к работе;

– достоверности результатов;

– приспособленности к обслуживанию или ремонту (maintainability);

– информационной безопасности (security);

– конфиденциальности (confidentiality), секретности и целостности информации (integrity);

– безопасности (safety) работы системы без катастрофических последствий;

– эксплуатационной завершенности ПО и способности к восстановлению работоспособности системы.

Отказоустойчивость (fault-tolerance) обозначает способность системы автоматически за ограниченное время прогнозировать, предупреждать и восстанавливать функциональность системы после отказов с помощью механизмов поддержки всех составляющих гарантоспособности. Системы или процессы, которые обладают таким комплексным свойством, называют гарантоспособными. Им присущи традиционные надежностные свойства (безотказность, готовность, безопасность, целостность, конфиденциальность, восстанавливаемость и др.).

Вопросы разработки и использования гарантии качества систем обсуждаются более 25 лет на международных форумах и конференциях (Conference on Dependable Systems and Networks (DSN), European Dependable Computing Conference, (EDCC), International Conference on Computer Safety, Reliability and Security (SAFECOM), Probabilistic Safety Assessment and Management Conference (PSAM), Dependable Systems, Services and Technologies (DESSERT), Conference on Dependability of Complex Systems (DepCoS) и др.).

Сложность и масштабность решения проблемы обеспечения гарантоспособности КС ставит принципиально новые задачи по организации существенно более эффективных отказоустойчивых архитектурных решений, реализации новых подходов по обеспечению высокой надежности, живучести и безопасности.

Актуальность исследований в области гарантоспособности обосновывается, прежде всего, возможными объемами материального и морального ущерба, причиняемого отказами функционирования КС в составе критических технологий и инфраструктур. Имеются примеры техногенных аварий и катастроф в различных областях применения КС в авиации и космосе (катастрофы многомоторных самолетов и космических аппаратов) до энергетики (аварии на атомных и гидроэлектростанциях) и военной техники.

В связи с задачами Проекта РФФИ 19-01-00206, ставящим своей целью создания сложных Веб-систем их сервисных ресурсов, ставится задача исследование использования веб-сервисов и GRID-технологий и проведения анализ рисков с помощью FPGA и ASIC-технологий. Для обеспечения надежность программных компонент и систем использовать вероятностные методов метрической оценки надежности и качества систем, а также микро-контроллеров.

Чем интенсивнее проводится эксплуатация системы, тем интенсивнее выявляются ошибки, быстрее растет надежность системы и соответственно ее качество. Надежность, по существу, является функцией от ошибок, оставшихся в системе после ввода ее в эксплуатацию. Системы без ошибок считаются абсолютно надежными. Для оценки надежности систем используются собранные статистические данные – время безотказной работы, дефекты и частота (интенсивность) отказов.

Исследование надежности систем проводится с помощью методов теории вероятностей, математической статистики, теории массового обслуживания и математических методов надежности и безопасности. Главным источником информации для оценки надежности являются процессы тестирования, эксплуатации и испытания системы и данные, полученные при разработке систем в соответствии со стандартами жизненного цикла (ЖЦ) (ISO/IEC 15846-1998, 15939:2002) системной инженерии.

В нашей стране работы по обеспечению защиты. безопасности, качества проводились в рамках ВПК под руководством В.В.Липаева*.

Теория гарантоспособных систем постоянно развивается с учетом современных достижений в элементной базе и технологий производства компонентов КС, автоматизации проектирования, сетевых технологий, методов технической диагностики, прогнозирования неисправностей, моделирования математического аппарата теории надежности и др. В настоящее время данная теория обсуждается в статьях и докладах на конференциях:

Avizienis A. Dependable Computing: From Concepts to Application / A. Avizienis, J.-C. Laprie // IEEE Trans. On Computers. – 1986. – N 74 (5). – P. 629 – 638.

Basic Concepts and Taxonomy of Dependable and Secure Computing / A. Avizienis, J.-C. Laprie, B. Randell [et al.] // IEEE Trans. on Dependable and Secure Computing. – 2004. – Vol. 1, N 1. – P. 11 – 33.

Липаев В.В. Качества программного обеспечения.-Синтег.-2011. IX international conference “Strategy of Quality in Industry and Education”. – Varna, Bulgary, 2013. – Vol. 1. – 396p.

5.3. Базовые понятия моделей надежности и безопасности

К базовым понятиям, которые используются в моделях надежности систем, относятся следующие:

Отказ (failure) – это переход системы из рабочего состояния в нерабочее.

Дефект (fault) – это последствие выполнения элемента программы, приводящее к некоторому непредвиденному событию (неверной интерпретации компьютером); невыявленные дефекты – источник потенциальных ошибок и отказов системы.

Ошибка (error) может быть следствием недостатка в спецификациях любой из программ или при принятии неверных действий в процессе испытания системы.

Интенсивность отказов – это частота появления отказов или дефектов в системе при ее тестировании, эксплуатации и сопровождении системы.

Одним из подходов к исследованию надежности на основе отказов систем является классическая теория вероятностей, согласно которой отказы в системе считаются случайными и зависят от ошибок, внесенных при разработке системы. Все модели оценки надежности базируются на статистике отказов и интенсивности выявленных отказов в

процессе верификации, тестирования и испытания системы для обеспечения ее работоспособности и гарантоспособности.

Модели надежности основываются на нахождении случайной величины в системе, числом и интенсивностью возникновения отказов в системе. Для случайной величины вычисляется математическое ожидание и дисперсия (среднее отклонение). Если случайная величина дискретна, т.е. принимает конечное число значений x1, x2, …, xn , то ее распределение описывается вероятностью $P(\xi = xi)$, и в общем случае $F(x) = P(\xi < xi)$ является функцией распределения случайной величины.

Случайный процесс с непрерывным временем, который описывается однородными событиями, называется пуассоновским процессом.

Если случайные величины, полученные на всем ЖЦ системы, распределены по показательному, эрланговскому или гиперэрланговскому законам, то поведение системы описывается Марковским процессом без непрерывных компонентов.

Надежность по существу очень близка задачам безопасности. При разработке систем научными и некоммерческими институтами их трудно оценить на надежность и безопасность из-за того, что они делаются, как правило, не по стандартам. В то время как системы управления авиацией, атомной энергетикой и оборонной промышленностью разрабатываются по стандартам. В них надежность и безопасность определяют работоспособность системы в соответствии с требованиями и с минимум отказов и дефектов.

В характеристиках безопасности учитываются только те отказы, которые могут привести к катастрофическим последствиям и ущербам (например, пожар, взрыв, разрушение здания и др.). Оценка безопасности системы базируется на надежности функционирования ПО и БД. Оценка надежности зависит от метрик стандарта качества (внешние, внутренние, эксплуатационные). Они сравниваются с требованиями заказчика на систему и используются при сертификации продукта. Для оценки надежности и функциональной безопасности используются стандарты ISO/IEC 12207 для ПО и ISO 15288 -2006 систем.

На работоспособность системы влияют отказы, дефекты и ошибки проектирования, которые приводят к длительности восстановления и к необходимости устранять в программе ошибки средствами программной и информационной избыточности. Согласно стандарта ISO 9126 (1-4) определяются характеристики надежности с учетом обнаруженных дефектов и ошибок при функционировании гарантоспособного ПО систем.

Степень работоспособности/гарантоспособности зависит от соответствия характеристик требований, заданных в проекте, выявленным ошибкам и отказам в ПО и возможным неисправностям в конструктивных элементах систем.

Анализ рисков безопасности для ПТС

Традиционно оценка безопасности системы производится инженерами вручную на основе анализа различных неформальных требований и проектных документов. Такой подход, во-первых, делает оценку безопасности субъективной и серьезно подверженной человеческому фактору, начиная от уровня квалификации инженера и заканчивая состоянием его здоровья. Во-вторых, при использовании такого подхода для оценки безопасности достаточно крупной системы со сложной архитектурой возникает необходимость привлечения для проведения оценки безопасности не одного инженера, а достаточно большой команды инженеров, что влечет за собой необходимость четкой организации коммуникации внутри команды и тщательной интеграции результатов работы каждого инженера по оценке безопасности частей системы в единую картину, а это чрезвычайно усиливает риск потери качества оценки безопасности системы в целом.

В настоящее время при проектировании сложных систем критических по безопасности большие преимущества демонстрирует подход, основанный на построении формальных моделей и спецификаций разрабатываемой системы. Одним из ключевых преимуществ

модельного подхода является возможность полностью автоматизировать анализ требований, формализованных в виде моделей и спецификаций, что позволяет, во-первых, обеспечить объективность анализа, а во-вторых, чрезвычайно повышает скорость и качество анализа.

При использовании для формализации архитектуры разрабатываемой системы языка AADL (Architecture Analysis and Design Language (SAE International standard AS5506C)), требования по безопасности этой системы формализуются с использованием специального расширения этого языка – Error Model Annex. Требования по безопасности системы формализуются с привязкой к AADL-описанию архитектуры системы, т.е. с привязкой к конкретным компонентам и соединениям внутри системы, описанным на AADL. Основными средствами описания требований по безопасности в Error Model Annex являются:

– описание взаимовлияния компонентов системы посредством распространения ошибок между различными типами компонентов;

– описание штатных и нештатных состояний компонентов и переходов между состояниями в виде конечного автомата;

– иерархическая композиция моделей ошибок подкомпонентов для описания модели ошибок компонента, составленного из этих подкомпонентов.

Анализ дерева неисправности

Одним из основных инструментов при проведении оценки безопасности системы является анализ дерева неисправности (ГОСТ Р 27.302-2009). Этот вид анализа относится к методам анализа “сверху-вниз”, когда последовательно рассматриваются все более детальные (т.е. более нижние) уровни конструкции.

Анализ дерева неисправности - дедуктивный анализ нарушения, который сосредотачивается на одном специфическом нежелательном событии и предоставляет метод для определения причин этого события. Выполнение анализа начинается с нежелательного события верхнего уровня и затем систематически определяются все вероятные одиночные отказы и комбинации отказов функциональных блоков системы на следующем нижнем уровне, которые могут вызывать это событие. Анализ развивается вниз, последовательно через более детальные (то есть нижние) уровни конструкции объекта, до достижения первичного нераскрываемого события или до получения подтверждения, что требования к нежелательному событию верхнего уровня удовлетворяются. Первичное событие определяется как событие, которое по той или другой причине далее не разрабатывается (то есть событие не нуждается в разбиении на более точные уровни детализации для показа соответствия применяемым к анализируемой системе требованиям по безопасности). Основное событие может быть внутренним или внешним по отношению к анализируемой системе и может быть отнесено к отказам/ошибкам аппаратных средств или к ошибкам программного обеспечения.

Графическое представление дерева неисправности иерархично и получило наименование из-за показываемый ветвлений. Это формат, который делает результаты анализа наглядными и для разработчиков, и для авиационной власти. Как один из семейства методов оценки безопасности, используемых для гарантии выполнения системой/оборудованием установленных функции безопасности, анализ дерева неисправности связан с гарантией того, что аспекты безопасности конструкции идентифицированы и контролируемы.

Применение анализа дерева неисправности обеспечивает:

Помощь при выполнении оценок и рассмотрений техническими и сертифицирующими органами. (Завершенное дерево неисправности показывает только отказные события, которые могли в отдельности или в комбинации привести к установленному нежелательному событию верхнего уровня).

Оценку влияния модификации конструкции на безопасность.

Определение значения вероятности встречаемости события верхнего уровня.

Распределение бюджетов вероятности по событиям нижних уровней.

Наблюдаемость вклада ошибок конструкции посредством обеспечения формата для смешанных количественной и качественной оценок.

Оценку последствий одиночных и множественных отказов.

Оценку интервалов воздействия, скрытого состояния и интервалов "риска" в отношении их общего влияния на систему.

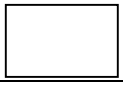
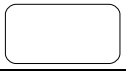
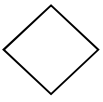
Наблюдаемость потенциальных границ событий с общей причиной.

Оценка источников отказов с общей причиной.

Оценка свойств "отказобезопасности" конструкции (устойчивая к отказам и устойчивая к ошибкам).

Анализ дерева неисправности использует булевские логические символы для демонстрации связи последствий отказа с видами отказа. Двумя наиболее общими логическими символами являются И-символы и ИЛИ-символы. И-символ представляет условие, когда требуется одновременное наличие всех входных событий для получения выходного события более высокого уровня. ИЛИ-символ представляет условие, когда для получения выходного события более высокого уровня необходимо одно или более входных событий.

Таблица 1. Дерево неисправностей визуализируется посредством узлов вида.

Символ	Описание
	Событие, которое происходит, если произошли одно или более события следующего нижнего уровня (в зависимости от нижележащих условий)
	Условие И: Событие происходит тогда и только тогда, когда произошли все события следующего нижнего уровня
	Условие ИЛИ: Событие происходит тогда и только тогда, когда произошло любое одно событие следующего нижнего уровня
	Показывает, что поддерево для данного узла идентично поддереву, расположенному под треугольником с тем же идентификатором
	Первичное внутреннее в анализируемой системе событие, не требующее дальнейшего раскрытия.
	Первичное событие, которое далее не раскрывается, потому что оно имеет незначительное влияние на события верхнего уровня или потому что подробности, необходимые для дальнейшего раскрытия события, трудно доступны

Набор минимального сечения дерева неисправностей – это такое множество первичных событий, наступление которых влечет наступление события верхнего уровня, однако ни для какого его собственного подмножества это свойство не выполняется.

Определение наборов минимального сечения производится следующим образом. Для дерева неисправностей строится соответствующая булевская формула, которая затем приводится к дизъюнктивной нормальной форме (ДНФ) путем применения операций булевой алгебры, а также посредством сокращения формулы на оспове свойств этих операций, в частности свойств идемпотентности и законов поглощения. В результате каждая конъюнкция полученной ДНФ соответствует одному набору минимального сечения исходного дерева неисправностей.

Ранжирование первичных событий в соответствии с их значимостью в смысле наступления события верхнего уровня может производиться несколькими методами. Пусть S – событие верхнего уровня, E – первичное событие, $\bar{E}$ – отрицание E, p – функция вероятности событий. Если A и B – два события, то p(A|B) – это вероятность наступления события A при условии, что наступило событие B.

Значимость «стоимость возрастания риска»: p(S|E) / p(S). Стоимость возрастания риска показывает, насколько увеличивается риск при условии, что событие E происходит постоянно (т. е. Соответствующий компонент абсолютно не надежен). Это индикатор, насколько важно поддерживать текущий уровень надежности соответствующего компонента.

Значимость «стоимость уменьшения риска»: p(S) / p(S|$\bar{E}$). Стоимость уменьшения риска показывает, максимальное возможное уменьшение риска, которое можно ожидать при увеличении надежности соответствующего компонента (вплоть до его абсолютной надежности, когда событие E никогда не происходит.

Значимость по Бинбауму: $\partial p(S)/\partial p(E)$ = p(S|E) - p(S|$\bar{E}$). Эта величина показывает, в какой степени изменение надежности E влияет на повышение надежности S.

Значимость по Фусселу-Весли: (p(S) - p(S|$\bar{E}$)) / p(S). Эта величина показывает степень критичности события E, т.е. относительную величину вклада события E в величину вероятности наступления S.

Значимость «важность диагностики»: p(E|S) = p(E)·p(S|E) / p(S). Эта величина показывает вероятность сбоя E при условии сбоя S, т.е. насколько приоритетно проверять, был ли сбой E, при возникновении сбоя S.

Марковский анализ

Одним из методов оценки безопасности программно-аппаратных систем является марковский анализ.

Вероятностный или стохастический автомат — это конечный автомат, в котором нет входных и выходных символов, а переходы между состояниями осуществляются с заданной вероятностью.

Исследуемая на предмет безопасности система может быть описана в виде вероятностного автомата, содержащего особые состояния, соответствующие отказам. Вероятность перехода в такое состояние — это вероятность возникновения отказа.

Во многих случаях можно считать, что будущее технической системы определяется ее текущим состоянием, так что построенный вероятностный автомат является *марковской цепью* и к нему применимы все результаты теории марковских цепей.

Построение марковской цепи

Для данной модели, состоящей из компонентов $c_1, \ldots, c_N$, рассмотрим следующий вероятностный автомат. Состояниями этого автомата являются кортежи состояний всех компонентов $s_1 \times \ldots \times s_N$. Переход автомата в другое состояние происходит при возникновении некоторого набора внутренних сбоев в компонентах. Для данного состояния s автомата и данного набора внутренних сбоев $\{e_{i,j}\}$ итеративно анализируется распространение сбоев между компонентами и переходы компонентов в новые состояния до тех пор, пока для модели не будет достигнуто такое состояние s', в котором модель стабилизируется. Таким образом в вероятностном автомате имеется переход $s \rightarrow s'$ с вероятностью $\prod_{i,j} p(e_{i,j})$.

Заметим, что если в модели имеется N компонентов и L различных внутренних сбоев, причем каждый компонент может находиться не менее чем в двух состояниях («рабочее», «нерабочее», а также, возможно, несколько промежуточных состояний), то вероятностный

автомат будет содержать не менее чем 2^N состояний, и из каждого состояния будет выходить 2^L переходов.

Однако, на практике не все состояния вероятностного автомата являются достижимыми из некоторого начального состояния. Для заданного начального состояния модели вероятностный автомат строится постепенно, по мере построения переходов из уже достигнутых состояний. В результате количество полученных состояний оказывается существенно меньше экспоненты.

Кроме того, в реальных системах отдельные сбои имеют довольно низкую вероятность, порядка 10^{-4} .. 10^{-12}. Соответственно, вероятность одновременного возникновения нескольких сбоев является исчезающе малой величиной. Таким образом, на практике наборы сбоев $\{e_{i,j}\}$ можно ограничить парами или тройками. В результате количество переходов из каждого состояния вероятностного автомата будет ограничено полиномом степени 2 или 3.

Построение системы дифференциальных уравнений и граничных условий для задачи Коши

Для всех марковских процессов верно уравнение Колмогорова-Чепмена:

$$P^{(t+dt)}(S_j/S_i)=\sum_{k=1}^{n} P^{(dt)}(S_k/S_i)P^{(t)}(S_j/S_k)$$

Данное уравнение означает, что вероятность перехода из состояния S_j в состояние S_i за некоторое время $t+dt$ равна сумме вероятностей попаданий в целевое состояние S_i через промежуточные состояния S_k .

Рассмотрим стационарную модель, в которой интенсивность перехода между состояниями S_i и S_j равна $\lambda(S_i/S_j)$. Тогда для систем с непрерывным временем из уравнения Колмогорова-Чепмена вытекает система дифференциальных уравнений вида

$$\frac{dP^{(t)}(S_j/S_i)}{dt}=-\sum_{k=1}^{n}\lambda(S_i/S_k)P^{(t)}(S_j/S_i)+\sum_{k=1}^{n}\lambda(S_k/S_i)P^{(t)}(S_j/S_k)$$

Пусть S_1 -- некоторое начальное состояние системы. Рассмотрим полученные дифференциальные уравнения для случая $S_1=S_j$. Пусть $P_i(t)$ обозначает функцию $P^{(t)}(S_1/S_i)$. Тогда уравнения примут следующий вид:

$$\frac{dP_i(t)}{dt}=-\sum_{k=1}^{n}\lambda(S_i/S_k)P_i(t)+\sum_{k=1}^{n}\lambda(S_k/S_i)P_k(t)$$

Кроме того, появляются следующие граничные условия:

$$P_1(0)=1, P_i(0)=0, i=\overline{2,n}$$

Таким образом, мы получили задачу Коши.

Анализ видов последствий и критичности отказов

Одним из методов оценки безопасности программно-аппаратных систем является анализ видов и последствий отказов (ГОСТ 27.310-95).

Анализ видов и последствий отказов является анализом “снизу вверх”, когда анализ стартует с некоторого сбоя в отдельном компоненте и определяет, к каким последствиям может привести этот сбой. При проведении этого анализа:

выявляют возможные сбои компонентов и системы в целом, изучают их причины, механизмы и условия возникновения и развития;

определяют возможные неблагоприятные последствия возникновения выявленных отказов, проводят качественный анализ тяжести последствий отказов и/или количественную оценку их критичности.

Критичность отказов оценивают с использованием показателей, учитывающих для каждого анализируемого отказа:

вероятность его возникновения за время эксплуатации;

условные вероятности наступления всех возможных неблагоприятных последствий отказа;

размер возможного ущерба в результате наступления каждого из ожидаемых последствий отказов.

Инструментальная поддержка анализа рисков

В рамках проекта РФФИ в ИСП РАН реализованы следующие методы анализа надежности технических систем:

- Анализ дерева неисправностей методами логико-вероятностного анализа.
- Марковский анализ методами теории марковских процессов.
- Анализ видов последствий и критичности отказов.
- Анализ надежности технической системы производится на основе формального описания системы на языке AADL.

В рамках анализа дерева неисправностей:

- для заданного отказа целевого компонента автоматически строится дерево неисправностей;
- автоматически вычисляются минимальные сечения;
- автоматически вычисляется вероятность возникновения заданного отказа;
- атомарные отказы подкомпонентов автоматически ранжируются по мере значимости относительно величины вклада в вероятность возникновения заданного отказа.

В рамках марковского анализа:

- автоматически строится марковская цепь;
- автоматически составляются уравнения Колмогорова-Чепмена и формулируется задача Коши;
- автоматически численными методами находится решение задачи Коши;
- автоматически вычисляется вероятность отказа целевого компонента в заданный момент времени.

Решение полученной задачи Коши и нахождение вероятностей отказов данной (под)системы в заданные моменты времени осуществляется численно методом Рунге-Кутты.

В рамках анализа видов последствий и критичности отказов:

- для заданной комбинации отказов целевых компонентов автоматически вычисляются ее возможные причины и их вероятности;
- для заданной комбинации отказов целевых компонентов автоматически вычисляются ее последствия;
- для заданной комбинации отказов целевых компонентов автоматически вычисляется ее критичность для системы в целом.

Поддерживаются две разновидности анализа видов и последствий отказов:

1. исходными сбоями считаются внутренние ошибки компонентов;
2. исходными сбоями считаются нерабочие состояния компонентов.

В первом случае (когда исходными сбоями считаются внутренние ошибки компонентов) таблица видов и последствий отказов соодержит следующие данные:

Item: компонент, в котором произошел исходный сбой;

Initial failure mode: исходный сбой (событие) в данном компоненте;

End effect: набор финальных последствий данного исходного сбоя;

Sev: тяжесть данного последствия для системы (по 10-балльной шкале);

P: вероятность достижения данного последствия в результате возникновения данного исходного сбоя.

Во втором случае (когда исходными сбоями считаются нерабочие состояния компонентов) таблица видов и последствий отказов соодержит следующие данные:

Item: компонент, в котором произошел исходный сбой;

nitial failure mode: исходный сбой (нерабочее состояние) в данном компоненте;

Potential cause: возможные причины данного сбоя:

Cause: описание непосредственной причины сбоя;

P: абсолютная вероятность возникновения данного сбоя;

Occ: рейтинг вероятности возникновения данного сбоя (по 10-балльной шкале);

End effect: набор финальных последствий данного исходного сбоя;

Sev: тяжесть данного последствия для системы (по 10-балльной шкале);

P (cond): условная вероятность достижения данного последствия в результате возникновения данного исходного сбоя.

Пример анализа рисков сложной технической системы

В качестве примера рассмотрим систему торможения самолета (см. авиационный стандарт ARP-4761).

Эта система описана в стандарте ARP-4761 следующим образом:

Торможение колес основного шасси используется для безопасного замедления самолета во время фаз руления и посадки, а также в случае прерванного взлета. Колесные тормоза также предохраняют от непреднамеренного движения самолета при его парковке, и могут использоваться для обеспечения дифференциального торможения для управления направлением движения самолета. Второй функцией системы торможения колеса является остановка вращения основного шасси после его уборки.

Торможение на земле управляется либо вручную, при помощи тормозных педалей, либо автоматически (автоматическое торможение под управлением компьютерного модуля "блок управления тормозной системы" – BSCU) без необходимости использования тормозных педалей.

С учетом того требования, что вероятность потери всего торможения колес должна быть менее 5E-7 на полет, то было принято конструкторское решение, чтобы каждое колесо имело узел тормоза, управляемый двумя независимыми комплектами гидравлических поршней. Один комплект работает от ЗЕЛЕНОЙ гидравлической системы и используется в НОРМАЛЬНОМ режиме торможения. Другой режим является резервным и выбирается автоматически при отказе НОРМАЛЬНОЙ системы. Он работает независимо с использованием ГОЛУБОЙ гидравлической системы и поддерживается аккумулятором, который также используется для привода парковочного тормоза. Аккумулятор питает АЛЬТЕРНАТИВНУЮ систему в АВАРИЙНОМ режиме торможения, когда происходит отказ ГОЛУБОЙ гидравлической системы, и НОРМАЛЬНЫЙ режим не доступен. Снижение давление в ЗЕЛЕНОЙ гидравлической системе ниже порогового значения, либо от отказа самой ЗЕЛЕНОЙ системы подачи, либо от ее отключения при помощи BSCU вследствие наличия неисправностей, приводит к тому, что автоматический селектор подключает ГОЛУБУЮ систему подачи к АЛЬТЕРНАТИВНОЙ системе торможения. Как в НОРМАЛЬНОМ, так и в АЛЬТЕРНАТИВНОМ режимах имеется устройство против скольжения, которое работает при всех скоростях, превышающих 2 м/с.

В НОРМАЛЬНОМ режиме электрический сигнал о положении тормозной педали

передается в блок BSCU. Он в свою очередь выдает соответствующие сигналы управления на тормоза;

для обеспечения требований, предъявляемых к BSCU по работоспособности и целостности, используется резервирование:

– BSCU состоит из двух независимых подсистем, отвечающих требованиям работоспособности;

– каждая подсистема BSCU содержит как канал передачи команд торможения, так и канал контроля для выполнения требований по целостности.

Каждая подсистема BSCU генерирует необходимые напряжения в своем собственном электропитании. Предусмотрен контроль электропитания для выявления условий, выходящих за пределы напряжения, заданного спецификацией. Предусмотрены входные сигналы от тормозных педалей к командным и контрольным каналам, которые используются для расчета необходимых команд на торможение. Команды, генерируемые каждым каналом, сравниваются, и если они не согласуются, то выдается сообщение о неисправности. Результаты контроля электропитания и компаратора также представляются на монитор достоверности системы. Неисправность, о которой сообщается каждой подсистемой BSCU, приведет к тому, что система отключит свои выходные сигналы и переведет монитор достоверности подсистемы в состояние недействительных значений. Предусмотрен монитор достоверности каждой подсистемы BSCU для общего монитора достоверности BSCU. Отказ как подсистемы 1, так и подсистемы 2 приведет к тому, что селекторный клапан будет переведен на альтернативную систему торможения.

При нормальной работе подсистема BSCU 1 обеспечивает команды торможения и противоскольжения на колесные тормоза. Когда подсистема 1 сообщает об отказе через монитор достоверности подсистемы, то включается выход подсистемы 2, если она работоспособна, на обеспечение этих команд. В том случае, когда последовательно выходит из строя подсистема 2, все выходы BSCU выключаются, и монитор достоверности BSCU устанавливается в состояние недействительных значений.

AADL-описание системы BSCU выглядит следующим образом:

```
SYSTEM bscu
        FEATURES
         pow_1 : IN DATA PORT;
         pow_2 : IN DATA PORT;
         ped_1 : IN DATA PORT;
         ped_2 : IN DATA PORT;

         cmd : OUT DATA PORT;
         as_1 : OUT DATA PORT;
         as_2 : OUT DATA PORT;
         valid_out : OUT DATA PORT;
  END bscu;

    SYSTEM IMPLEMENTATION bscu.impl
        SUBCOMPONENTS
         sys_1 : SYSTEM bscu_subsys.impl;
         sys_2 : SYSTEM bscu_subsys.impl;
         sw : PROCESS bscu_subsys_switch;
         valid : PROCESS bscu_validity_mon;
        CONNECTIONS
         c_pow_1 : PORT pow_1 -> sys_1.pow_in;
         c_pow_2 : PORT pow_2 -> sys_2.pow_in;
         c_ped_1 : PORT ped_1 -> sys_1.ped_in;
         c_ped_2 : PORT ped_2 -> sys_2.ped_in;
```

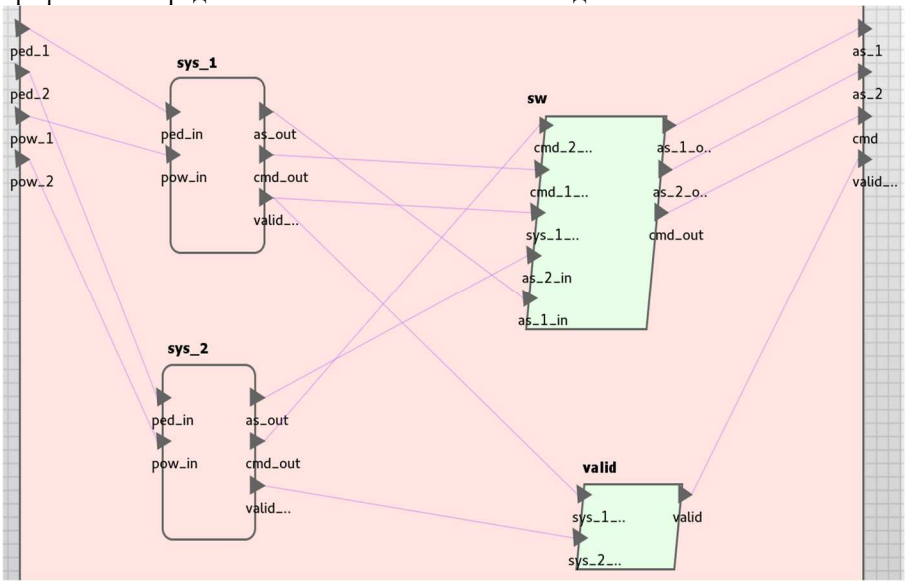
c_1_sw : **PORT** sys_1.valid_out -> sw.sys_1_valid;

Модель ошибок для системы BSCU описывается на Error Model Annex следующим образом:

```
  use types   ErrLib;
  use behavior ErrLib::DataStates;
error propagations
  cmd: out propagation { NoService };
  as_1: out propagation { NoService };
  as_2: out propagation { NoService };
  end propagations;
component error behavior
propagations
  Failed -[]-> cmd( NoService );
  Failed -[]-> as_1( NoService );
  Failed -[]-> as_2( NoService );
end component;
  composite error behavior
  states
      [ sys_1.FalseData
      or (  ( sys_1.Failed or sw.StuckTo2 )
        and sys_2.FalseData
        )
      ]-> FalseData;

      [ sys_1.Failed and sys_2.Failed
      or sys_1.Failed and sw.StuckTo1
      or sys_2.Failed and sw.StuckTo2
      or sw.StuckIntermediate
      ]-> Failed;
  end composite;
```

Графическое представление системы BSCU выглядит так:



Результаты анализа дерева неисправностей.

Вероятность отказа системы BSCU через 5 часов полета = 0.000155

Значимости первичных событий для отказа системы BSCU приведены в таблице:

Компонент	**Отказ**	**Вероятность**	RAW	RRW	Birnbaum	Fussel-Vesely	Diagnostic
Командный канал	Отказ	0.00015	1.96745	1.00015	0.00015	0.00015	0.000295
Контрольный канал	Отказ	0.00015	1.96745	1.00015	0.00015	0.00015	0.00029
Питание	Отказ	0.00015	6451.15	30.863	0.999995	0.9676	0.9676
Переключатель подсистем BSCU	Переключатель залип в промежуточном положении	0.000005	6451.15	1.0333	0.99985	0.0323	0.0323

Результаты марковского анализа.

Размер марковской цепи для одной подсистемы BSCU равен 130 элементов.

Вероятность отказа одной подсистемы BSCU через 5 часов полета равна 0,00015.

Результаты анализа видов последствий и критичности отказов.

Виды последствий и критичности отказов в системе BSCU, когда исходными сбоями считаются внутренние ошибки компонентов, приведены в следующе таблице:

Item(s)	**Initial failure mode(s)**	**End effect**	**Sev**
Pow	BadElecticity (P=0.00006)	pow.FalseData	7
Pow	Failure (P=0.00015)	sys_1.Failed	8
pow_mon	AlwaysTrue (P=1E-6)	pow_mon.FalseData	7
Cmd	BadData (P=0.00006)	cmd.FalseData	7
Cmd	Failure (P=0.00015)	cmd.Failed	7
Mon	BadData (P=0.00006)	mon.FalseData	7
Mon	Failure (P=0.00015)	mon.Failed	7
Cmp	AlwaysTrue (P=1E-6)	cmp.FalseData	7
Pow	BadElecticity (P=0.00006)	pow.FalseData	7
Pow	Failure (P=0.00015)	sys_2.Failed	8
pow_mon	AlwaysTrue (P=1E-6)	pow_mon.FalseData	7

Cmd	BadData (P=0.00006)	cmd.FalseData	7
Cmd	Failure (P=0.00015)	cmd.Failed	7
Mon	BadData (P=0.00006)	mon.FalseData	7
Mon	Failure (P=0.00015)	mon.Failed	7
Cmp	AlwaysTrue (P=1E-6)	cmp.FalseData	7
Sw	StuckTo1Failure (P=0.000005)	sw.StuckTo1	8
Sw	StuckTo2Failure (P=0.000005)	sw.StuckTo2	8
Sw	StuckIntermediateFailure (P=0.000005)	func.BscuFailed	10

5.4. Классификация моделей надежности ПО

Большинство моделей надежности исходят из предположения, что найденные ошибки и дефекты устраняются немедленно или определяются временем их устранения и новые дефекты не вносятся. В результате количество дефектов в системе уменьшается, а надежность возрастает, такие модели получили название моделей роста надежности. Shick G. предложил следующую классификацию моделей надежности.

Прогнозирующие модели надежности основаны на измерении технических характеристик создаваемой программы: длина, сложность, число циклов и степень их вложенности, количество ошибок на страницу операторов программы и др.

Модель Мотли–Брукса основывается на длине и сложности структуры программы (количество ветвей и циклов, вложенность циклов), количестве и типах переменных, а также интерфейсов.

Модель Холстеда дает прогнозирование количества ошибок в программе в зависимости от ее объема и таких данных, как число операций (n1) и операндов (n2), а также общее число обращений к ним (N1 + N2).

Измерительные модели предназначены для измерения надежности ПО, работающего с заданной внешней средой и следующими ограничениями:

– ПО не модифицируется во время периода измерений свойств надежности;

– обнаруженные ошибки не исправляются;

– измерение надежности проводится для зафиксированной конфигурации ПО.

Примером таких моделей является модель Нельсона, Рамамурти–Бастани и др.

Модель Нельсона основывается на выполнении k прогонов программы при тестировании и позволяет определить надежность по формуле:

$$R(k) = \exp[-\sum tj\ \lambda(tj)],$$

где tj – время выполнения j-го прогона, $\lambda(tj) = -\ln(1-qj)/tj$ и при $qi \le 1$ интерпретируется как функция интенсивности отказов.

Оценочные модели основываются на серии тестовых прогонов и проводятся на этапах тестирования системы. В тестовой среде определяется вероятность отказа программы при ее тестировании или выполнении. Эти типы моделей могут применяться на этапах ЖЦ и могут быть следующих видов:

- Модели без подсчета ошибок основаны на измерении интервала времени между отказами и позволяют спрогнозировать количество ошибок, оставшихся в программе. К этим моделям относятся модели Джелински и Моранды, Шика Вулвертона, и Литвуда–Вералла.

- Модели с подсчетом отказов базируются на количестве ошибок, обнаруженных на заданных интервалах времени. К этому классу моделей относятся модели Шика–Вулвертона, Шумана, Пуассоновская модель и др.
- Модели с подсевом ошибок основаны на количестве устраненных ошибок и подсеве, внесенном рограмму искусственных ошибок, тип и количество которых заранее известны. При внесении изменений в программу проводится повторное тестирование и оценка надежности. Этот подход базируется на тестировании и редко используется из-за дополнительного объема работ для покрытия тестами компонентов системы.
- Модели с выбором области входных значений основываются на генерации множества тестовых выборок из входного набора. К этому типу моделей относится модель Нельсона и др. На процессах выявления отказов и их интенсивности используются также:

1) модели, рассматривающие интенсивность отказов, как Марковский и пуассоновский процесс;

2) модели роста надежности.

Четкой границы между этими моделями провести нельзя, однако по фактору интенсивности отказов и моделей поведения их можно разделить на экспоненциальные, логарифмические, геометрические, байесовские и др.

Для практической оценки надежности более всего представляет интерес оценочная модель Мусы, Мусы-Окомото и др. Рассмотрим их.

1). Оценочная модель Мусы основана на следующих положениях:

– тесты адекватно представляют среду функционирования;

– происходящие отказы учитываются (оценивается их количество);

– интервалы между отказами независимы;

– время между отказами распределено по экспоненциальному закону;

– интенсивность отказов пропорциональна числу ошибок;

– корость исправления ошибок (относительно времени функционирования) пропорциональна интенсивности их появления.

Эта модель учитывает интервалы между отказами, которые распределяется по экспоненциальному закону, а интенсивность отказов пропорциональна числу обнаруженных ошибок. Исходя из этой модели, можно установить зависимость:

1) среднего числа отказов от времени функционирования τ (рис.1), которое задается в виде:

$$m = M0\,[1 - \exp(-\frac{c\tau}{M_0 T_0})],$$

где M0 – общее число ошибок; T0 – начальная наработка на отказ; c – коэффициент времени испытаний; τ – время функционирования.

2) средней наработки на отказ T от времени функционирования τ (рис.2):

$$T = T0 \exp(\frac{c\tau}{M_0 T_0}),$$ где M0, T0, c – зависят от наработки на отказ.

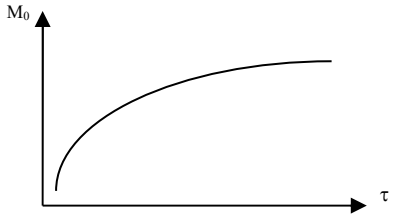


Рис. 1. Зависимость числа отказов от времени τ

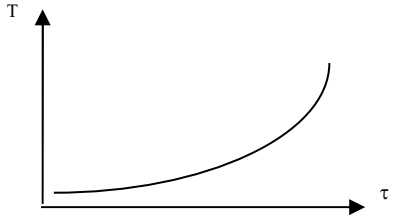


Рис. 2. Зависимость средней наработки на отказ от времени τ

График этой зависимости представлен областью 1 (рис. 3), для которой $M_1 = 1, 2, ...$ – номера наблюдений, а $\tau_1, \tau_2 \ldots \tau_{M1}$ – время между отказами. Область 2 (рис.3) соответствует достижению средней наработки T_p на отказ за время $\Delta\tau$.

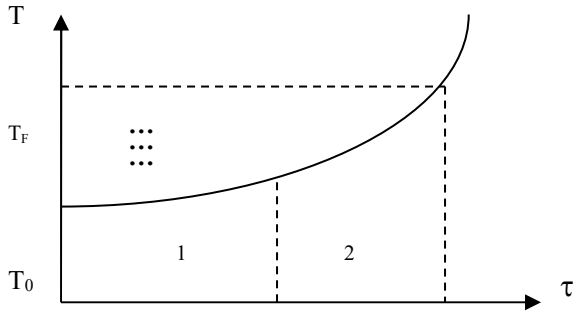


Рис. 3. Области отказов и наработки на отказ от времени τ

По собранным данным об ошибках оцениваются параметры T0 и M0, с помощью которых определяются дополнительное число ошибок по формуле:

$$\Delta m = M_0 T_0 [\frac{1}{T} - \frac{1}{T_0}].$$

2). Модель Мусы–Окумото (логарифмическая) допускает, что некоторые ошибки имеют большую вероятность проявления в виде отказов, снижают интенсивность отказов с каждой устраненной ошибкой и дают экспоненциальное распределение. Мат.ожидание найденных ошибок $m(t)$ имеет вид:

$$m(t) = (1/\theta) \ln (\lambda_0 \theta t + 1),$$

где λ_0 – исходная интенсивность отказов, θ – скорость спада интенсивности отказов с каждым устраненным дефектом, а функция интенсивности отказов $\lambda(t)$ имеет вид:

$\lambda(t) = \lambda_0 / (\lambda_0 \theta t + 1)$.

3). Модель Гоело–Окумото (экспоненциального роста) описывает обнаружение ошибок с помощью неоднородного пуассоновского процесса. В ней интенсивность отказов зависит от времени, а количество выявленных ошибок при тестировании трактуется как случайная величина.

Исходные данные m, X_i и T аналогичны данным предыдущих моделей. Функция среднего числа отказов, обнаруженных к моменту t, имеет вид:

$$m(t) = N(1 - e^{-bt}),$$

где b – интенсивность обнаружения отказов; $q(t)=b$ - показатель роста надежности.

Функция интенсивности $\lambda(t)$ зависит от времени работы системы до отказа:

$$\lambda(t) = Nbe^{-bt},\ t \geq 0,$$

где N и b решаются с помощью уравнения:

$$-m/N - 1 + \exp(-bT) = 0$$

4). S-образная модель. Функция интенсивности $\lambda(t)$ выявления ошибок в зависимости от времени работы имеет вид:

$$\lambda(t) = a\beta^2 t \exp(-\beta t),$$

где a – общее количество дефектов, обнаруженных от начала и до конца тестирования;

β – скорость изменения функции интенсивности выявления отказов.

Введение в формулу параметра в степени 1 модели Мусы и Гоэла–Окомото дает изменение формы кривой так, что она сначала растет, а потом спадает. Практика применения этих моделей в автоматизированных системах привела к уточнению функции интенсивности при введении дополнительного параметра n:

$$\lambda(t) = a\beta^{n+1} t^n \exp(-\beta t),$$

где n отражает сложность и размер проекта некоторой системы. Это позволяет более точно определить форму кривой интенсивности с учетом получаемых практических результатов.

Процесс оценки надежности включает:

– протоколирование отказов в ходе функционирования системы, измерение надежности по отказам и использование результатов измерений для определения потерь надежности в период времени эксплуатации;

– анализ частоты и серьезности отказов при определении порядка устранения соответствующих ошибок;

– оценка влияния времени функционирования системы на надежность с помощью инструментов измерения надежности.

5.5. Оценка надежности систем реального времени

Некоторые типы систем реального времени с обеспечением безопасности требуют высокой надежности (недопустимость ошибок, точность, достоверность и др.), которая в значительной степени зависит от количества оставшихся и не устраненных ошибок в процессе ее разработки на этапах ЖЦ.

В ходе эксплуатации системы также могут обнаруживаться и устраняться ошибки. Если при их исправлении не вносятся новые ошибки или их меньше, чем устраняется, то в ходе эксплуатации надежность системы непрерывно растет. Чем интенсивнее проводится эксплуатация, тем интенсивнее выявляются ошибки и быстрее растет надежность.

На надежность ПО влияют, с одной стороны, угрозы, приводящие к неблагоприятным последствиям, риск нарушения безопасности системы, с другой стороны, способность совокупности компонентов системы сохранять устойчивость в процессе ее эксплуатации.

Риск уменьшает свойства надежности, особенно если обнаруженные ошибки могут быть результатом проявления угрозы извне.

Методы и модели надежности постоянно развиваются и уточняются, поскольку надежность является одной из ключевых проблем измерения качества современных распределенных по Интернет систем.

Появилось новое направление – *инженерия надежности ПО* (Software reliability engineering – SRE), которое ориентировано на количественное изучение операционного поведения компонентов системы по отношению к пользователю, ожидающему получить надежную работу системы. Надежность обеспечивается путем:

1) измерения надежности, т.е. проведения количественной ее оценки методами предсказаний, собранными данными о поведении системы в процессе тестирования и эксплуатации системы;

2) оценки стратегии и применения метрик для готовых компонентов, созданных в процессе разработки компонентов системы в заданной среде и стандартов на измерение надежности системы;

3) применения современных методов инспектирования, верификации, валидации и тестирования в процессе разработки отдельных компонентов и системы в целом.

Обеспечение надежности на этапах ЖЦ

Для получения высокой надежности системы требуется наблюдать за достижением показателей надежности и качества на всех этапах ЖЦ согласно рекомендациям стандарта ISO/IEC 12207: ЖЦ [Андон Ф.И., Коваль Г.И. и др. Основы инженерии качества программных систем. К.: Наук. думка, 2007, 670 с.]. К основным процессам стандарта ЖЦ относятся:

– спецификация требований,
– проектирование,
– реализация,
– тестирование,
– испытание,
– сопровождение.

На этапе спецификации требований определяются задачи и внешние спецификации основных (целевых) требований к системе с заданием метрик для оценки надежности, в терминах интенсивности отказов или вероятности безотказного его функционирования. Разработчики системы формируют:

– приоритеты функций системы по критерию важности их реализации;
– параметры среды и интенсивности использования функций и их отказов;
– входные и выходные данные для каждой функции системы;
– категорий отказов и их интенсивности при выполнении функций в единицу каледарного времени.

На этапе проектирования определяются:

– размеры информационной и алгоритмической сложности всех типов проектируемых компонентов;
– категории дефектов, свойственные всем типам компонентов системы;
– стратегии функционального тестирования компонентов по принципу «черного ящика» с помощью тестов для выявления ошибок в данных.

Для обеспечения надежности продукта проводится:

– анализ вариантов архитектуры системы на соответствие требованиям к надежности;
– анализ рисков, режимов отказов, деревьев ошибок для критических компонентов с целью - обеспечения отказоустойчивости и восстанавливаемости системы;
– прогнозирование показателей размера системы, чувствительности к ошибкам, степени тестируемости, оценки риска и сложности системы.

На этапе реализации *и тестирования системы* проектные спецификации переводятся в коды и подготавливаются наборы тестов для автономного и комплексного их тестирования. При проведении автономного тестирования обеспечение надежности состоит в предупреждении появления дефектов в компонентах и создание эффективных методов защиты от них. Все последующие этапы разработки не могут обеспечить надежность систем, а лишь способствуют повышению уровня надежности за счет обнаружения ошибок с помощью тестов различных категорий.

На этапе испытаний проводится системное тестирование для соответствия внешних спецификаций функций целям проекта. Испытание проводится или в реальной среде функционирования, или на испытательном стенде, который имитирует работу аппаратуры. При подготовке к испытаниям изучается "история" тестирования на процессах ЖЦ в целях использования ранее разработанных тестов, а также составления специальных тестов испытаний. На этом этапе осуществляется:

– управление ростом надежности путем неоднократного исправления и регрессионного тестирования ПО;

– принятие решения о степени готовности ПО и возможности его передачи в эксплуатацию;

– оценка надежности по результатам системного тестирования и испытаний по соответствующим *моделям надежности*, подходящих для заданных целей.

На этапе сопровождения оценка надежности ПО проводится:

– протоколирование отказов в ходе работы системы, измерения надежности функционирования и использования результатов измерений для определения потерь надежности в период времени эксплуатации;

– анализ частоты и серьезности отказов для определения порядка их устранения;

– оценка влияния функционирования системы на надежность в условиях усовершенствования технологии и применения новых инструментов разработки.

5.6. Инженерия надёжности ПО

Термин «инженерия надёжности ПО» SRE (Software reliability engineering) понимается как количественное изучение операционного поведения систем, содержащих ПО по отношению к пользователю, интересующемуся его надёжностью. SRE включает:

1. Измерение надёжности, т.е. оценка и предсказание её с помощью моделей надёжности.

2. Атрибуты и метрики конструирования программного продукта (ПП), процесс его разработки, архитектуру системы, среду функционирования ПП и их влияние на надёжность.

Применение полученных знаний при выборе архитектуры ПО, в процессе его разработки, тестирования, использования и эксплуатации.

В проблеме надежности ПО (НПО) рассматривается такое понятие как способность ПО обладать свойствами, желательными для пользователя (английский термин dependability). ПО означает, что компьютерная система выполняет свои функции (так как их понимает пользователь). Пользователь – это другая система (человек или физический объект), который взаимодействует с ПО компьютерной системой.

Dependability (D) может характеризоваться различными атрибутами (свойствами):

– готовность к использованию (availability),

– готовность к непрерывному функционированию (reliability),

– безопасность (safety) для окружающей среды (для внешнего окружения).

– способность не вызывать катастрофических последствий в случае отказа.

– конфиденциальность (confidentability) - секретность, сохрананность секретности информации,

– способность к сохранению информации (integrity), устойчивость к её самопроизвольному изменению,

– эксплуатационная способность ПО (aintainability), простоту выполнения операций обслуживания, например, устранение ошибок, восстановление после ошибки и т.п.

– готовность и сохранность защиты (security) по отношению к административному лицу + скрытность confidentability,

– отказ (failure), отклонение поведения системы от предписанного, т.е. когда система перестаёт выполнять предписанные ей функции,

– ошибка (error), состояние системы, которое вызывает отказ (в случае человека используется термин mistake),

– отказ (fault) обозначает причина ошибочной ситуации, что вызывает её.

– Defect означает различие между fault и failure и фактически означает либо fault (причина), либо failure (действие).

Обеспечение требуемой надежности достигается:

– предотвращением отказа (fault preventation),

– устранением отказа (remival fault),

– возможность выполнения ПО (fault tolerance) при наличии ошибок,

– оценкой возможности появления отказа (fault forecasting) и его последствия.

Составляющие надежности – это всевозможные повреждения ПО и средства обеспечения надежной работы системы. Повреждения и причины, вызывающие потерю надежности (fault, error, failure, undependability), т.е. причины, вызывающие потерю пригодноспособности системы приведены на рис.3.

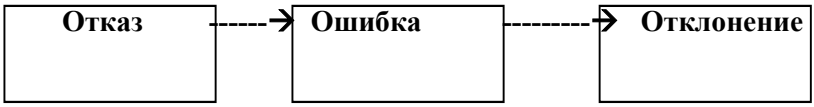


Рис.3 Повреждение надежности

Основу надежности ПО (рис.4) составляют:

– средства (means) достижения, предотвращение причин отказа, устранение причин отказа, отказоустойчивость – нечуствительность к возникновению причин отказа, предвидение причин возникновения отказа,

– атрибуты (attributes) – составляющие пригодноспособности: готовность, безопасность, секретность, сохраняемость.

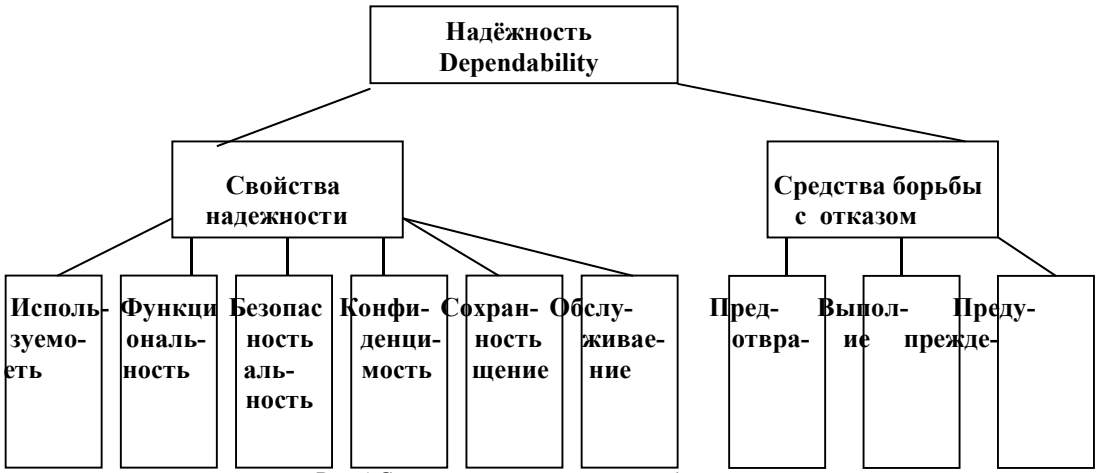


Рис.4 Составляющие элементы надежности

Существует большое разнообразие видов отказов, типичные из них: внезапные, постепенные, перемещающиеся (сбои). Причины отказов могут быть физические, конструктивные, отказы взаимодействия и др. Они могут возникать по разным причинам: естественным путём, вносятся человеком или внешней средой в период создания системы и её эксплуатации, быть постоянными или носить временный характер.

Проблема надёжности оборудования существенно отличается от проблемы надёжности ПО.

Основные отличия состоят в следующем.

1. Размерность. Программные системы создают огромное число кодов, которое обычно намного превышает число физических элементов системы. Кроме того, сложность взаимодействия компонент ПО намного превышает сложность взаимодействия технических элементов. Таким образом, ПО как объект исследования является более сложным.
2. Элементы ПО не стареют, не деградируют во времени.
3. Методы введения избыточности в ПО существенно отличаются от методов используемые для оборудования. Это не касается и методов обеспечения эксплуатационной надёжности, таких как профилактика и замена. Так как в принципе весьма трудно получить программный продукт, не содержащий причин возникновения отказа, дефектов и т. п.
4. Хотя, с одной стороны, компоненты ПО не стареют, однако, в процессе эксплуатации могут вноситься искажения в компоненты ПО, что может приводить к возрастанию интенсивности отказов.
5. В отличие от отказов аппаратуры отказы ПО являются не физическими, т.е. вызываемыми необратимыми изменениями в элементах, а конструктивных зон отказа труднее поддаются визуализации, классификации, обнаружению и коррекции. Как результат, надёжность ПО трудней измерить и анализировать, чем надёжность аппаратуры.
6. Структура и состав ПО динамически изменяются, что бывает трудно учесть в соответствующих моделях надёжности ПО.
7. Надёжность ПО в большей степени зависит от среды (например, от вируса). Вирус может изменить ПО, в то время как изменить структуру аппаратной части случайным образом нельзя. Т.е. характер взаимодействия с внешней средой существенно различен.

Однако между проблемами надёжности аппаратуры и ПО имеется и сходство. В основе этих проблем лежат случайные явления и способы их анализа основываются на соответствующих методах теории вероятностей и случайных процессов.

В компьютере компонента ПО, команда и данные (кодовые слова) обрабатываются в дискретные моменты времени б, 2б, Обработка такого элемента может быть произведена «удачно» или нет.

Пусть отказ означает появление первого неудачно обработанного компонента и Т есть время до появления отказа. Пусть qb – вероятность неудачи.

Тогда $P\{ T > n\delta \} = (1 - q\delta)^n$ и среднее время ожидания $T = \delta / q\delta$. Пусть δ убывает так, что Т остаётся фиксированным. Тогда при малом δ и фиксированном Т имеем $P\{ T > t \} = (1 - \delta / T)^{t/\delta} \approx e^{-t/T}$

Т.е. время Т до отказа ПО в данном случае является непрерывной с…l распределённой экспоненциально с параметром Т…1/Т аналогично тому, как это имеет место и в случае надёжности аппаратуры.

Так как надёжность ПО бывает трудно оценить, то важное место в проблеме надёжности ПО принадлежит методам создания устойчивости к отказам ПО. В методологии создания такого ПО важное место принадлежит концепции “coverage” – т.е вероятности того, что система восстановится самопроизвольно после возникновения в ней соответствующей причины отказа (fault).

5.7. Проведение по моделям оценки надежности ПО

Практика применения моделей показывает, что среди названных моделей наиболее перспективными являются модели оценочного типа, которые базируются на пуассоновских процессах (модели Мусы, Гоэла–Окомото, S-образные и др.). По этим моделям надежность стремиться к 1. Одним из недостатков является форма кривой интенсивности выявленных отказов (экспоненциальная), которая строго спускается при t>0 вблизи t=0. Это говорит о том, что при тестировании проведено недостаточно экспериментов или мало найдено ошибок, когда интенсивность отказов была близка 0. В системе остаются ошибки и их поиск требует больше времени.

В таблице 1 представлены практические значения функций интенсивности отказов $\lambda(t)$ и количество отказов $\mu(t)$ для базовых и общих моделей. В них значения a и β находятся в следующих соотношениях:

$N= a,\ \beta = a,\ b = \beta,\ \beta^1 = \beta,\ a_0 = a\beta.$

Таблица 1. Характеристика моделей надежности Пуассоновского типа

Название модели	Функции интенсивности отказов $\lambda(t)$	Функции кумулятивного количества отказов $\mu(t)$
Модель Гоэла-Окумото	$\lambda(t) = Nb \exp(-bt)$	$\mu(t)= N(1- \exp(-bt))$
Модель Мусы	$\lambda(t) = \beta_0 \beta_1 \exp(-\beta_1 t)$	$\mu(t)= \beta_0 (1- \exp(-\beta_1 t))$
S –подобная модель	$\lambda(t) = a\beta^2 t \exp(-\beta t)$	$\lambda(t) = a\{1-(1+\beta t) \exp(-\beta t)$
Модель Шнайдевинда	$\lambda(t) = a_0 \exp(-\beta t)$	$\mu(t)= a_0/\beta (1- \exp(-\beta t))$
Общая модель пуассоновского процесса	$\lambda(t) = a\beta^{n+1} t^n \exp(-\beta t)$	$\mu(t)= a (n^1 - \sum n\beta^{n-1}/(n-1)^1 t^n exp(-\beta t)))$

Для метода максимального правдоподобия задаются данные $a, \beta, n,$ решим систему уравнений:

$$\begin{cases} \alpha = \dfrac{m}{1-\sum_{i=0}^{n} \dfrac{n!\beta^{n-i} t_m^{n-i}}{(n-i)!} \exp(-\beta t_m)} \\ \dfrac{n+1}{\beta} m = \sum_{k=1}^{m} t_k + \dfrac{m\beta^n t_m^{n+1} \exp(-\beta t_m)}{1-\sum_{i=0}^{n} \dfrac{n!\beta^{n-i} t_m^{n-i}}{(n-i)!} \exp(-\beta t_m)} \end{cases}$$

где параметр n зависит от процесса тестирования и его рекомендуемых значений:

n=0 – для небольшого проекта, в котором разработчик является также тестером (модель Мусы, Гоэло-Окомото и др.);

n=1 – для среднего проекта, в котором тестирование и проектирование ПО исполняются несколькими разработчиками из одной рабочей группы (S- образная модель);

n=2 – для большого проекта, в котором группы тестирования и проектирования работают параллельно;

n=3 – для очень большого проекта, в котором группы тестирования и разработки работают независимо друг от друга.

На основе экспериментальных данных получены функции о количестве отказов $\mu(t)$ и интенсивности отказов $\lambda(t)$ на выходных данных и значениях параметра n (рис. 4). Этот рисунок показывает вид функций $\mu(t)$ при разных значениях n=0,1, 2, 3.

Наибольшее приближение достигается при n=3, а наименьшее при n=0 (модель Мусы, Гоэло-Окомото и др.). Это подтверждается соответствующими статистическими данными (табл.2), которые задают разницу между выходными данными (t_2) и соответствующими значениями функции $\mu(t)$ при значениях n = 0, 1, 2, 3.

На основе экспериментальных данных a, β, n, (табл. 2) приведены значения функций $\mu(t)$ и $\lambda(t)$ при n = 3, 2, 1, полученные при использовании методов оценки надежности Мусы, Мусы-Окомото и Шнайдевинда. Функции $\mu(t)$ для этих методов приведены на графике (рис. 5). Им соответствуют кривые экспоненциального типа. Графики этих функций близки друг другу из-за близких значений, полученных по заданным моделям.

Табл. 2. Статистические данные функции μ(t) при n=3, 2, 1 и данных t_2

Статистические показатели	**Разница функций *t_2 - μ_3***	**Разница функций *t_2 - μ_2***	**Разница функций *t_2 - μ_1***	**Разница функций *t_2 - μ***
Среднее отклонение	16.13522	16.22889	19.88387	58.93807
Медианное отклонение	15.27700	14.11600	16.0000	60.89700
Максимум отклонение	33.58100	54.23600	49.10800	88.80200
Минимум отклонение	4. 848000	-1.280000	4.175000	15.96200
Среднеквадратиче-ское отклонение	8.374089	17.37143	14.07056	23.63765

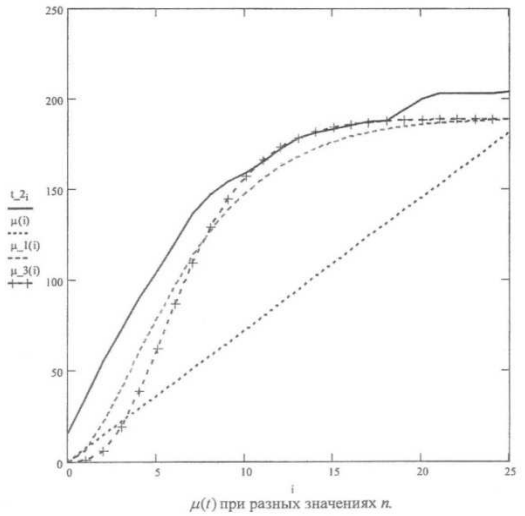


Рис. 5. Графики функций μ(t) по моделям Мусы, Окомото и Шнайдевинда

Для более эффективного применения приведеннях моделей надежности требуется значительное количество статистических данных о количестве и распределении отказов. А это требует увеличения количества экспериментов на процесах тестирования, системного тестирования для покрытия тестами всех компонентов и маршрутов прохождения путей в ПО.

5.8. Технологические модули (ТМ) оценки надежности систем

ТМ разработан в рамках работ по проекту ПРОТВА ВПК (1986-1989). В состав ТМ надежности входит четыре технологических модуля (ТМ) [15, 16, 22 с.283-296]: распределение надежности, прогнозирование плотности дефектов, прогнозирование надежности и оценки надежности.

ТМ «Распределения надежности» реализует метод распределения надежности по компонентам системы путем парного их сравнения и построения квадратной матрицы A размером $n \times n$ из элементов вида:

$$a_{11} = a_{22} = ... = a_{nn} = 1,\ a_{ij} = \frac{1}{a_{ji}},\ u, j = 1,...,n;\ i \neq j;\ n=k, l, m,$$

где n – количество сравниваемых компонентов, k, l, m – количество функций и модулей соответственно. Матрица включает относительный вес i-го компонента и вычисляется по формулам:

$$w_i = \frac{\sum_{j=1}^{n} a_{ij}}{\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}},\ \sum_{i=1}^{n} w_i = 1$$

В случае больших размеров матрицы в целях получения более точных относительных оценок компонентов иерархии, вычисляются, так называемые, собственный вектор и собственные значения матрицы согласно известным уравнениям [24]. В них используются следующие данные: λ_{max} – максимальное собственное значение матрицы A n–порядка, w_i – коэффициент относительного веса элементов матрицы A, W = ($w_1, w_2, ... w_n$) – собственный вектор, которому соответствует λ_{max}. Общность решения задачи сравнения устанавливается соотношением $\alpha = \sum_{i=1}^{n} w_i$ и значением $\sum_{i=1}^{n} w_i = 1$. Если матрица A имеет $n-1$ собственных значений λ, равных нулю и $\lambda_{max} = n$, то она является согласованной.

Определение индекса согласованности CI и коэффициента согласованности CR проводится по формулам:

$$CI = \frac{\lambda_{\max} - n}{n - 1},\ CR = \frac{CI}{E(CI)}, \text{ где}$$

$E(CI)$ – математическое ожидание для матрицы парных сравнений A $(n \times n)$.

Критерий приемлемости парного сравнения элементов в матрицах размером $n \geq 3$ получен такой: $CR \leq 0.05$ и $CR < 0.1$ для $n>5$. По результатам сравнения формируется квадратная матрица $F(k \times k)$.

Аналогично проводится сравнение приложений ПС. В результате сравнения получают k матриц. Возможный порядок каждой матрицы – l, а максимальный порядок каждой из них – m.

Инструмент для поддержки метода сравнения – ExpertChoice для входной матрицы A автоматически получает собственный вектор W, собственное значение λ_{max} и коэффициент согласованности CR. Для вычисления λ_{max} и W используются соответствующие функции пакета Matlab.

Результаты сравнений заносятся в форму, содержащую перечень весовых коэффициентов программ, критерии, индексы и коэффициенты согласованности. Они предоставляются в виде готовых результатов обработки матриц. Полученные весовые коэффициенты синтезируются с помощью пакета MATLAB 6.5. Результаты отображаются в виде отчета о распределении надежности по объектам системы.

ТМ «Прогнозирование плотности дефектов» реализует набор моделей надежности для заданного класса программ системы обработки данных.

Прогнозирование плотности дефектов проводится по модели RLM (Rome Laboratory Model) и состоит в оценке влияния на плотность дефектов согласно следующих действий:

1) анализ значений параметров модели прогнозирования надежности, включая остаток дефектов от предыдущего этапа работ с ПО, используется для целевого распределенного значения надежности ПО;
2) сравнение прогнозируемого значения надежности с целевым распределенным значением;
3) корректировки переменных параметров для учета текущего состояния проекта ПО;
4) оценка параметров модели прогнозирования надежности;
5) прогнозирование плотности дефектов;
6) определение пороговых значений (допусков) для оценок результатов прогнозирования и анализа альтернатив;
7) расчет прогнозного значения надежности для ПО.

Полученная оценка является модификатором базового значения плотности дефектов для определенного класса ПО.

Расчет плотности дефектов делается по модели RLM (Rome Laboratory Model). Сначала выполняется однократное прогнозирование плотности дефектов по формуле:

$$D_g = \prod_{i=1}^{9} K_i,$$

где K_i – модификаторы плотности дефектов D_0, с учетом пороговых значений данных о плотности дефектов.

Затем для каждого ПО результаты сравниваются с полученными по модели RLM. Проверка показала, что для ПО объемом 10 - 25 KSLOC погрешность прогнозировании плотности дефектов – примерно составляет 30-35%. Это объясняется некоторыми ограничениями системы Hugin Lite 6.5. Полученные результаты по определению плотности дефектов используются при прогнозировании надежности ПО.

ТМ «Прогнозирование надежности» реализует метод прогнозирования значения надежности по каждому модулю системы по следующей модели надежности:

$$R_i = \exp[\, -D_i\, I_i \cdot (1 - \exp(\, \frac{\rho_i \cdot K}{I_i \cdot \varphi_i} \cdot t))] \,,$$

где ρ_i – параметр среды эксплуатации i-го модуля, φ_i – характеристика среды ее разработки, I_i – оцененный размер начального кода, а D_i – прогнозируемая плотность дефектов в системе.

Коэффициент дефектов K – константа, предвиденная для всех объектов ПС, а значения ρ_i и φ_i – известны на момент первоначального прогнозирования надежности, они не изменяются во время разработки компонентов системы.

ТМ «Оценка надежности системы» согласно классификации дефектов (Orthogonal Defect Classification), в соответствии с которой для каждого выявленного дефекта определяются параметры: тип дефекта, триггер дефекта, влияние дефекта. Эти параметры используется одной или двумя подходящими моделями надежности из выше приведенного в целях проведения оценки прогнозного значения надежности отдельных модулей и системы в целом. Результаты оценки сравниваются, и выбирается из них наиболее правдоподобная модель.

ТМ «Оценка качества ПО систем»

Под качеством ПО понимается совокупность свойств, обуславливающих его пригодность к использованию по назначению при заданных требованиях и условиях функционирования.

После 1992 года вопросы разработки и гарантии качества программных средств обсуждались в течение 20 лет на международных форумах и конференциях: Conference on

Dependable Systems and Networks (DSN), European Dependable Computing Conference (EDCC), International Conference on Computer Safety, Reliability and Security (SAFECOM), Probabilistic Safety Assessment and Management Conference (PSAM), Dependable Systems, Services and Technologies (DESSERT), Conference on Dependability of Complex Systems (DepCoS) и др.

В 2004 году ассоциация IEEE издает журнал «Dependable and Security Computing». В нем обсуждаются вопросы применения новых стандартов для бизнес-критических приложений, электронной коммерции, банковских технологий и др. В рамках этих стандартов проводится оценка надежности и качества программных продуктов, опубликована статья по технологии обеспечения качества, основанная на интерфейсах и готовых программных ресурсах. Это показывает важность проблемы обеспечения качества программных продуктов и средств.

Создана модель качества (стандарт ISO/IEC 9126- 1992). В ней определено шесть показателей качества (q-quality) q_1-q_6 (табл.3). Каждый показатель (характеристика) обладает свойством и признаками, которые могут оцениваться экспертным и аналитическим путем. Определение *количественных* характеристик (показателей) качества начинается с ранних этапов жизненного цикла (ЖЦ) согласно международному стандарту ISO/IEC 12207: 2007 «Процессы ЖЦ программного обеспечения». В этом стандарте определен процесс «*управления качеством»,* включающий верификацию, тестирование и обеспечение гарантий качества ПО.

Системы управления качеством на эталонной модели качества включают методы управления процессом проектирования компонентов на процессах ЖЦ, учет текущих технических показателей средств в ходе ЖЦ и окончательную оценку показателей качества по модели качества и их метрикам. Эталонная модель качества представлена шестью базовыми характеристиками, имеющими подхарактеристики, значения которых могут быть определены количественно или качественно.

Таблица 3. Характеристики качества в стандарте ISO/IEC 9126

№	Наименование характеристики	Определение характеризуемых свойств ПС
q1	Функциональность (functionality)	Свойства ПП, обусловливающие способность выполнения функций в соответствии требованиям в процессе тестирования и испытания системы в заданной среде
q2	Надежность (reliability)	Свойства ПП, обусловливающие ее способность сохранять уровень функционирования и низкую вероятность отказов в процессе выполнения
q3	Применимость (usability)	Свойства ПП, обусловливающие ее способность быть понимаемой и удобной для использования в указанных условиях
q4	Эффективность (efficiency)	Свойства ПП для рационального использования выделенных ресурсов при работе системы в установленных условиях
q5	Сопровождаемость (maintainability)	Свойства ПП, которые обеспечивают модификацию, усовершенствование или адаптацию системы к изменениям среды, требований и функциональности.
q6	Переносимость (portability)	Свойства ПП, обусловливающие ее способность быть перенесенным из одной среды в другую.

На основе полученных данных о надежности и других показателях качества (функциональность, эффективность и др.) рассчитывается целевое значение завершенности и полезности системы (ПС), адекватных потребностям заказчика. При этом мера эксплуатационного качества системы определяется функцией полезности вида:

$$Q_{nc} = \sum_{i=1}^{k} a_i \cdot R_i \qquad (3.10)$$

где a_i – мера важности i-й функции системы для процесса, R_i – надежность выполнения функций в заданном периоде t эксплуатации системы.

Данные по всем показателям качества (q-quality) q_1-q_6 в табл. 3 оцениваются по формуле:

$$q_1 = \sum_{j=1}^{6} a_{1j} m_{1j} w_{1j}$$

где a_i - атрибуты каждого показателя качества (i=1-6); m_{ij} – метрики q_i показателя с j-атрибутами качества; w_{ij} - вес i -показателя качества системы с j-атрибутами. Полученные данные по показателям (характеристикам) модели качества и R_i – надежность выполнения функций входят в сертификат качества.

Выводы

В данном разделе прроекта РФФИ 206 - 2020 рассмотрены подходы к оценке надежности технических и программных систем с применением моделей надежности из множества существующих моделей разных видов и типов. Определены основные базовые понятия надежности, обеспечивающие оценку надежности по соответствующим моделям надежности ПС, основанным на времени функционирования и/или количестве отказов (ошибок), получаемых в компонентах в процессах ЖЦ тестирования, системного тестирования и эксплуатации системы. Согласно приведенной классификации моделей надежности процессы обнаружения ошибок в программах носят случайный Марковский и пуассоновский характер и обеспечивают поиск ошибок, дефектов и отказов.

Некоторые модели надежности позволяют прогнозировать число ошибок в процессе тестирования, другие оценивать надежность с помощью функций надежности по данным, собранным на этапах ЖЦ разработки системы и испытания. Для примера приведены экспериментальные данные для оценки интенсивности отказов λ (t) и количества отказов μ(t) с помощью базовых (Мусы, Гоэла-Окомото и др.) и общей модели надежности, собранных данных на этапах ЖЦ и приведены сравнительные оценки результатов оценки.

Дано описание инструментального комплекса модулей ПТМ, обеспечивающих распределение надежности, прогнозирование плотности дефектов и оценки надежности. Приведены показатели качества в стандартной модели ISO 9126 (1-4) и оценки качества, включая измерение показателя надежности, а также других показателей качества, которые входят в сертификат готового продукта.

Литература к разделу 5 проекта РФФИ 206-2020

[1]. Липаев В.В. Надежность программного обеспечения. М.: СИНТЕГ, 1998, 231 стр.

[2]. Липаев В.В. Методы обеспечения качества крупномасштабных программных систем. М.: СИНТЕГ, 2003, 510 стр.

[3]. Майерс Г. Надежность программного обеспечения. М.: Мир, 1980, 360 стр.

[4]. Мороз Г.Б., Лаврищева Е.М. Модели роста надежности программного обеспечения. Киев: Институт кибернетики АНУ, препринт 92–38, 1992, 23 стр.

[5]. Липаев В.В.. Надежность и функциональная безопасность комплексов программ реального времени. Москва, ЗАО «Светлица», 2013, 193 стр.

[6]. Shick G.J., Wolverton R.W. An analysis of computing software reliability models. IEEE Transactions on Software Engineering, vol. SE–4, № 2, 1978, pp. 104–120.

[7]. Shanthikumar J.G. Software reliability models: A Review. Microelectronics Reliability, vol. 23, № 5, 1983, pp. 903–943.

[8]. Goel Amrit L. Software reliability models: Assumptions, limitations, and applicability. IEEE Transactions on Software Engineering, vol. SE–11, № 12, 1985, pp. 1411–1423.
[9]. Musa J.D. Okumoto K. A. Logarithmic Poisson Time Model for Software Reliability Measurement. In Proc. of the 7th International Conference on Software Engineering, 1984, pp. 230–238.
[10]. Yamada S., Ohba M., Osaki S. S–shaped software reliability grows modeling for software error detection. *IEEE Transactions* on *Reliability, vol*. R–32, № 5, pp. 475–478.
[11]. Chulani S. Constructive quality modeling for defect density prediction: COQUALMO. In Proc. of the International Symposium on Software Reliability Engineering (ISSRE'99), 1999.
[12]. Гнеденко Б.В., Коваленко И.Н. Введение в теорию массового обслуживания. М., Наука, 1966, 432 стр.
[13]. Коваленко И.Н., Шпак В.Д. Вероятностные характеристики сложных систем с иерархическим управлением. Известия АН СССР. Техническая кибернетика, no. 6, 1972, стр. 30-34.
[14]. Duval P., Matyas R., Grover A. Continuous integration improving Software quality and reducing risk. Addison Wesley, 2009, 691 p.
[15]. Коваль Г.И. Модели и методы инженерии качества систем на ранних этапах ЖЦ. Реф. дис. ИК НАНУ, 2005, 20 стр.
[16]. Андон Ф.И., Коваль Г.И. и др. Основы инженерии качества программных систем. К.: Наук. думка, 2007, 670 с..
[17]. Горбенко А.В., Засуха С.А., Рубан В.И., Тарасюк О.М., Харченко В.С. Безопасность ракетно-космической техники и надежность компьютерных систем: 2000-е годы. Авиационно-космическая техника и технология, №1(78), 2011, стр. 9-20.
[18]. A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, 2004, pp. 11-33.
[19]. IEC 62628. Guidance on software aspects of dependability. Geneva: IEC, 2011, 63 p.
[20]. ISO 15288:2002. Systems Engineering. Cycle Life Processes of Systems.
[21]. Лаврищева Е.М. Методы программирования. Теория, инженерия, практика. К.: Наукова думка, 2006, 452 стр.
[22]. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование. Основы индустрии программных продуктов. К.: Наукова думка, 2009, 372 стр.
[23]. Лаврищева Е.М. Software Engineering компьютерных систем. Парадигмы, технологии, CASE-средства. К.: Наукова думка, 2014, 284 стр.
[24]. Саати Т. Принятие решений. Метод анализа иерархий. М.: Радио и связь, 1993, 315 стр.

5.9. Принципы обеспечения надежности компонентов Веб- систем

Главной проблемой при разработке Веб-систем является обеспечение надежности функционирования системы и способности системы сохранять свои свойства (безотказность, устойчивость, безопасность и др.) при обработке исходных данных в результаты в течение определенного промежутка времени и при заданных условиях эксплуатации. Для многих веб-приложений и сайтов надежность является главной целевой функцией реализации. К надежности предъявляются высокие требования (недопустимость ошибок, достоверность, защищенность, безопасность и др.).

К настоящему моменту создано большое количество (более 100) математических моделей надежности, учитывающих разные аспекты работы разных видов прикладных систем (возникновение ошибок, сбоев, отказов, дефектов и др.).

Формально модели оценки надежности сложных систем представления знаний базируются на теории надежности и математическом аппарате с допущением некоторых ограничений, влияющих на эту оценку. Источником информации, используемой в моделях надеж-

ности, является процесс тестирования, эксплуатации и обработка разного рода ситуаций, возникающих в них. Ситуации порождаются ошибками в системах и требуют их устранения и продолжения тестирования.

К базовым ошибочным понятиям, которые используются в моделях надежности ПС, относятся следующие.

Ошибка (error) может быть следствием недостатка в одном из процессов разработки систем, который приводит к неправильной интерпретации промежуточной информации, заданной разработчиком или при принятии им неверных решений.

Отказ ПС (failure) – это переход системы из рабочего состояния в нерабочее, или полученные результаты не соответствуют заданным допустимым значениям. Отказ может быть вызван внешними факторами (изменениями элементов среды эксплуатации) и внутренними – дефектами в самой ПС.

Дефект (fault) в системе – это следствие ошибок разработчика на любом из процессов разработки – в описании спецификаций требований, проектных спецификациях, эксплуатационной документации и т.п. Дефекты в программе, не выявленные в результате проверок, является источником потенциальных ошибок и отказов ПС. Проявление дефекта в виде отказа зависит от того, какой путь будет выполняться для нахождения ошибки в коде или во входных данных. Не каждый дефект ПС может вызвать отказ или любой отказ может вызвать аномалию от проявления внешних ошибок и дефектов.

Интенсивность отказов – это частота появления отказов или дефектов в ПС при ее тестировании или эксплуатации.

При выявлении отклонения результатов от ожидаемых во время тестирования или сопровождения осуществляется поиск и выяснение причин этих отклонений для исправления связанных с этим ошибок. Надежность ПС зависит от числа оставшихся и не устраненных ошибок на этапах ЖЦ. В ходе эксплуатации ПС ошибки обнаруживаются и устраняются. Чем интенсивнее проводится эксплуатация, тем интенсивнее выявляются ошибки и быстрее растет надежность системы и соответственно ее качество.

На надежность ПС влияют факторы:

– риск через угрозы, приводящие к неблагоприятным последствиям и ущербу системы или среды;

– угрозы как проявление неустойчивости системы и нарушения ее безопасности;

– причины возникновения угроз или рисков, их частота и последствия;

– способность системы сохранять устойчивость работы и др.

Риск уменьшает надежность, а обнаруженные ошибки могут привести к угрозе, если отказы носят частотный характер.

В виду большого разнообразия моделей надежности, разработано несколько подходов к классификации этих моделей. Эти подходы в целом основываются на истории ошибок проверяемой и тестируемой ПС. Одной из классификаций моделей надежности ПО является классификация Хетча.

Прогнозирующие модели надежности основаны на измерении технических характеристик создаваемой программы: длина, сложность, число циклов и степень их вложенности, количество ошибок на страницу операторов программы и др. Например, модель Мотли–Брукса основывается на длине и сложности структуры программы (количество ветвей и циклов, вложенность циклов), количестве и типах переменных, а также интерфейсов. В этих моделях длина программы служит для прогнозирования количества ошибок, например, для 100 операторов программы можно смоделировать интенсивность отказов.

Измерительные модели предназначены для измерения надежности ПО, работающего с заданной внешней средой и следующими ограничениями:

– ПО не модифицируется во время периода измерений свойств надежности;

– обнаруженные ошибки не исправляются;

– измерение надежности проводится для зафиксированной конфигурации ПО.

Типичным примером таких моделей является модель Нельсона и Рамамурти–Бастани и др. Модель оценки надежности Нельсона основывается на выполнении *k*-прогонов программы при тестировании и позволяет определить надежность по формуле

$$R(k) = \exp[-\sum \nabla t_j \lambda(t)],$$

где t_j – время выполнения *j*-прогона, $\lambda(t) = -[\ln(1-q_i)\ \nabla j]$ и при $q_i \le 1$ она интерпретируется как интенсивность отказов.

Оценочные модели основываются на серии тестовых прогонов и проводятся на этапах тестирования системы. В тестовой среде определяется вероятность отказа программы при ее выполнении или тестировании.

Эти типы моделей могут применяться на этапах ЖЦ. Кроме того, результаты прогнозирующих моделей могут использоваться как входные данные для оценочной модели. Имеются модели (например, модель Мусы), которые можно рассматривать и как оценочную, и как измерительную модель.

Согласно Гоэл модель надежности базируются на отказах и разбиваются на четыре класса моделей:

– без подсчета ошибок,

– с подсчетом отказов,

– с подсевом ошибок,

– модели с выбором областей входных значений.

К таким моделям относятся модели Джелински и Моранды, Шика Вулвертона и Литвуда–Верала.

Кроме рассмотренной классификации моделей надежности, имеются и другие классы моделей, например, которые определяются процессами выявления отказов, их интенсивностью и условно их можно разделить на следующие группы:

1) модели, рассматривающие интенсивность отказов как марковский процесс;

2) модели, рассматривающие интенсивность отказов как пуассоновский процесс;

3) модели роста надежности.

Четкой границы между этими моделями нет, однако по фактору распределения интенсивности отказов и их поведению эти модели можно еще разделить на экспоненциальные, логарифмические, геометрические, байесовские и др.

Измерение роста надежности ПС по результатам тестирования требует выбора и объединения разных моделей оценки надежности в течение ЖЦ, которые задают изменение надежности ПС во времени, а также адекватного процесса тестирования, как главного условия роста надежности. Оно состоит в том, что тестовые данные и функции систем выбираются соответственно частоте их ожидаемого использования, а среда тестирования максимально приближена к условиям эксплуатации.

В инженерии надежности определен новый термин dependability, означающий гарантоспособность (пригодноспособность) системы в широком смысле [31]. По сравнению с термином reliability новый термин обозначает способность системы обладать свойствами, желательными для пользователя и дающими ему уверенность в качественном выполнении функций, заданных в требованиях к системе.

Dependability добавляет дополнительные атрибуты, которыми должна обладать система:

– готовность к использованию (availability);

– готовностью к непрерывному функционированию (reliability);

– безопасность для окружающей среды, т.е. способность системы не вызывать катастрофических последствий в случае отказа (safety);

– секретность и сохранность информации (confidential);

– способность к сохранению системы и устойчивости к самопроизвольному ее изменению (integrity);

– способность к эксплуатации ПО, простоту выполнения операций обслуживания, а также устранения ошибок, восстановление системы после их устранения (maintainability) и т.п.;

– готовность и сохранность информации (security) и др.

Таким образом, показано, что надежность является одной и главных характеристик создаваемых веб-систем и сайтов.

5.10 Функция надежности модели Дж. Мусы в управлении качеством ПС

В работе An approach to the software quality management. E. M. Lavrishcheva, G. I. Koval and T. M. Korotun 2006, Volume 42, Number 5, Pages 758-768 в качестве ключевой характеристикой качества ПС взята *завершенность (зрелость)* - свойство системы избегать отказов при наличии скрытых дефектов. Эта характеристика используется в процессе управлению качеством. Для эффективного управления качеством по этой разработана *модель качества*, которая устанавливает взаимосвязь мер и соответствующих метрик внутреннего, внешнего и эксплуатационного качества ПС. На начальных этапах разработки ПС специфицируются область значений *внешних метрик для* достижения установленного уровня качества при испытании ПС, а затем определяются наиболее пригодные (с точки зрения прогноза значений внешних метрик) *внутренние метрики* для достижения внешних требований к качеству. В контексте *завершенности* ПС главным показателем ее внутреннего качества являются дефекты, внешнего – отказы.

Для ПС задана модель требований к завершенности компонентов ПС, построенная с учетом дифференцированного подхода и необходимости максимизировать уровень достижения установленных целевых значений завершенности компонентов ПС, адекватных потребностям заказчика и пользователей.

Мера эксплуатационного качества ПС как функция полезности вида: $Q_{nc} = \sum_{i=1}^{k} a_i \cdot R_i$

где a_i – мера важности i-й функции ПС процесса, R_i – надежность (безотказность) выполнения функции в заданном периоде t эксплуатации системы.

Безотказность выполнения функций ПС оценивается во время системного тестирования и эксплуатации ПС. Проблема нахождения оптимального уровня завершенности декомпозируется по четырехуровневой иерархии:

ПС $\rightarrow$ *функции ПС* $\rightarrow$ *программные приложения* $\rightarrow$ *программные модули.*

Задача состоит в априорном нахождении оптимального целевого уровня *завершенности* каждого компонента (модуля и ПС) системы, который обеспечивает максимизацию функции полезности Q_{nc} с учетом технических и ресурсных ограничений ПС при условии:

u_i – коэффициент относительного веса функции F_i в достижении эксплуатационного качества Q_{nc}, $i = 1,\ldots, k$;

v_{ij} – коэффициент относительного веса j-го ПС в обеспечении выполнения i-й функции, $u = 1,\ldots, k;\ j = 1,\ldots, l;$

w_{js} – коэффициент относительного веса s-го модуля в обеспечении выполнения j-го программного приложения, $s = 1,\ldots, m; j = 1,\ldots, l$;

r_s – безотказность модуля M_s в период эксплуатации t;

E_j – множество номеров модулей, которые необходимы для выполнения j-го ПС;

α_s – нижняя граница безотказности модуля M_s;

β_s – верхняя граница безотказности модуля M_s;

G – общая цена ПС;

C – себестоимость создания ПС организацией-разработчиком;

c_s – накладные расходы, связанные с разработкой модуля M_s ;

d_s – расходы, необходимые для достижения единичного уровня безотказности модуля M_s (с учетом или без учета затрат на тестирование, а также косвенным учетом времени на разработку);

δ – доля прибыли в цене ПС.

Определяются целевые значения *безотказности* модулей $r_1 \ldots r_m$, при которых функция полезности Q_{nc} достигает максимума:

$$Q_{nc}(r_1,...,r_m) = \sum_{j=1}^{l}(\sum_{i=1}^{k} u_i v_{ij} \cdot \prod_{n \in E_j} r_n) \to \max \qquad (1)$$

при ограничениях $\quad 0 < \alpha_s \le r_s \le \beta_s \le 1 \qquad s = 1,..., m \qquad (2)$

$$c_s + d_s \cdot r_s \le (1-\delta) \cdot G \cdot \sum_{j=1}^{l}\sum_{i=1}^{k} u_i v_{ij} w_{js} \qquad (3)$$

$$\sum_{s=1}^{m}(c_s + d_s \cdot r_s) \le C \qquad (4)$$

Данная задача нелинейной оптимизации с линейными ограничениями (2) – (4) практически решается с помощью математического пакета MATLAB.

Параметры u_i, v_{ij}, w_{js} ($u = 1,\ldots, k$; $j = 1,\ldots, l$; $s = 1,\ldots, m$) находятся методом анализа иерархий (МАИ) путем парного сравнения и последовательного определения локальных приоритетов компонентов ПС в пределах каждого уровня иерархии по отношению к компонентам предыдущего (высшего) уровня.

Ограничения (2) задают допустимые нижние α_s и верхние β_s границы безотказности модулей, исходя из оценок важности каждого модуля.

Ограничения (3) устанавливают взаимосвязь общих расходов на разработку модуля, с одной стороны, и части цены системы G, которая приходится на этот модуль с учетом его вклада в надежность работы системы, и с учетом доли прибыли δ разработчика, с другой стороны. Допускается линейная зависимость между стоимостью модуля и уровнем его безотказности.

Ограничение (4) устанавливает взаимосвязь суммарных расходов на разработку всех модулей и себестоимости создания ПС. Расходы на разработку модуля состоят из накладных (косвенных) и прямых расходов непосредственно на его разработку.

Таким образом, с помощью данной модели устанавливаются требования к завершенности каждого модуля (r_i), а затем и каждого программного приложения (q_i) (с учетом независимости модулей в структуре ПС, то есть определяются целевые уровни завершенности всех ПС, достижение которых должно контролироваться в ходе выполнения проекта.

Контроль достижимости целевого уровня завершенности q_i, установленного для каждого i-го ПС, базируется на построении прогнозов двух видов: поискового прогноза, , и нормативного. Для обеспечения выполнения прогнозов разработаны:

– модель для раннего *прогнозирования безотказности* ПС (внешняя метрика качества ПС в контексте завершенности);

– модель для раннего *прогнозирования скрытых дефектов* в ПС (внутренняя метрика качества ПС для завершенности;

– метод *анализа альтернатив* достижения целевого уровня качества ПС;

– модель для определения *оптимального времени тестирования* компонентов ПС с учетом риска их отказов.

При раннем прогнозировании определяется минимальную плотность дефектов к моменту начала системного тестирования. *Позднее» прогнозирование* безотказности ПС связывается с применением аналитических моделей роста надежности на этапе системного тестирования после сбора определенного количества данных об отказах ПС. Все известные

модели роста надежности ПС фактически охватывают период *после* раннего прогнозирования, когда надежность повышается в результате тестирования и устранения дефектов. При прогнозировании безотказности ПС процесс отказов целесообразно моделировать неоднородным *пуассоновским процессом*.

Для пуассоновских моделей надежности условная *функция надежности* $R(t|T)$ определяется по формуле: $R(t|T)=\exp((m(T+t)-m(T))$,

где $R(t|T)$ - условная вероятность того, что в течение заданного времени *t* эксплуатации ПС в определенных условиях среды его функционирования не возникнет последовательность входных данных, которая приведет к отказу, если ПС тестировалось в течение времени *T*;

m(t) – функция роста надежности, представляющая собой среднее количество дефектов в ПС, выявленных во время его функционирования в течение времени *t*.

Сравнительный анализ ряда пуассоновских моделей с позиций их эффективности для раннего прогнозирования безотказности ПС показывает целесообразность использования экспоненциальной модели Дж. Мусы. Эта модель не требует данных об отказах и пригодна для раннего прогнозирования эксплуатационной надежности ПС в начале системного тестирования.

Функция роста надежности определяется так: $m(t) = N_0(1 - \exp(-\frac{\lambda_0}{N_0} \cdot t))$,

где N_0 – количество скрытых дефектов в ПС в начале системного тестирования;

λ_0 - интенсивность отказов ПС в начале системного тестирования $\lambda_0 = N_0 \cdot \frac{\rho \cdot K}{I \cdot \varphi}$,

где ρ – интенсивность выполнения кода; $K=4\cdot10^{-7}$ – коэффициент выявления дефектов (постоянный для модели Дж. Мусы); *I* – количество инструкций исходного кода;

φ – коэффициент расширения кода.

Условная функция надежности эксплуатации ПС имеет вид:

$$R(t \mid T) = \exp(-(m(T+t) - m(T))) = \exp[-N_0 \exp(-\frac{\lambda_0}{N_0} T)(1 - \exp(\frac{\lambda_0}{N_0} \cdot t))] \cdot \quad (5)$$

Если *T=0,* то ПС не будет пребывать на этапе системного тестирования,

$$R(t) = R(t|0) = \exp[-D_0 \cdot I \cdot (1 - \exp(\frac{\lambda_0}{D_0 \cdot I} \cdot t))], \quad (6)$$

где $D_0 = N_0/I$ – плотность скрытых дефектов в начале тестирования.

Формулы (5) и (6) – это метрики раннего прогнозирования безотказности ПС с учетом времени его тестирования. *Размер* ПС(*I*) можно определять одним из методов в методологии FPA (Function points analysis) в условных единицах функциональности и конвертировать полученное значение в единицы KSLOC (тысяча строк некомментированного исходного кода).

Для прогнозирования *вероятной плотности дефектов* D_0 (или количества дефектов N_0), которую будет иметь ПС на момент начала системного тестирования, могут использоваться существующие мультипликативные модели, например, модель Rome Laboratory. Для выполнения прогнозов дефектов на практике используются графические модели. С помощью графических моделей, в основе которых лежат байесовские сети, можно сформулировать предположение о существовании зависимости между различными факторами качества.

Основные преимущества использования графических байесовских моделей для управления качеством – поддержка *прогнозирования* дефектов и *диагностики* причин их возникновения с помощью эффективных алгоритмов и доступных инструментов.

Одна из базовых графических моделей прогнозирования дефектов является модель дефектов для определения множества факторов качества, анализа причинно-следственных связей между ними, комбинирования качественных (экспертных) и количественных оценок их влияния на плотность дефектов с учетом ограничений выбранного *доступного* инструмента моделирования Hugin Lite 6.5.

Модель позволяет прогнозировать плотность дефектов D_{0i} в *i*-м ПС, если не изменятся условия его разработки. Прогнозное значение плотности дефектов используется для нахождения вероятного прогнозного значения безотказности ПС $R_i(t)$.

Анализ альтернатив достижения целевого уровня завершенности *i*-го программного приложения базируется на сравнении прогнозируемого значения безотказности $R_i(t)$ с установленным значением q_i.

Для проведения анализа должны быть предварительно определены такие *критерии* для принятия решений по управлению качеством *i*-го ПС:

q_i – установленный *целевой уровень безотказности i*-го ПС;

D_i^* - максимальное значение плотности дефектов в ПС, которое не будет препятствием для достижения целевого уровня q_i *дефектов* в ПС, определяется из уравнения $q_i = \exp[-D_i^* I_i \cdot (1 - \exp(\frac{\lambda_{0i}}{D_i^* I_i} \cdot t))]$;

L_0 - минимальная вероятность прогнозного значения плотности дефектов; максимально допустимые *отклонения*:

σ_r - прогнозного значения безотказности $R_i(t)$ от целевого значения q_i;

σ_d прогнозного значения плотности дефектов D_{0i} от значения D_i^*;

σ_d - полученной наибольшей вероятности прогнозного значения плотности дефектов $L(D_{0i})$ от приемлемой L_0.

Для обеспечения качества *i*-го ПС следует выполнить следующие действия при $q_i \geq R_i(t)$ и $D_{0i} \geq D_i^*$:

Таким образом, результаты раннего прогнозирования безотказности ПС незначительно отличаются от оценок по данным об отказах, полученным во время тестирования и опытной эксплуатации приложений. Достигнутое качество прогнозов удовлетворяет общепризнанному критерию в инженерии качества $PR(0.25) = 0.75$.

Здесь PR - показатель качества прогноза: $PR(\varepsilon) = \frac{k}{n}$, где ε - допустимая относительная погрешность прогнозирования, n – количество ПС, k – количество ПС, для которых погрешность прогнозирования не превышает ε.

Усовершенствование моделей раннего прогнозирования дефектов для повышения точности оценок приводит к использованию модели позднего прогнозирования роста надежности по данным об отказах методами максимального правдоподобия, наименьших квадратов и т. п. Наряду с характеристикой *завершенность* “maturity” рассматривалась *безопасность информации* “security”, которая является подхарактеристикой *функциональности* “functionality” ПС и состоит в определении внутренних и внешних метрик характеристик качества для установлении *взаимосвязи* между ними.

Исследование вопросов обеспечения надежности и безопасности создаваемых Веб-сайтов и приложений в среде Интернет показывает, что для клиент-серверной архитектуры браузера Интернет, использующего сервисные модели SOA. SCA, SOAP и др. Интернета, а также широко используемых систем (Cloud computing, Dig Data, OWL, Ontiology и др.) для широкого пользователя необходимо проводить проверку правильности и надежности их функционирования, защиты данных в БД и в других хранилищах. Сайты должны быть защищены от всевозможных угроз и атак со стороны Интернета. Данный подход к созданию Веб-систем представлен на сайте ИСП РАН (www.ispras.ru/lavrischeva/, http://7dragons.ru/ru).

На статьи по вопросам инженерии и технологии Веб-систем много запросов в среде Интернет из Китая, Тайланда, Индии и др. Сегодня интеллектуализация касается Больших данных (Big Data). Это направление особенно стало актуальным в связи с роботизацией разных сфер человеческой деятельности, в том числе работы с огромными объемами данных, получаемых с космоса, недр земли, океана и др. (См. статья. Лаврищева Е.М. Рыжов А.Г. Применение теория общих типов данных стандарта ISO/IEC 12207 GDT применительно к Big Data.- The cconference “Actual problems in science and ways their development”, 27 desember 2016, http://euroasia-science.ru/ p.99-110.

5.11. Технология сборки интеллектуальных и информационных ресурсов с обеспечением качества и безопасности

Технология сборки разнородных ресурсов (интеллектуальных и информационных) модулей реализована в 70-80 годы XX- столетия при создании многочисленных специализированных комплексов программ в рамках ВПК (Липаев В.В.). Метод сборки модулей в разных языках программирования - ЯП (Algol-60, Fortran, Cobol, PL/1, Modula и др.) основывался на разработанных примитивных 64 функциях преобразования неэквивалентных обмениваемых данных между модулями в АПРОП ОС ЕС (IBM-360) и передан в ЕрНУЦ (1985). Эти средства адаптированы в другие системные среды — Sun Microsystems, .Net, VS.MS, Oberon, IBMSphere, Intel, Unix и др.

Сформировалось сборочное программирование. По словам А.П. Ершова, «сборочное программирование обеспечивает построение уже существующих (проверенных на правильность) готовых отдельных фрагментов КПИ модулей (типа reuses) в сложную структуру». Интерфейс модулей описывался в специальном языке описания связи в link, а затем в 90-х появились языки описания связей модулей (IDL, API, ABI, WSDL и др.) и реализованы в разных системных средах: *link* IBM, .Net, Corba; *make* BSD, GNU, MSBuild(.NET), ApacheAnt (Java, C#)); *config* REST, CDR, SPAROL; building, assembling, config SWMS, Grid, Semantic-OGSWA, OML, JSON; *weaver* BEA WebLogic Oracle, SAP Net, ASP и др.

Метод сборки разных ресурсов модульного типа апробирован в проекте РФФИ 16-01-00352 (2018) и получен экспериментальный вариант ядра OS Linux с помощью операций Kconfig стандарта IEEE 828-2012 в итоговом отчете НИОКТР 11020510073 от 5.02.16 N 16-01-00352. Отчет «Теория и методы разработки вариабельных программных систем». Рубрика 20.01.07. 002.6-027.21; 002.6.001.8 от 05.03.19).

Обобщением разных вариантов метода сборки стали фабрики программ в индустрии прикладных систем разного назначения:

Product Line/Product Family – конвейерная сборки ресурсов (K. Pochl).

Мультисборка генерации программ и данных (K. Czarnecki, A.Ezenaker).

AppFab of Enterprise VS.MS and IBMSphere

(https://www.ibm.com/developerworks/ru/websphere/newto/).

Потоковая сборка разноязычных модулей Дж. Гринфильд and Г.Ленц.

Сборка разноязычных модулей в ЯП (C, C++, Basic, Fortran и др.) И.Бей.

Непрерывная интеграция модулей (Continuous integration) М. Фаулер.

Конвейерная сборка технических и программных средств В.М.Глушков.

Фабрика ПО Веб-служб (p&h) GAX, сборка SAP NetWeaver и др.

Инструменты интеграции, сборки из качественных объектных и сервисных ресурсов обсуждались в докладе «Информатика-70» и на конференции OS DAY ISPRAS-2018. В них отображены вопросы информатизации и технологии программирования интеллектуальных, информационных и программных систем (ПО, ОС) из готовых ресурсов, которые накоплены в огромном количестве в репозиториях, библиотеках Reusability, Mat.Lab, SMT Интернет и

др. Эти доклады находятся в https://videonauka.ru/stati/30-metodika-prepodavaniya-tekhnicheskikh-istsiplin/237 (238).

Здесь рассматриваются подходы к интеллектуализации компонентов, сервисов (SOA) и сервисных компонентов (SCA) в рамках существующих систем сборки: link модулей АПРОП (CORBA); make компиляторных программ BSD, GNU, MSBuild; assemble, Kconfig системных и вычислительных ресурсов EuroGrid; integration, config сервисных SOA, SCA ресурсов Semantic Web; weaver BEA WebLogic Oracle, SAP Net. с обеспечением качества и безопасности систем (в рамках проекта РФФИ 19-01-00206-2019). Дан анализ систем сборки и обосновывается концепция создания общего сборщика ресурсов в Интернет, объединяющего все варианты сборки в общий процесс с обеспечением качества и безопасности https://videonauka.ru/stati/30-metodika-prepodavaniya-tekhnicheskikh-distsiplin/238.

Подходы к интеллектуализации систем и ресурсов

WWW и Семантик Веб Интернет ориентированы на создание интеллектуальных систем. В них процесс управления научными знаниями состоит в:

– *отображении знаний* в Базах Знаний и давать ими пользоваться другим;

– *моделировании* (knowledge modelling) знаний для их использования при решении конкретных прикладных задач;

– п*оиске и выборе* (knowledge retrieval) *знаний* из различных хранилищ в подмножество контента для релевантного решения конкретной задачи;

– *повторном использовании* (knowledge reuse) знаний в виде готовых описаний, образцов (patterns), моделей в разнообразных контекстах;

– *публикации* (knowledge publishing) знаний в стандартизованной форме для последующего распространения;

– *обновлении* (knowledge maintenance) и сохранении знаний в Базах Знаний;

– *приобретении* (knowledge acquisition) знаний для анализа неструктурированной информации (тексты, изображения, сценарии и др.), преобразования неявных знаний в явные, решения задач обработки данных огромных объемов и сохранения их в Базы Знаний.

Аспектом интеллектуализации информационных ресурсов (документов, текстов, таблиц, страниц и др.) являются процессы:

– форматирование данных больших объемов в БД глобального масштаба с обеспечением сохранности, защищенности и безопасности;

– поиска, выбора фрагментов данных и их транспортирование по запросам пользователей;

– анализа, интеграции, агрегации информации (например, документов, в требуемые структуры для обработки на разных уровнях управления организациями страны и др.) для проведения вычислений задач и др.

5.12. Интеллектуализация знаний

К современным средствам представления знаний в терминах дескриптивной логики относятся: *FaCT* (на LISP), *FaCT++* (на C++), *CEL*, KAON-2 (JAVA), *MSPASS* (C) и *Pellet* (JAVA) и др. Необходимой составляющей представления знаний являются *прикладные web-сервисы*, ориентированные на коллективное решение задач в предметных областях знаний в современных средах (problem-solving environments, PSE) в мировом сообществе.

В процессе создания *моделей знаний* предметных областей используются система KBS (Knowledge-based systems), CommonKADS и др. Они обеспечивают построение библиотек знаний, содержащих элементы решения задач, которые затем могут повторно использоваться и в других областях знаний.

Исходя из того, что программа на ЯП – это некоторый формальный или математический текст, то предложена система управления текстовой информацией и *терминологией* (System Quirk – SQ, http://www.computing.surrey.ac.uk/SystemSQ. Эта система ориентирована на создание и ведение терминов в терминологической базе данных (ТБД) и баз знаний (БЗ), а также организацию коллекций текстов на компьютере с помощью инструментов Virtual Corpus, KonText, Ferret, Grid и др.

Для ведения больших объемов данных и обеспечения функционирования систем в глобальной сети Интернет используются системные, сервисные и коммуникационные ресурсы, включая средства Семантик Веб. *Семантик-Web* — эта информационная среда в World Wide Web, которая предоставляет семантические ресурсы, языки, средства и инструменты разработки онтологий предметных областей знаний, прикладных систем и бизнес-процессов с использованием накопленных знаний в среде http://semwebprogramming.org.

В этой среде проводится автоматизированная обработка научных задач, больших данных в разных форматах, интеграция данных из коллажей (Mash-Ups), поиск и композирование веб-сервисов, управление интеллектуальными агентами в мобильных приложениях и др.

Одним из инструментов интеллектуализации знаний в Семантик Веб о разных предметных знаниях (математики, физики, биологии, медицины и др.) является онтология (www.semantic.web/ontology).

В основе онтологии представления знаний о некоторой области знаний лежит понятийная база, совокупность концептов (понятий) и отношений между ними, классификация понятий и их таксономия в виде тезаурусов, а также методы создания профильных онтологий. О*нтология* является инструментом построения концептуальной модели для областей знаний/доменов, которая включает объекты, классы объектов, структуры данных, связи и отношения (теоремы, ограничения) применительно к конкретной области знаний.

Имеется опыт разработки онтологии домена SE - ЖЦ ISO/IEC 12207 -2007 в языке OWL (Web Ontology Language) в среде Protégé 2.3 с выходным результатом в XML. Подход к автоматизации ЖЦ докладывался на Международной конференции «Science and Information-2015» (www.conference.thesai.org) в Лондоне и на XVII Всероссийской научной конференции-2015, 2016 «Научный сервис в сети Интернет» в г. Новороссийск, ИПМ им. М.В. Келдыша.

Таким образом, Семантик Веб предоставляет не только средства создания онтологий для предметных областей знаний, но и способ решения отдельных задач с использованием многочисленных сервисов и научных ресурсов по разным предметным областям в Интернет, в том числе и по бизнес-услугам. Для применения больших объемов информации при решении научных задач предметной области могут использоваться словари и концептуальные модели и приложений предметных областей знаний с помощью предметно-ориентированных языков (OWL, DSL и др.), инструментов FODA, Protégé, DSL Tool, KBuild, Kconfig (http://www.Sap.org) и др.

Интеллектуальные и информационные ресурсы

Технология сборки интеллектуальных и информационных ресурсов – это процессы сборки ресурсов (модульных элементов в разных ЯП), структур данных, процедур, классов, компонент и сервисов.

Ресурсы разрабатываются для разных прикладных областей знаний (математика, медицина, биология, генетика и др.). Они накоплены в библиотеках, хранилищах Интернет и в общесистемных средах (IBM, MS.Net, Unux, Intel и др.).

К хранилищам относятся библиотеки, репозитории, базы данных (БД). Библиотеки процедур, Базы знаний, Reusability Libraries и др. Функциональные элементы типа reuses создают высокоинтеллектуальные специалисты в области информатики, математики, биологии и др. (например, MatLab, Demral и др.). Каждый готовый reuse при размещении в библиотеки общего использования проверяется на правильность и качество реализации функции.

Интеллектуальные ресурсы (ИНР) – это компоненты повторного использования (КПИ и reuses), отображающие знания специалистов в разных областях знаний. Любая функция ПрО описывается в ЯП в виде модуля (объекта, компонента, сервиса и др.) и может взаимодействовать через link с другими модулями через операции типа CALL/RPC/RMI в текстах программ, в параметрах которых задаются данные для обмена между модулями, в IDL, API, ABI, WSDL и др. КПИ прикладных задач включают информационные данные, которые относятся к фундаментальным (FDT) и общим типам данных (GPD) стандарта ISO/IEC 11404, а также могут оперировать с большими объемами данных (Big Data). ИНР взаимодействуют с модулями, объектами или сервис-компонентами Интернет через интерфейс в языках IDL, API, ABI, WSDL и др.

Reuse — это готовый интеллектуальный программный объект, компонент, сервис, реализующий алгоритм функции в некоторой предметной области знаний. Он специфицируется в стандартном языке WSDL и может многократно использоваться, если удовлетворяет функциональным требованиям отдельных предметных областей. КПИ как Reuses имеет код (implementation) с интерфейсом (interface) и схемой развертки (deployment) для выполнения. КПИ, Reuses могут иметь общие переменные, которые образуют **классы** или **множества**. Внешние общие переменные и методы класса задаются в интерфейсе экземпляров класса.

Информационные ресурсы (ИР) – это данные разного объема, в том числе и Big Data, представлены в Базах Данных, файлах, каталогах**,** таблицах, документах и т.п. ИР обрабатываются сервисными, системными, клиентскими и серверными службами Интернет и помещаются в Хранилищах данных, БД малых и больших объемов со структурированными и неструктурированными данными.

Интерфейс — это спецификатор информационной части ИНР — КПИ, компонента, reuses для обращения к другим ресурсам и обмена данными между ними. Интерфейс содержит вызов метода или функции (RPC/RMI) с параметрами, которые управляют внешними переменными экземпляра класса (выборка значения *get-метод,* присвоение значения *set-метод*, *Home-интерфейс* в языке JAVA и др.) при их взаимодействии. Интерфейс описывается в языке WSDL и в общем случае содержит:

- *название* функции (метода) и *ID* — идентификатор ресурса;
- *описание (спецификацию)* функции средствами ЯП;
- *параметры* (входные и выходные) для передачи данных другим КПИ;
- *язык* описания КПИ (C, C++, Java, Python, Ruby и др.);
- *необязательные атрибуты* (дата, состояние, версия, право доступа, автор, срок использования и т.п.).

Среда программирования Eclipse позволяет автоматически создавать описания ресурсов на основе классов языка JAVA. В этом языке определены следующие основные типы данных:

7) строки (xsd:string);

8) целые числа (xsd:int, xsd:long, xsd:short, xsd:integer, xsd:decimal), числа с плавающей запятой (xsd:float, xsd:double);

9) логический тип (xsd:boolean);

10) последовательности байт (xsd:base64Binary, xsd:hexBinary);

11) дата и время (xsd:time, xsd:date, xsd:g);

12) объекты (xsd:anySimpleType).

В качестве переменных могут использоваться множества, последовательности, включающие фиксированное количество переменных простых типов.
Типичный WSDL-файл имеет следующую структуру.

```
<wsdl:definitions [.]>
<!-- Декларация типов, которые используются в сервисе -->
<wsdl:types>
<element name="someMethod">
<complexType>
<sequence>
<element name="arg0" type="xsd:double"/>
<element name="arg1" type="xsd:boolean"/>
</sequence>
</complexType>
</element>
<element name="someMethodResponse">
<complexType>
</wsdl: types> …
```

Приведенное WSDL-описание задает веб-сервис MyService с единственным методом String someMethod (double arg0, boolean arg1). На его основе можно сгенерировать два типа данных, которые соответствуют входным и выходным параметрам метода. Эти типы применяются в описаниях someMethodRequest и someMethodResponse — входного и выходного сообщений для операции someMethod.

Операции декларируются в описании интерфейса сервиса (декларация wsdl:portType) и затем дается описание привязки сервиса к протоколу SOAP (Simple Object Access Protocol) через декларацию wsdl:binding. При этом устанавливается способ вызова <wsdlsoap:body use="literal"/> с параметрами, заданными в методе класса. В конце WSDL-файла приводится декларация веб-сервиса (<wsdl:service>), в которой содержится информация о расположении <параметр location>**.**

Сервисные и информационные ресурсы Интернет

Web-сервисы основаны на открытых стандартах Интернет, которые широко поддерживаются на платформах Unix, Intel, Windows, IBM, Linux и др. и задаются PHP, ASP, JSP-скрипт, JavaBeans и др.

Формат запросов к Web-сервисам определяет протокол SOAP, который задается в XML и отправляется через HTPP к Web-серверу. IBM, Microsoft и Universal Description, Discovery and Integration (UDDI) способствуют созданию общего каталога Web-сервисов. SOAP, XML и UDDI обеспечивают также надежность функционирования приложений из Web-сервисов и сервис- компонентов.

К средствам создания прикладных систем в Интернет относятся сервис-ориентированная архитектура SOA (Service Oriented Architecture) и сервисно-компонентная архитектура SCA (Service-Component Architecture).

SOA задает функциональность (Functions) и качество сервисов (Quality service) на уровнях:

- транспортный (transport layer) для обмена внешними данными на коммуникационном уровне (service communication layer);
- описания сервиса и интерфейса (service description layer) с обеспечением безопасности и защиты (авторизации, аутентификации);
- операций публикации, поиска и вызова сервисов через реестр сервисов.

В Интернет имеется реестр сервисов (рис.1), который содержит описание сервисов в языках Семантик Веб (SOA, SCA, SMC и др.) и обеспечивает регистрацию, поиск и вызов сервиса по запросам поставщиков и потребителей сервисов.

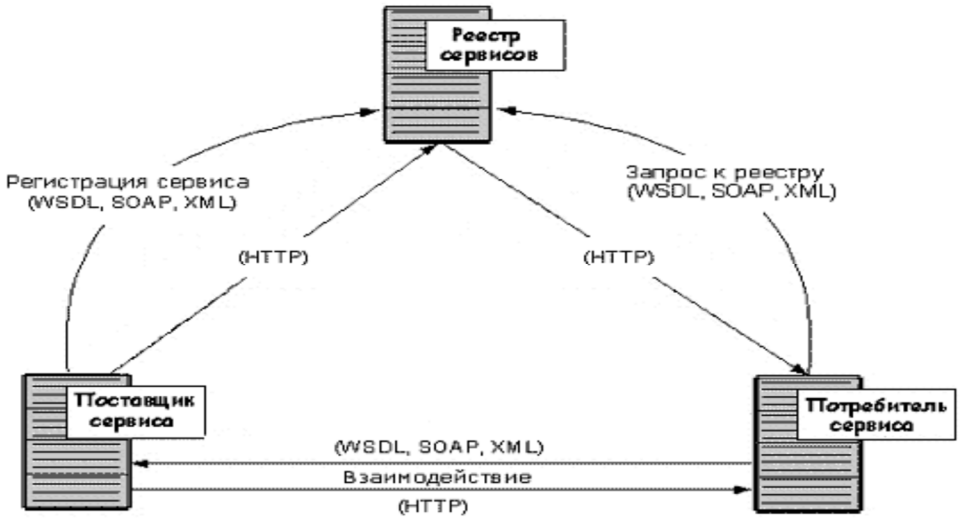


Рис. 1. Служба сервисов в сетевой среде

Рассмотрим сервисы SOA и SCA. Операции над сервисами SOA такие:

5) публикация сервиса через вызов сервиса и его интерфейс;
6) поиск сервиса с помощью протокола SOAP;
7) связь UDDI с моделями CORBA, DBMS, JNET и т. п.;
8) запрос к провайдеру за опубликованными интерфейсами сервиса.

Сервис SCA дает доступ к сервис-компонентам, которые упаковываются в модуль выполнения сервиса в среде WebSphere, эквивалентного EAR-файлу J2EE. Сервисы SCA интегрируются через интерфейс JAVA и реализуются в виде классов JAVA™. В модели SCA объекты данных представлены в JAVA common.sdo.DataObject, который включает в себя метод определения свойств данных. WebSphere Integration Developer платформы Eclipse 3.0 позволяет композировать (собирать) сервисы SCA и сервисные интерфейсы.

Для вызова внешнего сервиса используется JMS, Enterprise JavaBeans или готовые сервисы. Доступ к БД системы проводится через аппарат ссылок. Веб-система собирается из сервисных компонентов, описанных в языках Python, JAVA, BPEL, OWL и интерфейсов с помощью операции конфигурирование Kconfig SAP. Библиотека SCA содержит набор reuses композитных сетевых сервис-компонентовх и гетерогенных данных (http://www.ibm.com/developerworks/websphere/techjournal).

Эти сервисы могут создаваться в среде Java Enterprise Edition, которая позволяет проводить:

- динамическую генерацию серверных страниц (Java Server Pages);
- определение КПИ в виде Enterprise Java Beans;
- преобразование параметров КПИ к формату представления других сред;
- обмен сообщениями (Java Message Queue) со службами JMS (Java Message Service).

В SCA могут создаваться новые сервисные компоненты для проблемных областей знаний, а также объекты данных (Service Data Objects — SDO) и интерфейсы, обеспечивающие передачу данных другим КПИ. Для вызова внешнего сервиса используется JMS, Enterprise JavaBeans или готовые сервисы. Доступ к БД из системы (предприятия, фирмы) проводится через аппарат ссылок. Создаваемая Веб-система собирается из сервисных компонентов, описанных в языках Python, Java, BPEL, OWL и интерфейсов с помощью операцией конфигурирования Kconfig SAP (рис. 2).

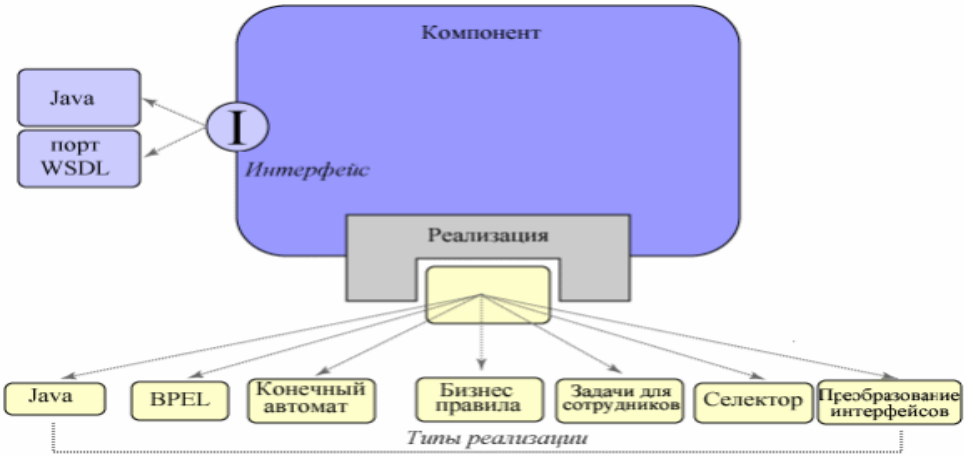


Рис.2. Общая схема сервиса SCA IBM

При работе веб-сервисов с даннями из класса Big Data используются такие средства Интернет: IBM WSDK+WebSphere; Apach Axis+Tomcat; Apach Axis+ Classfish; Microsoft.Net+IIS и др.

Системные ресурсы сети Интернет

К основным **системным ресурсам** Интернет относится клиент-серверная архитектура сети Интернет, которая является трехуровневой: клиент, сервер приложений (систем) и сервер БД.

На уровень *клиента* выносятся простые сервисные ресурсы веб-систем: интерфейс, операции с данными, алгоритмы шифрования, взаимодействия и безопасности для передачи серверу приложений и т.п. Клиентская часть отвечает за интерфейс и обработку приложений Интернет, которые записаны на ЯП (C, C++, Python, Ruby, Java, Basic и др.) и выполняют прикладные функции, обработку возникающих аварийных ситуаций, контроль безопасности и качества и др.

Сервер БД запускается с сервера приложений и выполняет обслуживание БД с обеспечением целостности, сохранности данных при доступе клиента к информации БД. Для работы с Big Data выполняются задачи анализа данных большого объема, а также манипулирования данными с малой и большой нагрузкой.

Клиенту соответствует Интернет-браузер (Chrome, Firefox, MS Internet Explorer, Safari, Opera и др.). Он посылает сообщения серверу через интерфейс следующего вида: $WebAppInterface = \{Request^p\}$, где $Request^p$ — р-й запрос.

Обработка запроса производится компонентами сервера вида: Internet Information Server, Apache, Tomcat, JBoss, WebSphere, WebLogic, Cloudscape.

Web-сервер Apache и JAVA обеспечивают обработку ИНР в современных ЯП и имеют вид: $IR_{APACHE} = \{PERL, PHP, PYTHON, XML, SQL\}$, $R_{JAVA} = \{JAVA, JSP, JSTL, JSF, XML, SQL\}$ в классе системных компонентов (Tomcat, JBoss, WebSphere, WebLogic и др.).

Сервер Интернета включает средства: ASP, JavaScript, VB Script, ASP.NET, .NET, J#.NET, XML, SQL) и языки (C, C++, C#, Python, Ruby и др.).

ИР данных включает: модели данных, операции хранения и доступа к данным, обработку данных и др. Они выполняют доступ к данным и взаимодействие с ресурсами типа entity в модели EJB. При работе с большими объемами данных метод ETL (Extract Transform Load) переносит данные из одного приложения в другое через клиент-серверную архитектуру и проводит извлечение данных из внешних источников, их анализ и трансформацию к виду требований концептуальных моделей и выполнение в среде Интернет.

5.13. Технология сборки сервисных ресурсов в Интернет

Основу технологии составляет метод сборки ресурсов и интерфейс связи разнородных КПИ в архитектуру с взаимно однозначным преобразованием типов входных и выходных данных. Межмодульный интерфейс — совокупность средств формального преобразования разноязыковых КПИ. Интерфейс между КПИ включает набор формальных средств организации обмена данными между модульными элементами. Метод сборки основан на математических формализмах спецификации связей (по данным и по управлению) между разнородными модульными элементами и генерации интерфейсных модулей-посредников для каждой пары связанных элементов. Каждая связь задается оператором вызова CALL/RPC/RMI в ЯП с набором фактических и формальных параметров.

Технология сборки – это метод, средства, инструменты и процесс сборки разнородных ресурсов, который состоит в определении:

– схем, графов и моделей предметной области знаний;
– репозитория модулей (КПИ, Reuses, процедур и др.);
– алгоритмов модулей с операциями CALL/RPC /RMI вызова модулей;
– интерфейсных модулей с параметрами передаваемых данных;
– операций хранения и выбора модулей, КПИ из репозиториев;
– верификации, тестирования модулей и их интерфейсов и др.;
– функциональной надежности и качества ресурсов и систем из них.

В практике программирования сформировались системы сборки модулей с помощью операций сборки: *link* АПРОП (CORBA); *make* компиляторных программ BSD, GNU, MSBuild; *assemble, Kconfig* системных, сервисных и вычислительных ресурсов EuroGrid; integration *config* сервисных SOA, SCA ресурсов в Semantic Web; *weaver* BEA WebLogic Oracle, SAP Net.

Способы сборки описаны в данной книге на стр. 205-209. По ним сделана общая публикация с А. К. Петренко А.К. и Б. А. Позином. На конференции Абрау 19 и сделан на англ. языке.

В разделе 5 описаны метод сборки информационных и интеллектуальных ресурсов и идея общего сборщика этих ресурсов в Интернет.

5.13.1. Индустриальные системы сборки ресурсов

Способ сборки по типу фабрик программ в IBM WebSphere**,** Microsoft Biz Talk, BEA WebLogic Oracle 10g, SAP NetWeaver и ИВК «Юпитер». В них содержатся CASE средства сборки (link) ресурсов (сервисов, компонентов и данных через брокера обмена данными stub и skeleton системы CORBA, переедаемых данные с помощью протокола Workflow с проверкой безопасности и качества (табл. 2).

Таблица 2. CASE–системы сборки ресурсов

Платформа	**Фирмы**	**Содержание платформы**
IBM WebSphere	Корпорация IBM	Сервер приложений J2EE, брокеры обмена данными, КПИ, портал, workflow/BPM, EII, SCA
Microsoft Biz Talk 2004 и компоненты .Net	Корпорация "Майкрософт"	Сервер приложений COM, брокеры обмена данными, RGB доставка, портал, workflow/BPMN
BEA WebLogic	Корпорация "BEA Systems" (с 2008г. входит в "Oracle")	Сервер приложений J2EE, брокеры обмена данными, ГОР, сервер прикладных приложений, портал, workflow/BPMN
Oracle 10g	Корпорация "Oracle" http://www.oracle.com	Сервер приложений J2EE, брокеры обмена данными, ГОР, портал, workflow/BPMN, средства EII

SAP NetWeaver	Корпорация SAP http://www1.sap.com/www.sap.ru	Сервер приложений J2EE/ABAP, брокеры обмена данными, портал, инструменты BPMN
ИВК "Юпитер	Компания ИВК (Россия) http://www.ivk.ru/	Брокеры обмена данными, КПИ, среда выполнения, сертификация, защита данных

На фабриках программ этих систем модифицируется архитектура в соответствии с требуемыми кодами сервисных ресурсов модели SOA в Visual Studio Industry Partners (VSIP). При этом используются модели Guidance Automation eXtensions (GAX) и Guidance Automation Toolkit (GAT) в Visual Studio. GAX — это среда исполнения в VSIP с использованием пакетов рекомендаций. Существует два варианта служб: веб-служба ASP.NET (ASMX) для Windows Communication Foundation (WCF) в среде .NET Framework 3.0. Версии для веб-службы WCF находятся у разработчиков фабрики ПО GotDotN. В ее состав входят процессы предприятия и пакеты документаций и рекомендаций для приложения типа Global Bank. Это аналогично модели SCA в IBM WebSphere.

5.13.2. CASE инструменты и стандарты автоматизации сборки программ

Способ сборки (интеграции) разных программ на CASE инструментах: Make, Apache Ant, Apache Maven, Gradle и др.

Make — это кросс-платформенная система автоматизации сборки систем из исходного кода. Она генерирует файлы управления сборкою, например, Makefile в системах Unix для сборки посредством операции make (см. 3.4.)

Apache Ant — JAVA-утилита для автоматизации процесса сборки программного продукта. Ant — платформо-независимый аналог UNIX-утилиты Make, но с использованием языка JAVA, приспособленного для JAVA-проектов. Самая важная разница между Ant и Make состоит в том, что Ant использует язык XML для описания процесса сборки и его зависимостей, а Make имеет свой собственный формат файл Makefile. XML-файл (или build.xml) осуществляет сборку исполняемых программ.

Apache Maven — программный инструмент для управления (*management*) JAVA-проектами методом сборки (building) ПО аналогично Apache Ant, но с более простой build-моделью конфигурации, которая базируется на формате XML. Двигатель ядра инструмента динамически загружает плагины из репозитория и обеспечивает доступ ко многим версиям разных JAVA-проектов с открытым кодом Apache.

Gradle — система автоматической сборки, построенная на принципах Apache Ant и Apache Maven, но вместо XML-подобной формы здесь используются языки DSL и Groovy для представления конфигурации создаваемого приложения или проекта.

Стандарт для сборки конфигурации систем из есурсов в нентной в среде GDM

Способ сборки ресурсов в индустрии. К. Чернецки создал генерационную мультисборку на ProductLine/ProductFamily. (см. Порождающее программирование. Методы, инструменты, применение, 2005.-750с.). В модель GDM введены новые понятия для представления предметных областей знаний: пространство задач (problem space), пространство решений (solution space) и база конфигурации (configuration base) семейства систем (СПС). *Пространство задач* отображает понятия СПС, членов СПС и их общие характеристики в модели MF (Feature Model), а также функции и задачи, которые описываются GPL (General-Purpose Language) или предметно-ориентированными языками DSL, UML2. *Пространство решений* – это компоненты, каркасы, шаблоны и КПИ реализации задач членов семейства СПС.

Механизмы, правила описания, генерации компонентов и подбора КПИ для отдельных задач СПС входят в *базу конфигурации.* При этом каркас оснащается изменяемыми

параметрами модели, что может привести к излишней его фрагментации и появлению "множества мелких методов и классов". Каркас обеспечивает динамическое связывание аспектов и компонентов в процессе реализации изменчивости между разными приложениями. Образцы проектирования обеспечивают создание многократно используемых решений в различных ПС. Для задания таких аспектов, как синхронизация, удаленное взаимодействие, защита данных и т. д., применяются компонентные технологии ActiveX и JAVABeans, а также новые механизмы сборки

Результаты описания модели СПС в этих пространствах находятся в конфигурационной базе знаний (БЗ). Это связи и характеристики (функциональные или не функциональные), заданные в соответствующей модели MF членов семейства и выполняются операциями конструирования и объединения компонентов в общую ПС или СПС. Иными словами, в БЗ отображены знания о конфигурации системы в виде абстракций общего и специального назначения, КПИ и знания о новых компонентах, результатах их тестирования, измерения и оценивании.

Сборка по модели трансформация и конфигурация

Трансформационная модель приложения или домена описывается в языке DSL (из пространства проблем может превращаться в пространство решений путем трансформации DSL-спецификации модели в реализацию более простых ЯП.

Главный механизм перехода от описания приведенных моделей к исходному результату – трансформация описаний понятий домена в промежуточный язык DSL-пространства решений, а дальше в язык реализации компонентов с учетом платформы, где расположены готовые компоненты и/или новые задачи.

Пространство проблемы1 входят отдельные аспектов проблем. В зависимости от них трансформация может задаваться в языке реализации или другом DSL-языке (фактически, язык другой предметной области знаний).

Модель конфигурации базируется на конструкторских правилах, которые оптимизируют абстракции и характерные черты домена, Модель СПС формируется конфигурационный файл системы.

Описание специфики домена в DSL-языке трансформируется к ЯП компонентов пространства задач для последующей их генерации. При конфигурационном способе представления пространства проблемы с общими характеристиками и ограничениями фактически задаются понятия в проблемно-ориентированном языке и множество компонентов решений в ЯП.

Данными для конфигурационного способа преобразования пространства задач являются понятия предметной области, их характеристики (свойства) и недопустимые сочетания характеристик, согласованные по умолчанию параметров и зависимостей между этими элементами. Созданная из них конфугурационная модель реализуется по правилам конфигурирования и оптимизации в пространстве решений.

После отображения это пространство содержит множество компонентов (КПИ, reuse), схемы сборки готовых компонентов, варианты архитектур некоторых членов семейства.

Все эти элементы входят в конфигурационный файл пространства решений, а также в конфигурационную базу знаний.

При конфигурационном подходе может применяться другой язык описания архитектуры ADL (Arhitecture Description Languages).

По правилам описания архитектуры, трансформация описаний характеристик и ограничений MF производится в описание обобщенной архитектуры семейства ПС в языке ADL. Следующая трансформация описаний компонентов выполняется средствами ЯП. Конфигурационный файл содержит все элементы реализации компонентов и интерфейсов связи компонентов в IDL, а также для выполнения и внесения изменений в структуру ПС и СПС.

5.13.3. Метод к преобразованию типов данных при сборке ресурсов

Сущность теории решения задачи сборки пар разнородных модулей или объектов состоит в построении взаимно однозначного соответствия между множеством формальных и фактических параметров.

$V = \{v^1, v^2, ..., v^\kappa\}$ вызов объекта с множеством формальных параметров;

$F = \{f^1, f^2, ..., f^{\kappa l}\}$ вызов объекта и их отображение с помощью алгебраических систем [2–4].

Каждому типу данных T_α^t языка l_α ставится в соответствие алгебраическая система вида:

$$G_\alpha^t = <X_\alpha^t , G_\alpha^t > ,$$

где X_α^t — множество значений рассматриваемого типа данных (ТД), а G_α^t — множество операций над объектами данного типа. Операциям преобразования ТД соответствует изоморфное отображение алгебраических систем G_α^t в G_β^d.

В классе алгебраических систем $\Sigma = \{G_\alpha^b , G_\alpha^c , G_\alpha^i , G_\alpha^r , G_\alpha^a , G_\alpha^z\}$

где тип t – это b — boolean, c — character, i — integer, r — real, a — array, z — record и др. В системе Σ реализованы все виды преобразований для представленных ТД и проведено доказательство изоморфизма алгебраических систем и его отсутствие для множеств неэквивалентных значений X_α^t и X_β^q. Установлено, что мощности алгебраических систем равны $|G_\alpha^t| = |G_\beta^q|$.

В парадигме сборки реализованы универсальные формальные основы преобразования типов входных / выходных данных (простых и сложных) Fundamental Data Types (FDT) к общим типам данных General Purpose Datetypes (GPD) стандарта ISO/IC 11404, включающего примитивные, агрегатные, генерированные и неструктурированные ТД. Ниже рассматриваются общие вопросы преобразования GPD<=>FDT, как очень важные для сборщика программ в средые Интернет.

Теоретические и практические аспекты преобразования ТД FDT и GPD

В мировых библиотеках Интернет существует большое количество разнородных программных ресурсов для использования при вычислении на компьютерах физических, биологических и других задач.

В этих ресурсах данные могут быть представлены в виде пространственных зрительных образов, отчетов и наборов данных, генерируемых с различных датчиков или специализированной аппаратуры. Такие данные относятся к классу больших данных и при вычислениях требуют нестандартных методов и приемов для их анализа, обработки и организации вычислений.

Общие типы данных – простые, структурные и неструктурированные данными описаны в стандарте ISO/IEC GPD 11404-2007.

Типы данных стандарта GPD ISO/IEC 11404

К общим типам данных (GPD - General Purpose Datatypes) стандарта ISO / IEC 11404-2007 относятся:

– независимые от языка типы данных, которые используются для формального описания концептуальных данных, их элементов и значений;

– полуструктурированные и неструктурированные совокупности данных, в которых ТД являются неизвестной или неопределенной заранее структурой данных;

– расширяемые общие типы данных, сгенеринованные ТД и др.

Стандарт GPD устанавливает номенклатуру и семантику наборов типов данных, которые используются в языках программирования и в интерфейсах программных систем. В этом стандарте специфицированы базовые типы данных и сложные, которые полностью или

частично определяются с помощью простых типов данных. Термин «независимые от языка типы данных» означает, что специфицированные типы данных образуют классы, представители которых в языках программирования (ЯП) соответствуют общей концепции типов данных стандарта GPD ISO/IEC 11404 и частично совпадают с фундаментальными типами данных (FDT) ЯП.

Фундаментальные и общие типы данных программ

Тип данных — совокупность элементов, выделенных на всём множестве предметной области (R. Hindley, 1960). Полиморфный тип — представление набора типов как единственного типа (R. Miller, 1960). Математический тип - это:

- множество всех значений, принадлежащих типу.
- предикат, определяющий принадлежность объекта к данному типу.

Тип данных (ТД) используется в описании программных модулей на ЯП (Algol, Пролог, PL/1, Fortran, Pascal и др.). Аксиоматика ТД разработана С.Дейкстрой, Н. Виртом, В. Турским, П. Науром, А.Замулиным, Н. Агафоновым и др. в 1970-х.

Любые данные, с которыми оперируют программы в ЯП, относятся к следующим стандартным ТД.

- FDT – Fundamental Data Type. Фундаментальные типы данных.
- GPD – Genaral-Purpose Datatypes. Общие типы данных (ISO/IEC 11404 GPD).
- Big Data – Большие данные.

Система типов данных реализована в .Net. В ней представлены *типы-значения (*value type) *и типы-ссылки (*reference type). *Типы-значения* – это статические типы в CTS, их значения могут занимать память от 8 до 128 байтов. Они копируются при присвоении им значений. *Типы-ссылки* используют указатели на объекты, которые они типизируют, а также механизмы хранения и освобождения памяти. Ссылочные типы включают: объектные типы (object type), интерфейсные типы (interface type), типы-указатели (pointer type) и др. (Лаврищева Е.М. Рыжов А.Г. Применение теория общих типов данных стандарта ISO/IEC 12207 GPD применительно к Big Data.- The conference “Actual problems in science and ways their development”, 27 december 2016, http://euroasia-science.ru/ p.99-110.

Реализована теория сборки и преобразования ТД в отечественных работах и сделаны доклады на конференции. «Развитие ВТ в России и в странах бывшего СССР» 3-5 октября 2017 в Зеленограде. История и перспективы. В том числе доклад: Лаврищева Е.М. Развитие теории программ и систем в СССР. История и современные теории.- Сборник SORUCOM-17, 2017. -с.162-176. (Lavrischeva E.M.. Development of the theory programs and systems in the USSR. History and modern theory .- Sorucom-2017, IEEE Springer-2017. P.31-47).

Типы данных стандарта GPD 11404-2007

Общие ТД GPD (General Purpose Datetypes) представлены в стандарте ISO / IEC 11404—2007 и являются:

– независимыми от языка (*Independed Language*) типами данных, используются для формального описания концептуальных типов данных как формализация метаданных для элементов данных, понятий элемента данных и значений областей;

– объединением технологий с текущими ЯП, интерфейсами и изображением данных ЯП (Java, IDL, Express, XML и др.);

– поддержкой полуструктурированных и неструктурированных совокупностей данных, где типы данных и навигация проводятся как для неизвестных или неопределенных заранее ТД, перспективных, устаревших и сохранившихся особенностей, такие как элементы данных и допустимые значения;

– расширяемыми и позволяют ТД GDT описывать их как расширения.

Данный стандарт устанавливает номенклатуру и распределенную семантику для набора типов данных, которые чаще всего используются в ЯП и в интерфейсах программных систем. В стандарте специфицированы как примитивные (базовые, независимые от других)

типы данных, так и сложные, которые полностью или частично определены с помощью простых типов данных.

Понятие «независимый от языков» ТД означает, что специфицированные типы данных составляют классы типов данных, реальные представители которых используются в ЯП и в других объектах на основе концепции типа данных.

Формальный синтаксис GPD

В данном стандарте определен формальый язык спецификации типов данных. Некоторые понятия, полученные на основе Бэкуса-Науровских форм, используются для определения этого языка. Слово «знак» используется для нотации символов, используемых для определения синтаксиса, тогда как слово «символ» используется для ссылок на символы в языке спецификации реальных типов данных.

Стандарт GPD ISO/IEC 11404 определяет ТД, которые частично совпадают с ТД FDT, и включают:

– примитивные ТД (*real*, *integer*, *char*, *boolean*) и другие;

– сложные типы данных (массив, запись, последовательность, портфель, множество, последовательность...);

– сгенерированные ТД — это типы данных, полученные в результате генерации ТД этого стандарта;

– генератор типов данных — это операция, которая создает новый тип данных.

Примитивные ТД GPD

Рациональный (*rational*) — математический тип данных, который отвечает рациональным (действительным) числам.

Масштабированный (*scaled*) — семейство типов данных, пространством значений которого является подмножество пространства рациональных чисел и каждый отдельный тип данных имеет фиксированный знаменатель и предусматривает аппроксимацию его значений.

Комплексный (*complex*) — семейство типов данных, каждый из которых задает числовой математический тип данных в структуре комплексные числа.

Пустой (*void*) — тип данных, который задает объект с необходимыми синтаксическими и семантическими описаниями, но не несет никакой информации.

Основные понятия GPD

Пространство значений в GPD — это совокупность (коллекция) значений типа данных, которая определяется одним из следующих способов:

1) перечислением;

2) аксиоматичным определением;

3) подмножеством уже определенного пространства значений, которое имеет тот же набор свойств;

4) комбинацией любых значений некоторого, определенного пространства значений с помощью специфицированной процедуры конструирования новых значений.

Каждое отдельное значение принадлежит только одному ТД, хотя оно может принадлежать и нескольким подтипам этого ТД.

Равенство. Для каждого пространства значений существует понятие равенства (*equality*) значений, задаваемого следующими аксиомами.

Аксиома 1. Для любых двух значений (*a, b*) из пространства значений выполняется условие равенства *b*, специфицированное как $a=b$, или неравенства *b*, специфицированное как $a \neq b$.

Аксиома 2. Не существует пары таких значений (*a, b*) из пространства значений, для которых одновременно выполняются условия $a = b$ и $a \neq b$.

Аксиома 3. Для значения *a* из пространства значений выполняется условие $a = a$.

Аксиома 4. Для любых двух элементов значений (*a*, *b*) из пространства значений $a = b$, тогда и только тогда, когда $b = a$.

Аксиома 5. Если для произвольных трех элементов значений (*a, b, c*) из пространства значений выполняются условия $a = b$ и $b = c$, то выполняется условие $a = c$.

Для каждого типа данных операция равенства *Equal* определяется как свойство равенства пространства значений. Для любых значений *a* и *b* из пространства значений *Equal* (*a*, *b*) есть *true*, если $a = b$, и *false* в противном случае.

Порядок. Пространство значений упорядочено, если для него установлено отношение порядка (*order*), которое задается знаком меньше или равно (≤) и удовлетворяет правилам:

1) для каждой пары значений (*a, b*) из пространства значений выполняется условие $a \le b$ или $b \le a$ или оба этих условия;

2) для любых двух значений (*a*, *b*), если $a \le b$ и $b \le a$, то $a = b$;

3) для любых трех значений (*a, b, c*), если $a \le b$ и $b \le c$, то $a \le c$.

Запись $a < b$ используется для нотации: $a \le b$.

Тип данных упорядочен, если отношение порядка определяется на его пространстве значений. Тогда операция InOrder определяется для произвольных двух значений *a* и *b* из пространства значений InOrder(*a*, *b*) есть true, если $b \le a$, и false в противном случае.

Ограниченность. ТД ограничен сверху, если он упорядочен и существует такое значение U из его пространства значений, при котором для всех значений *s* этого пространства выполняется условие $s \le U$. Значение *U* образует верхнюю границу пространства значений. Аналогично ТД ограничен снизу, если он упорядочен и существует такое значение *L* из его пространства значений, что для всех *s* этого пространства выполняется условие $L \le s$. Значение *L* образует нижнюю границу пространства значений. ТД называется ограничением, если его пространство значений имеет верхнюю и нижнюю границу.

Кардинальность. Пространство значений ТД основывается на математической концепции кардинальности (*cardinality*) и оно может быть конечным или бесконечным. ТД должен иметь кардинальность (мощность) своего пространства значений. В стандарте предусмотрены три важных категории ТД, пространство значений которых является:

1) конечнам;

2) точнам и бесконечнам;

3) приближеннам (пространство значений которой может быть конечным или бесконечным).

Точный и приближенный ТД. Если каждое значение в пространстве значений концептуального типа данных можно отличить от другого значения в пространстве, то ТД считается точным (*exact*).

Математические ТД, которые имеют значения и не имеют определенного представления, называются приближенными (*approximate*). Пусть *M* — математический ТД, а *C* — соответствующий вычисляемый ТД, *P* — преобразователь пространства значений *M* в пространство значений *C*. Тогда для каждого значения v' с *C* существует соответствующее значение типа данных *v* с *M* и такое действительное значение *h*, что $P(x) = v'$ для всех *x* с *M* и $|v - x| < h$. Таким образом, v' — это приближение *C* для всех значений *M*, находящееся в *h*-области значение v''. Кроме того, для одного значения v' в *C* существует более чем одно такое значение в *M*, что $P(y) = v'$. Таким образом, *C* не является точной моделью *M*.

Числовой ТД называется числовым (*numeric*), если концептуально его значения определяются количественно (в системе нумерации). ТД, значение которого не имеет этого свойства, называется нечисловым (*non-numeric*).

Пространство значений. Пространство значений — это совокупность (коллекция) значений ТД, которая определяется одним из следующих способов:

– исчислением;

– аксиоматическим определением согласно основным положениям;

– как подмножество уже определенного пространства значений, которое имеет тот же набор свойств;

– как комбинация любых значений некоторого, уже определенного пространства значений с помощью специфицированной процедуры конструирования новых значений.

Каждое отдельное значение в пространстве принадлежит только одному типу данных, хотя оно может принадлежать и нескольким подтипам этого типа данных.

Каждый концептуальный тип данных являются точным. Невычислимый ТД является бесконечным. Если каждое значение в пространстве значений концептуального типа данных можно отличить от другого значения в пространстве этой модели, то тип данных считается точным (exact)**.**

Сгенерированные типы данных

Сгенерированные ТД (*generated datatypes*) — это типы данных, полученные в результате применения генератора типов данных.

ТД, с которыми работает генератор, называются параметрическими или компонентными. Сгенерированный ТД семантически зависит от параметрических типов данных, но имеет собственные характеристические операции. Важной характеристикой всех генераторов ТД является то, что генератор может применяться до многих разных параметрических ТД. Генераторы указателя и процедуры генерируют ТД, значения которых атомарные, тогда как генератор Выбора и агрегатных типов данных генерирует ТД, значения которых позволяют производить их декомпозицию.

Генератор ТД (*datatype generator*) — это концептуальная операция над одним или несколькими ТД, которая создает новый ТД. Генератор типов данных оперирует с ТД, а не с его значениями и представляет собой:

1) набор критериев для характеристик ТД, над которыми будут выполнены операции;

2) процедуры конструирования, которые допускают набор ТД с данным критерием для создания нового пространства значений из пространств значений этих ТД;

3) набор характеристических операций, которые применяются в конечном пространстве значений для завершения определения нового типа данных.

Агрегатный ТД (*aggregate datatype*) — это сгенерированный ТД, каждое значение которого получено из значений параметрических ТД. Параметрические ТД агрегатного ТД или его генератор включают в себя имена компонентов ТД. Генератор агрегатного типа данных генерирует ТД с помощью алгоритмической процедуры пространства значений агрегатного ТД.

В отличие от других сгенерированных ТД агрегатный ТД обеспечивает доступ к компонентам значений через характеристические операции. Агрегатные значения разных типов различаются между собой свойствами, которые характеризуют отношение между компонентами ТД и отношение между каждым компонентом и агрегатным значением.

Генератор сложных типов данных в GPD

Сгенерированные типы данных (*generated datatypes*) — это ТД, полученные в результате применения генератора. **Генератор типов данных** — это концептуальная операция на одном или нескольких ТД, создает новый ТД. Генератор оперирует с ТД для создания нового ТД, а не его значений. ТД, с которыми работает генератор, называются параметрическими или компонентными. Сгенерированный ТД семантически зависит от параметрических ТД и имеет собственные характеристические операции. Генераторы для «Указателя» и «Процедуры» генерируют ТД, значения которых атомарные, тогда как генератор «Выбора» и агрегатных ТД генерирует ТД, значения которых можно декомпозировать.

Выбор генерирует ТД. Каждое значение образуется из любого набора альтернативных типов данных. Этот ТД логически учитывает их соответствие значению другого ТД с признаком (*tag*).

Указатель генерирует ТД, каждое значение которого устанавливает средства ссылки на значение другого типа данных, специфицированного типом данных *element-type*. Эти значения типа данных указателя являются атомарными.

Процедура генерирует ТД, каждое значение которого является значением других типов данных, которые называют параметром. Такой ТД включает в себя набор всех операций над значениями конкретной коллекции типов данных, концептуально атомарных.

Агрегатные типы данных

Генератор агрегатного типа генерирует ТД путем:

- применения алгоритмической процедуры к пространству значений его ТД для создания пространства значений агрегатного ТД;
- обеспечения набора характеристических операций, специфических для генератора.

Таким образом, многие свойства агрегатных ТД составляют свойства генератора, независимо от ТД компонентов. В отличие от других генераторов ТД, для агрегатных характерно то, что значение компонентов агрегатного значения получается с помощью характеристических операций.

Ниже приведены агрегатные ТД.

Запись генерирует ТД, значения которого составляют совокупность значений компонентов типов данных, и каждая совокупность имеет значение для каждого компонента типа данных, специфицированного фиксированным идентификатором поля *field-identifier.*

Набор генерирует ТД из пространства значений из поднаборов пространства значений типа данных элемент с операциями, свойственными математическому множеству *set.*

Портфель генерирует ТД, значения которого составляют коллекции образцов значений типа данных элемент. Многочисленные образцы того же значения могут подаваться в этой коллекции, а в каком порядке — несущественно.

Последовательность генерирует ТД, значениями которого являются упорядоченные последовательности значений типов данных из значений, несвойственных этому типу данных; одно и то же значение может встречаться многократно в этой последовательности.

Массив генерирует ТД, значения которого ассоциируются с произведением пространств одного или нескольких конечных типов данных, которые называются индексными ТД. Пространство значений этого типа данных таково, что каждому значению из пространства индексного типа данных соответствует только одно значение элемента.

Таблица генерирует ТД, значения которого составляют коллекции значений из пространства одного или нескольких типов данных поле, такое что каждое значение из пространства задает ассоциации между значениями его полей.

Объявленные типы данных

Объявленный тип данных (defined) — это тип данных, определенный с помощью описания типа type-declaration. **Type-identifier** — идентификатор некоторого объявленного типа, который ссылается на ТД или генератор ТД. **Actual-type parameters** соответствует номеру и типу formal-type parameters объявленых в type-declaration. Таким образом, каждый actual-type-parameter соответствует formal-type-parameter в соответствующей позиции списка formal-type-parameter-list. Если formal-parameter-type составляет type-specifier, то actual-type parameters будет value-expression, определять значение ТД, специфицированным как formal-parameter-type. Если он является “type”, то actual-type-parameter будет type-specifier и иметь нужные свойства этого параметрического ТД.

Type-declaration идентифицирует type-identifier в type-reference с одним типом данных, семейством типов данных или генератором типов данных. Если идентификатор типа type-identifier задает семью типов данных, то type-reference ссылается на тот член семьи, пространство значений которого определяется посредством type-definition после замены каждого значения actual-type-parameters для всех входов formal-parametric-value. Если type-identifier задает генератор типов данных, то type-reference означает ТД, который получается применениями генератора типов данных к реальным параметрическим типам данных.

Характеристические операции

Это операции, которые создают значение любого типа с помощью генератора ТД, создавая пространство значений параметрических ТД. Такие операции необходимы для

выделения ТД по их названиям и генерации агрегатных ТД как композиции следующих операций:

1) с нулевой арностью для генерации значений этого ТД;

2) с унарной операциею (арности 1), которая превращают значение этого ТД в новое значение этого же ТД или в значение *boolean*;

3) с арностью 2, которые преобразуют пары значений этого ТД в значение этого же ТД или в значение *boolean*;

4) с *n*-арностью, преобразующей упорядоченные n-элементные группы значений, каждая из которых относится к определенному ТД и может быть параметрическим типом или агрегатным.

Практически не существует уникальной коллекции характеристических операций для заданного ТД. Одна коллекция операций для ТД (или генератора типов) достаточна для выделения этого ТД среди других из пространства значений той же мощности.

Таким образом, существует посимвольная замена, которая преобразует все пространство значений одного ТД (*domain*) в подмножество значений пространства другого ТД (диапазон, *range*) так, чтобы значение отношений и характеристических операций домена сохранялись в соответствующих значениях отношений и характеристических операций диапазона ТД.

Преобразование ТД разнородных ресурсов

В мировых библиотеках Интернет существует большое количество разнородных программ для вычисления физических, биологических, математических и других задач. Данные могут быть представлены в виде пространственных зрительных образов, отчетов и наборов данных, генерируемых с различных датчиков или специализированной аппаратуры. Такие данные относятся к классу больших данных и при вычислениях требуют нестандартных методов и приемов для их анализа, обработки и организации вычислений.

Типы данных стандарта GPD ISO/IEC 11404

К общим типам данных (GPD - General Purpose Datetypes) стандарта ISO / IEC 11404-2007 относятся:

- независимые от языка типы данных, которые используются для формального описания концептуальных данных, их элементов и значений;

- полуструктурированные и неструктурированные совокупности данных, в которых ТД являются неизвестной или неопределенной заранее структурой данных;

- расширяемые общие типы данных.

Стандарт GPD устанавливает номенклатуру и семантику наборов типов данных, которые используются в языках программирования и в интерфейсах программных систем. В этом стандарте специфицированы базовые типы данных и сложные, которые полностью или частично определяются с помощью простых типов данных. Термин «независимые от языка типы данных» означает, что специфицированные типы данных образуют классы, представители которых в языках программирования (ЯП) соответствуют общей концепции типов данных стандарта GDT ISO/IEC 11404 и частично совпадают с фундаментальными типами данных (FDT) ЯП.

Примитивные типы данных стандарта GPD

Рациональный (rational) – математический тип данных, который соответствует действительным числам.

Масштабированный (scaled) – это семейство типов данных, пространством значений которого является подмножество рациональных чисел, а каждый отдельный тип данных имеет фиксированный знаменатель и предполагает аппроксимацию его значений.

Комплексный (complex) – это семейство типов данных, каждый из которых задает числовой математический тип данных для комплексных чисел.

Пустой (void) – это тип данных, который задает объект с необходимыми синтаксическими и семантическими описаниями и не несет никакой информации.

Значение типа данных (ТД) определяется путем:

1) перечисления;

2) аксиоматического определения;

3) подмножество пространства значений;

4) комбинация любых значений путем конструирования новых значений.

Сгенерированные типы данных стандарта GPD

Сгенерированный ТД – это ТД, полученные в результате генерации типов данных. Тип данных, с которым работает генератор, называется параметрическим или компонентным. Сгенерированный ТД семантически зависит от параметрических типов, но имеет собственные характеристические операции. Важной характеристикой всех генераторов ТД является то, что генератор может применяться к разным параметрическим ТД. Генераторы указателя и процедуры дают типы данных, значения которых атомарные, тогда как генератор выбора и агрегатных типов данных выдает типы данных, значения которых позволяют производить их декомпозицию.

Генератор ТД – это концептуальная операция над одним или несколькими типами данных, которая создает новый тип данных. Генератор оперирует с типами данных, а не с его значениями и представляет собой:

1) набор критериев для характеристик типов данных, над которыми будут выполнены операции;

2) процедуры конструирования, которые допускают набор типов данных с данным критерием для создания нового пространства значений из пространств значений этих типов данных;

3) набор характеристических операций, которые применяются в конечном пространстве значений для завершения определения нового ТД.

Агрегатный тип данных - это сгенерированный ТД, каждое значение которого получено из значений параметрических ТД. Параметрические ТД агрегатного ТД или его генератор включают в себя имена компонентов ТД. Генератор агрегатного ТД выдает ТД с помощью алгоритмической процедуры в пространстве его значений. Агрегатный ТД обеспечивает доступ к компонентам значений через характеристические операции. Агрегатные значения разных типов различаются между собой свойствами, которые задают отношение между компонентами ТД и между каждым компонентом и агрегатным значением.

Сложные типы данных стандарта GPD и генераторы ТД

Множество (set) задает тип данных, пространство значений которого составляет набор всех поднаборов пространства значений. Операции соответствуют математическому множеству set. Стандарт GPD включает генераторы ТД сложных типов данных: выбор (choice), указатель (pointer), процедура (procedure), запись (record), набор (set), портфель (bag), последовательность (sequence), массив (array), таблица (table) и т.п.

Характеристические операции создают значение любого типа с помощью генератора ТД в пространстве значений параметрических ТД. Практически не существует уникальной коллекции характеристических операций для заданных ТД. Одна коллекция операций ТД достаточна для выделения этого ТД среди других из пространства значений той же мощности.

Преобразование типов данных ISO/IEC 11404-96

Стандарт определяет LI–язык, который преобразует ТД независимо от ЯП, и включает следующие виды преобразований:

– внешнее преобразование внутренних ТД ЯП в LI–типы данных;

– внутреннее преобразование LI–типа данных в ТД ЯП;

– обратное внутреннее преобразование к внешнему.

Внешнее преобразования ТД состоит в следующем:

а) установки связи с LI–типом данных;

в) задания связи между допустимым значением и его эквивалентным значением из LI–ТД;

с) LI–типа данных определяется значением любого внутреннего типа данных.

Внутреннее преобразование задает связь примитивного ТД или сгенерированного в LI–тип данных с внутренним ТД ЯП.

Обратное внутреннее преобразование LI–типа данных состоит в преобразовании значений внутреннего ТД в соответствующее значение LI–типа при наличии соответствия и отсутствия двусмысленности. Это преобразование для ЯП является коллекцией обратных внутренних преобразований LI–типа данных.

К проблемам преобразования ТД в разных ЯП относятся:

а) несоответствие количества (формальных и фактических) параметров или неверное их описание;

б) несогласованность типов передаваемых параметров или их значений для форматов компьютеров;

е) отсутствие прямых и обратных преобразований араметров и др.

Так как FDT не охватывал все виды данных и типов новых ЯП после 1992г., то появился новый стандарт общих типов донных GPD, которые и использовались в информационных и прикладных системах. В стандарт GPD включены FDT и новые сложные типы данных — контейнеры, указатели, множества, списки, последовательности, неструктурные данные и т.п.

Данный стандарт ISO/IEC 11404 GPD — 1996 прошел многолетнюю апробацию и вышел новый вариант в 2007г. В его состав входят такие типы данных: агрегатные, генеративные, расширяемые и др., которые требуют генерации к фундаментальным ТД для последующего применения в ресурсах Глобальной сети Интернет. Требуется разработка новых примитивных функций преобразования неэквивалентных ТД для новых ЯП (C, C++, Python, Basic, Ruby, JAVA и др.) и других типов (рис.3).

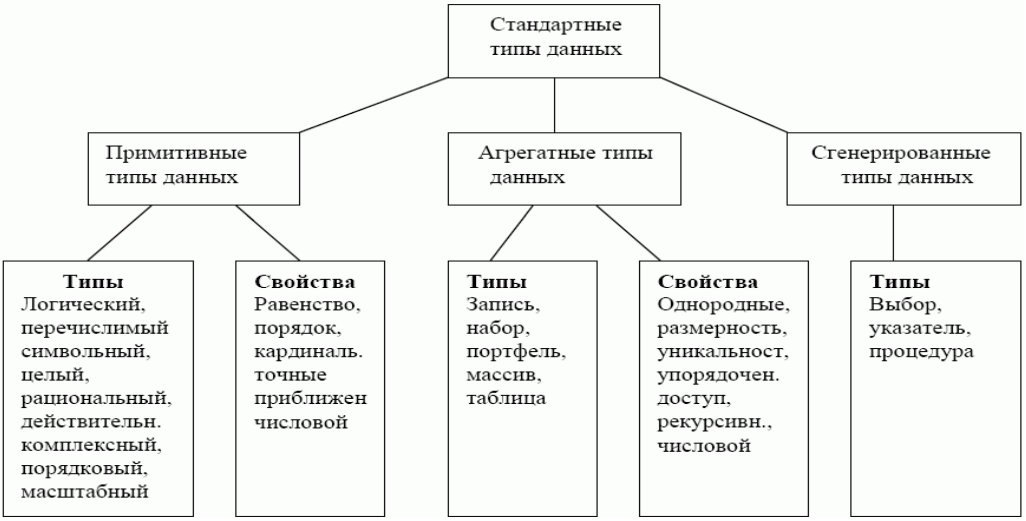


Рис. 3. Типы данных стандарта GPD

Для решения проблем преобразования данных согласно стандарта GPD и FDT при связывании (взаимодействии) КПИ и сервисов, заданных в разных современных ЯП, предложен подход к генерации GPD<=>FDT (рис.13), Основу этого подхода составляет задачи преобразования типов данных в сборщике ресурсов с использованием ранее созданных для FDT .

Согласно приведенной схемы (рис.4) реализуются следующие задачи преобразования данных для поддержки сборщика ресурсов:

– специфицирование внешних ТД в WSDL, сохранения их в БД и в репозиториях;

– преобразование новых ТД в новых $ЯП_1$, …, $ЯП_n$;

– реализацию ТД GPD к виду специальных примитивных функций FDT;

– представление ТД FDT к виду специальных примитивных функций и формату Фреймворка;

– эквивалентные отображения и генерацию данных GPD<=>FDT с учетом платформ современных компьютерных и кластерных систем Интернет где ресурсы будут работать

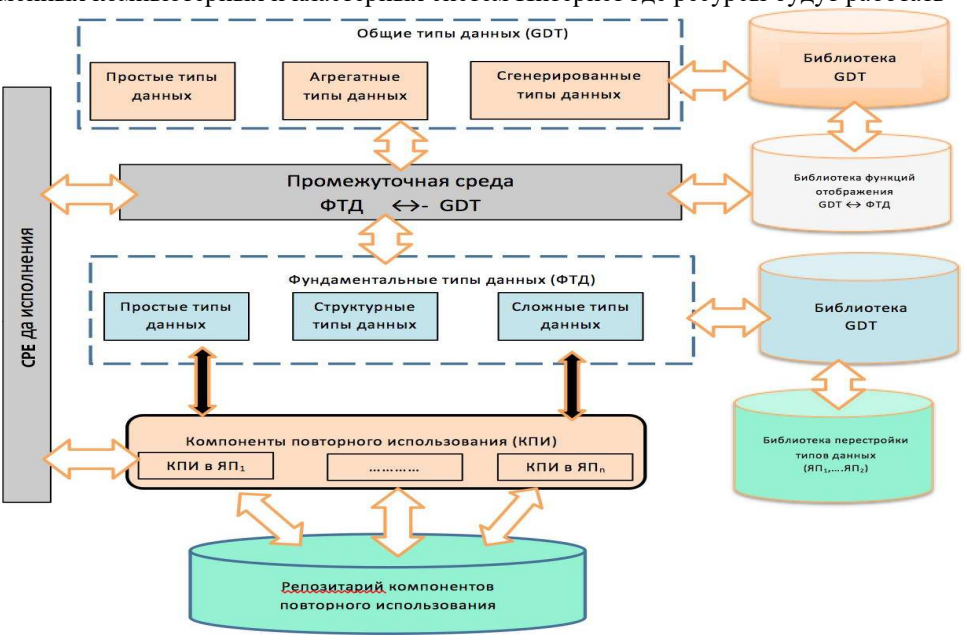


Рис. 4. Схема преобразования ТД стандарта GPD

Полуструктурированные, неструктурированные и расширяемые ТД используются в информационной среде, в связи с исследованиями недр земли, космоса и океана. Потребуется практическая обработка этих новых структур данных, чтобы с ними решались эффективно прикладные задачи в разных прикладных областях, особенно работающих с Big Data.

5.12.4. Общие теоретические задачи преобразования ТД стандарта GPD

В разработанной схеме рис.3 дана схема генерации ТД GPD в структуры фундаментальных ТД ЯП. Предлагается разработать библиотеку функций генерации ТД стандарта GPD, элементы которой для всех новых ТД этого стандарта выполняют следующие виды операций:

– преобразование ТД, содержащихся в ЯП ($ЯП_1$, …, $ЯП_n$) и которые входят в состав ФДТ ТД ЯП;

– функции трансформации сложных ТД стандарта GPD, включенных в стандартную библиотеку CTS VS.Net и используются трансляторами с ЯП этой системы для преобразования сложных ТД к более простым;

– операции взаимодействия компонентов повторного использования, записанных в разных ЯП, интерфейс которых задается в языке IDL.

Все ТД стандарта GPD представлены в виде следующих классов алгебраических систем:

$\sum 1 = \{G\alpha^{b}, G\alpha^{c}, G\alpha^{u}, G\alpha^{r}\}$,

$\sum 2 = \{G\alpha^{a}, G\alpha^{z}, G\alpha^{u}, G\alpha^{e}\}$,

$\sum 3 = \{G^{s}, G^{ns}\}$, где

$G\alpha^{t} = <X\alpha^{t}, \Omega\alpha^{t}>$, t – ТД языка L, $X\alpha^{t}$ – множество значений ТД;

$\Omega\alpha^{t}$ – множество операций над t -ТД ;

$\sum 1$ - простые ТД: $t = b$ (bool), c (char), i (int), r (real)),

$\sum 2$ – сложные (структурные) типы данных: t = a (array), z (record), u (union), e (enum).), как комбинации простых ТД;

$\sum 3$ - сложные, неструктурированные ТД (портфель, контейнер, протокол и т.п.) и сгенерированные из них более простые данные.

В каждом классе этих систем преобразование $t \rightarrow Tq$ для пары языков lt и lq основано на таких свойствах отображений:

1) системы $G\alpha^{t}$ и $G\beta^{q}$ – изоморфны, если их q, t определены на одном и том же множестве ТД;

2) между значениями $X\alpha^{t}$ и $X\beta^{q}$ типов данных t, q существует изоморфизм, если множество операций $\Omega\alpha^{t}$ и $\Omega\beta^{q}$ разные;

3) если множество $\Omega = \Omega\alpha^{t} \cap \Omega\beta^{q}$ не пустое, то имеет место изоморфизм двух систем $G\alpha^{t'} = < X\alpha^{t'}, \Omega>$ и $G\beta^{q'} = < X\beta^{q}, \Omega>$.

4) если типы данных отличаются, например, t - строка, а тип q – вещественное, то между множествами $X\alpha^{t}$ и $X\beta^{q}$ не существует изоморфного соответствия.

Отображения сохраняют линейный порядок элементов к виду линейной упорядоченности элементов алгебраических систем из этих классов.

Неструктурированные данные больших данных GPD

В результате проведенных нами исследований неструктурированных данных из класса больших данных (Big Data), рассмотрена применимость к ним стандартных ТД ЯП и GPD.

Неструктурированные данные GPD – это совокупность данных, которые:

1. Структурированные ТД или метод доступа;

2. Наполовину структурированные данные, которые имеют один ТД или метод доступа;

3. Неструктурированные данные, включающие набор данных неодинаковой природы.

Приложения, основанные на работе со структурными ТД, включающими реляционные и нереляционные данные, которые используют одну из трех архитектур:

– реляционные данные находятся в БД, а большие нереляционные данные двоичных объектов (BLOB), которые находятся в файловых системах или на файловых серверах;

– нереляционные данные из хранилищ, предназначенных для BLOB-данных;

– реляционные и нереляционные данные, которые находятся в БД.

Эти данные используются разными приложениями путем:

– создания, загрузки, обновления и удаления неструктурированных данных и использования транзакционной согласованности между источниками неструктурированных данных;

– индексирования неструктурированных данных и их поиска;

– извлечения метаданных в явной форме и предоставление их пользователям;

– анализа и преобразования содержимого документов к форматам для выполнения поиска и составления запросов (например, преобразование звуковых файлов в текстовые и выполнение поиска по запросу БД).

Хранение неструктурированных данных в хранилищах проводится с помощью BLOB-данных. При хранении BLOB-данных в БД централизованного хранилища снижаются затраты и быстродействие.

Анализ нестандартных данных из наборов больших данных (Big Data)

К методам анализа данных относятся статистические методы (дескриптивный анализ, корреляционный и регрессионный анализ, компонентный анализ и др.), а также методы синтаксического анализа раздела описания сложных данных GPD.

Регрессионный и компонентный метод математически ориентированы на оценку необходимой величины экспертом и сравнения ее с другими величинами.

Описание ТД представляется в виде таблицы СМ метода терминальных символов и семантических программ их реализации. Каждому представлению терминальных символов соответствует операционный знак его обработки (+, - , / и др.).

Функции их реализации могут повторяться и для других ТД.

Для проведения анализа ТД предлагается создать таблицу ТД и набор функций их реализации в языке XML.

Таблица функций включает набор неструктурированных ТД, которым прикреплены функции трансформации таких ТД к имеющимся в первой таблице. Результатом обработки этой таблицы является разложение неструктурных данных в виду простых данных в том порядке, в котором они заданы в исходной таблице.

Подходы к обработке неструктурированных данных Big Data

Big Data — набор данных большого объема, а также подходов, средств, инструментов и методов представления неструктурированных огромных объёмов данных для получения данных и эффективного использования их на многочисленных узлах сети Интернет при решении прикладных Intelligence ИС и ИТ систем с использованием СУБД.

Для работы с большими объемами данных сформировался метод ETL (Extract Transform Load), с помощью которого производится:

- извлечение данных из внешних источников;
- трансформация и очистка данных с учетом требований;
- загрузка данных в хранилища данных;
- анализ данных и их перенос из одного приложения в другое и др.

К основным свойствам Big Data относятся**:**

- *горизонтальная масштабируемость* обрабатываемых больших данных и большого количества кластеров и серверов;
- *отказоустойчивость* по отношению к сбоям на процессорах кластеров и в узлах сети. Так, инструмент Hadoop-кластер Yahoo имеет более 42000 компьютеров, среди которых часть из них может выходить из строя;
- *локализация данных и о*бработка их на серверах, где практически хранятся большие данные для решения соответствующих задач;
- *изменение числа работающих* на кластере с помощью средств MySQL Cluster. Большие данные могут быть представлены как tensors, которые управляют вычислением (например, полилинейное обучение подпространств Multilinear Subspace Learning и др.).

К системным средствам обработки Big Data относятся:

NoSQL-БД нереляционные и распределенные данные с открытым кодом и горизонтальной масштабируемостью, эффективно поддерживают случайное чтение, запись и версионность.

MapReduce — модель распределённых вычислений, которая используется при параллельных вычислениях с большими данными в компьютерных кластерах.

Hadoop — свободно распространяемый набор утилит, библиотек и фреймворков для

разработки и выполнения распределённых приложений (в том числе, MapReduce-программ), работающих на кластерах с сотнями и тысячами узлов.

CMB— системы обработки больших данных в рамках Cloud-вычислений.

Big Data анализируются средствами**:** статистических и динамических методов анализа искусственного интеллекта, нейронных сетей, математической лингвистики; A/B Testing, Crowdsourcing Data Fusion; Integration Genetic Algorithms Machine Learning; Natural Language Processing; Signal Processing Simulation and Visualization; Massively Parallel Processing; Search-Based Applications, Data Mining, Multilinear Subspace Learning и др.

5.13.5. Задачи общего сборщика ресурсов систем и индустрии фабрик программ

В качестве общего вывода по рассмотрению операций сборки для разных сред (3.1-3.7) можно сделать вывод о том, что приведенные операции сборки (link, make, config, assembling, building, weaver, integration) ресурсов в системных средах, являются базовыми средствами для создания общего «сборщика» готовых ресурсов для прикладных, сервисных, информационных и технических систем в Глобальной сети Интернет.

Приведенные операции сборки (link, make, config, assembling, weaver) ресурсов в заданных системных средах в п.3.1-3.8 имеют схожие операционные способы сборки разных вариантов ресурсов: модулей в разных ЯП, исходных и выходных текстов компиляторных программ, сервисных и системных ресурсов Веб среды Интернет, технических средств компьютеров имеют похожие способы сборки, которые повторяются в разных общесистемных средах. Как бы не назывались способы сборки семантика сборки остается одинаковой. Способ сборки зависит от данных, которые используются при обмене данными между связываемыми разнородными ресурсами.

Рассмотренные средства генерации общих типов данных стандарта ISO/IEC 11404 GPD – 2007 требуют создания набора новых примитивных функций для нестандартных типов данных (портфелей, контейнеров, шаблонов, указателей, неструктурных данных и др.) и использоваться в названных способах сборки разнородных ресурсов Интернет.

Для создания общего «сборщика» готовых ресурсов в прикладные и технические (макроконвейерные) системы в Интернет, необходимо на базе конфигурационной сборки сделать следующее:

1. Определить формальный вариант задания операции сборки ресурсов
2. Создать библиотеку примитивных функций для всех систем Интернет и дорабатывать примитивные функции для новые ТД (контейнеров, шаблонов и др.) согласно стандарта GPD.
3. Создать анализатор (агент) операции сборщика и взаимодействия (компиляторных, системных, прикладных, технических) с учетом всех типов ресурсов.
4. Проводить анализ соответствия типа ресурса и взаимодействия с другим ресурсом через интерфейс и осуществлять обращение к соответствующим программам преобразования обмениваемых данных для генерации соответствующего преобразования по теории преобразования типов данных стандарта GPD.
5. Управлять конфигурационной сборкой при выполнии всех процессов, определенных в стандарте IEEE 828-2006 Configuration.
6. Выдавать сведения о результатах сборки и завершения работы сборщика.
7. Обеспечить размещение ресурсов в библиотеках и хранилищах Интернет из разных предметных областей знаний (медицины, биологии, генетики, математики и др.).

Техническим результатом общего сборщика является:

1) простота поиска ресурсов в хранилищах Интернет и снижение затрат на разработку за счет готовых правильных многоразовых ресурсов и настройку их на конкретные условия применения;

2) повышение качества и производительности создаваемых из ресурсов Веб-приложений и систем за счет системных операций замены отдельных более правильных ресурсов и изменения конфигурации (архитектуры) варианта конфигурационного файла с получением качественных решений функциональных задач приложений в разных предметных областях знаний.

Конфигурация сервисных ресурсов Web-систем

Согласно стандарта IEEE 828-2012 Configuration Management (см.п.3.7) проводится конфигурирование сервисных ресурсов, которое включает задание следующих необходимых данных:

- базис конфигурации — BC (Configuration Baseline);
- элементы конфигурации (Configuration Item);
- компоненты, ГОР, входящие в описание моделей M_{sys}, M_{wsys};

Управление конфигурацией (Configuration Management) заключается в наблюдении за модификацией параметров конфигурации и компонентов системы, а также в проведении систематического контроля, учета и аудита внесенных изменений, целостности и работоспособности системы на процессах:

1. Идентификация конфигурации (Configuration Identification).
2. Контроль конфигурации (Configuration Control).
3. Учет статуса конфигурации (Configuration Status Accounting).
4. Аудит конфигурации (Configuration Audit).
5. Трассировка конфигурации при сопровождении и эксплуатации системы;
6. Верификация компонентных сервисов по моделям систем и веб-систем.

При конфигурационной сборке ГОР используется модели систем и модель характеристик MF (Model Feature). КПИ хранятся в репозиториях или библиотеках системы. Они выбираются их библиотек, адаптируются и интегрируются в систему

Основную роль в этих процессах выполняет конфигуратор (рис.5), построенный аспирантом Колесником А. и сохранен на сайте http:///7dragons.ru/ru.

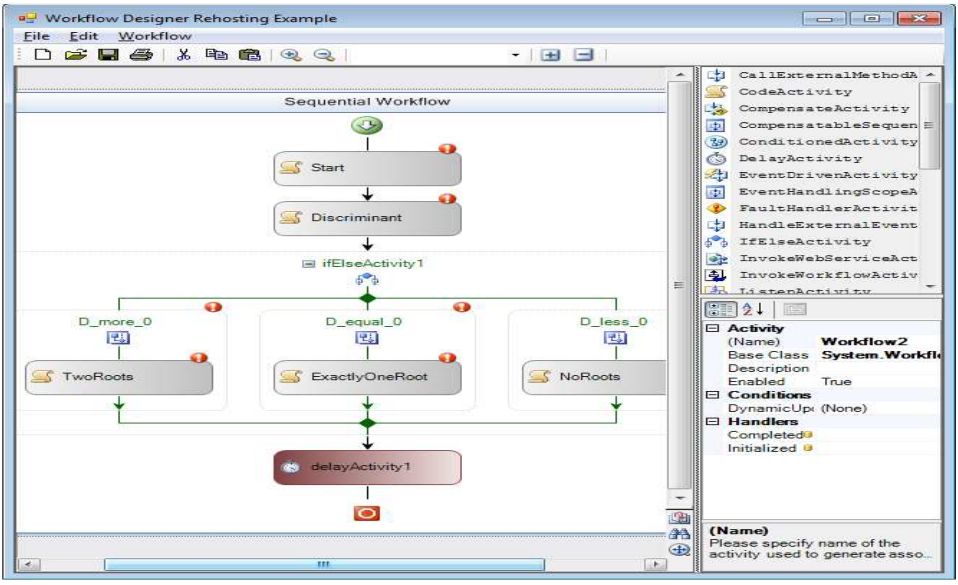


Рис.5. Модель конфигуратора

Данный *конфигуратор* включает:

– графовую структуру системы из ресурсов;
– модель вариантов системы;
– конфигурационную модель и операции сборки КПИ;
– перацию проверки - аудит конфигурации системы;
– верификацию моделей и ресурсов, используемых в Веб-системе;
– оценку качества КПИ и секонфигурированной Веб-свистем.

Конфигуратор, используя операции сборки config, собирает Веб-систему с учетом моделей M_{sys}, M_{MF} и заданных ресурсов (КПИ, ГОР, Reuses) предметной области и их интерфейсов, а также и сервисных служб Интернет. Конфигуратор проводит оценивание заданных ресурсов и сконфигурированного файла Веб-системы на надежность, защиту данных и безопасность их функционирования.

Обеспечение качества Веб-систем

Стандарт ISO/IEC 12207 ЖЦ ПО- 2009, 2012 регламентирует процессы планирования, управления качеством и оценки затрат на создание системы. На этих процессах ЖЦ проводится анализ достижения качества; верификация и валидация (V&V) ресурсов и оценивание степени достижения отдельных показателей качества; тестирование готовой системы; сбор данных об отказах, дефектах и др. ошибках; оценивание надежности по соответствующим моделям надежности с учетом результатов тестирования.

Модель качества стандарта ISO/IEC 9000 (1-4) Quality задает шесть показателей (характеристик) q_1 — q_6 (q — quality) качества:

q_1 — функциональность (functionality),
q_2 — надежность (reliability),
q_3 — удобство (usability),
q_4 — эффективность (efficiency),
q_5 — сопровождаемость (maintainability),
q_6 — переносимость (portability).

Каждая характеристика q_i рассчитывается по специальным формулам и метрикам качества. к. Надежность (reliability) оценивается согласно полученных на процессе тестирования ошибок, дефектов и отказов в ПО по моделям надежности (оценочным, измерительным и др.).

Все показателям качества q_1 — q_6 оцениваются по формуле вида:

$$q_1 = \sum_{j=1}^{6} a_{1j} m_{1j} w_{1j}$$

где a_i — атрибуты каждого показателя качества (i = 1..6);
m_i — метрики каждого атрибута качества;
w_i — вес каждого атрибута показателя качества системы.

Полученные значения по показателям модели качества входят в сертификат качества изготовленного продукта. Варианты оценки показателей качества сконфигурированных систем приведены на сайте ИТК.

5.13.6. Заключение по технологии сборки ресурсов в Интернет

Реализация технологии сборки Web-систем из интеллектуальных и сервисных ресурсов проводилась в рамках проекта РФФИ №16-01-00352 «Теория и методы разработки изменяемых программных систем» основывается на операциях сборки (Link, Make, Assemling, Config и др.) и стандарной конфигурационной сборке Веб-систем из готовых ресурсов — КПИ, reuses, service-components, Web-services, которые описываются в современных ЯП (C, C++, JAVA, Python, Ruby, Basic, Smalltalk и др.), а их интерфейсы в языке WSDL. Приведены операции конфигурационной сборки систем из готовых сетевых

ресурсов. Дан анализ подходов к определению интеллектуальных элементов в среде Семантик Веб и открытых моделей SOA и SCA для представления этих элементов с помощью системных сервисов, серверов, в среде клиент-серверной обработки данных Big Data в Cloud Computing. Отработана сборка модульных элементов с помощью операций (link, make, cmake, config, weaver) на примере действующих систем (АПРОП, CORBA, BSD, GNU, MSBuild; EuroGrid; Semantic Web; weaver BEA WebLogic Oracle, SAP Net и др.). Приведен теоретический аппарат сборки ресурсов с использованием фундаментальных и общих типов данных и механизмов преобразования, передаваемых через операции сборки неэквивалентных триваются данные (включая Big Data) в среде Интернет на основе стандарта ISO/IEEE 11404 GPD. Описана модель оценки качества по стандарту ISO/IEC 9000 (1-4) Quality, основанного на математических методах оценки надежности и рассмотренных ранее в разделе 2 данной монографии по - проекту РФФИ 16-01-00209 (2019–2021), а также обеспечение функциональной безопасности, защиты и оценки показателей качества.

Раздел 6. Форма 506. Проект РФФИ 19-01-00206 (2021). Методы и средства обеспечения надежности, безопасности и защиты технических, программных и операционных систем

Лаврищева Е.М., Мутиин В.С., Зеленов С.М. Новиков Е.М.

Основными задачами научных исследований по проекту №19-01-00206-2021 были:

1). Отработка теоретического подхода к моделированию и программированию информационных, программно-технических и операционных систем (ПТС) из соответствующих ресурсов предметных областей знаний (медицина, биология, математика, физика, химия) методом сборки ПТС и создание экспериментального ядра OS Linux средствами стандарта IEEE 828: Configuration (1996, 2007 (make, cmake, config, assembly, waver) с обеспечением надежности, безопасности и защиты данных создаваемых ОС, электронных средств и компьютерных систем из функциональных, информационных и сервисных ресурсов Интернет.

2). Развитие средств обеспечения современной гарантоспособной надежности, безопасности и защиты технических, программных и операционных систем, создаваемых методом моделирования и программирования информационных, интеллектуальных и сервисных ресурсов Интернет, накапливаемых в Библиотеках и Хранилищах данных Интернет и создаваемых конфигурационных структур операциями сборки стандарта IEEE 828: Configuration 1996, 2007, 2012: make, cmake, config, assembly, build, waver. Ресурсы, объекты и данные ОС и ПТС верифицируются на уровне ЯП и тестируются после сборки вариантов ПТС и ОС с обработкой возникших исключительных и аварийных ситуаций. После теоретического и практического опробования стандартных операций сборки предложен сборщик информационных, интеллектуальных и сервисных ресурсов Интернет операциями сборки: make, cmake, config, assembly, build, waver для предметных областей знаний (медицина, физика, биология и др.).

Methods and means of ensuring reliability, security and protection of technical, Software and operating systems

The main tasks of the scientific research project No. 19-01-00206-2021 was:

Testing a theoretical approach to modeling and programming information, software, hardware, and operating systems (PTS) from appropriate resources subject areas of knowledge (medicine, biology, mathematics, physics, chemistry) a method configuration Assembly PTS and creation of experimental kernel Linux OS means IEEE 828: Configuration (1996, 2007) make, cmake config, assembly, waver ensuring reliability, safety and protection of the data generated by the OS, electronic and computer systems from the functional, information and service resources on the Internet. 2). Development of means to ensure modern dependable reliability, security and protection of technical, software and operating systems created by modeling and programming of information, intellectual and service resources of the Internet, accumulated in Libraries and Data Repositories of the Internet and created configuration structures by assembly operations of the IEEE 828 standard: Configuration 1996, 2007, 2012: make, cmake, config, assembly, build, waver. The resources, objects and data of the OS and the PTS are verified at the level of the YAP and tested after assembling the variants of the PTS and the OS with the processing of the exceptional and

emergency situations that have arisen. After theoretical and practical testing of standard assembly operations, an assembler of information, intellectual and service resources with Internet assembly operations is proposed: make, smake, config, assembly, build, waver for subject areas of knowledge (medicine, physics, biology, etc.).

Основные задачи и цели заключительных этапов проекта РФФИ 206 в 2021.

1. Развитие моделирования и программирования вариабельных ПТС и операционных систем

Рассматривается технология создания вариабельных ПТС и ОС из информационных, интеллектуальных и сервисных ресурсов Интернет и проверку правильности используемых ресурсов в ПТС путем верификации исходного кода отдельных ресурсов, тестирования и оценивания качества продукта в условиях безопасности, гарантированной надежности и защиты данных в Базах данных. Основные публикации по этому направлению:

– Лаврищева Е.М., Рыжов А.В. Подход к созданию систем и сайтов из готовых ресурсов. - Научный сервис в сети Интернет: труды XX Всесоюзной научной конференции (17-22 сентября 2018.-Новороссийск)ю-М.;ИПМ им. Келдыша, 2018.ISBN 978-5-98354-046-0.-с321-246.

– Лаврищева Е.М., Петренко А.К. Технология сборки интеллектуальных и информационныхресурсов Интернет // Научный сервис в сети Интернет: труды XXI Всероссийской научной конференции (23-28 сентября 2019 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2019. — С. 469-488. — URL: http://keldysh.ru/abrau/2019/theses/93.pdf. doi:10.20948/abrau-2019-93

– E.M. Lavrischeva, A.K. Petrenko and B.A.Pozin Technology of Assembly of Intellectual

– Лаврищева Е.М. Теория графового моделирования сложных систем из модулей для прикладных областей.- Научно-практический журнал, Высшая школа. №14/ 2019.-с.27-46: www.ran-nauka.ru © ООО «Инфинити», 2019. ISSN 2409-1677.

– Лаврищева Е.М. Теория графового моделирования сложных систем из модульных элементов для прикладных областей знаний.- Austria-science»1 часть №28/2019.- p.12-30. http://austria-science.info.

– Е.М.Лаврищева, И.Б.Петров. Теория моделирования технических и математических задач предметных областей знаний. - Труды ИСП РАН 2020, Том 32 № 6.-с.175-199;

– Лаврищева Е.М., Петров И.Б., Петренко А.К. Парадигмы моделирования и программирования задач и данных предметных областей знаний. Direct Medio. Москва.-Берлин, 2021.- 495с.

2. Отработка стандартного метода конфигурационной сборки IEEE 828: Configuration

Для объединения требуемых ресурсов в общую структуру системы. Приводятся варианты операций сборки в рамках современных систем BSD, BUS. JAVAE, VS.Net, IBMSphere, Grid и др. и предложен общий сборщик ресурсов Интернет.Основные публикации:

- С.В. Козин. Конфигурационная сборка варианта ядра Linux для прикладных систем. Труды ИСПРАН-2018.- 99-110.

- Е.М. Лаврищева, А.К.Петренко. Технология сборки интеллектуальных и информационных ресурсов Интернет // Научный сервис в сети Интернет: труды XXI Всероссийской научной конференции (23-28 сентября 2019 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2019. — С. 469-488. — URL: http://keldysh.ru/abrau/2019/theses/93.pdf doi:10.20948/abrau-2019-93

- Технология сборочного создания экспериментального варианта ядра OS Linux с обеспечением качества для применения в прикладных и технических системах/ Е.М.

Лаврищева, А.К.Петренко, С.В.Зеленов, С.В.Козин//Конференция ИСП РАН -2021. Труды ИСП РАН.-2021.

- E.M. Lavrischeva, A.K. Petrenko and B.A.Pozin Technology of Assembly of Intellectual and Information Resources Internet.- CEUR-WS.org/2019.-vol-2543/paper-22.pdf.-27p.

- Е.М. Лаврищева, А.К.Петренко, С.В.Зеленов, С.В.Козин/ Труды ИСП РАН-2021.- с.14.

- Е.М.Лаврищева, И.Б.Петров. Теория моделирования технических и математических задач предметных областей знаний. - -Евроазиатское Научное Объединение. 2021.№ 1- (1).- с.35-43.

3. Развитие средств обеспечения гарантоспособной надежности, безопасности и защиты технических, программных и операционных систем

Ресурсы информационные, интеллектуальные и сервисные накапливаются в Библиотеках и Хранилищах данных Интернет и после проведения конфигурационной сборки* - make, cmake, config, assembly, build для создания ПС, СПС и фабрики программ.

В рамках проекта для управления предметными областями знаний создан вариант ядра OS Linux, котрый проверялся на безопасность, защиту и гарантированную надежность согласно современных международных и отечественных стандартов по анализу уязвимости ресурсов и систем в среде общесистемных средств Интернет (VS.MS, BSD, BSU, JAVAE и др.) для борьбы с кибератаками, нарушением целостности и конфиденциальности данных ПО ОС и прикладных систем с оценкой показателей надежности (отказоустойчивость, безопасность…) и качества версии систем и варианта ядра OS Linux.

*С.В. Козин. Конфигурационная сборка варианта ядра Linux для прикладных систем. Труды ИСПАН-2018.

Результаты по моделирования и программирования качественных ПТС и ОС

Теории моделирования и программирования ПТС, операционных и информационных систем развита в направлении создания вариабельных, вариантных систем в рамках Международного научного направления VAMOS, Reusebility и др. (2001-2017). Создана технология сборки таких систем с использованием стандарта IEEE 828 Configuration-2012 с участием членов проекта РФФИ. Был сделан доклад на конференции OS DAY 14-15октября -2021 по технологии сборки экспериментального варианта ядра OS Linux. Доклад вызвал большой интерес присутствующих на конференции [1]. Сделанная версия ядра OS Linux проверялась на надежность и безопасность с анализом уязвимостей ресурсов и систем в среде общесистемных средств Интернет (VS.MS, CORBA, BCD, BSU, JAVAEE, GRID, ETICS и др.) к кибератакам, нарушению целостности и конфиденциальности данных ПО ОС и прикладных систем с оценкой надежности и качества версии систем и Legasy Systems Отдельные аспекты средств моделирования и программирования вариантных ПТС и ОС с обеспечением надежности, качества и безопасности ресурсов и систем представлены в работах в работах. По методу сборки с обеспечением качества систем были сделаны доклады и публикации:

1. E.M.Lavrischeva, A.K. Petrenko and B.A.Pozin Technology of Assembly of Intellectual and Information Resources Internet.- CEUR-WS.org/2019.-vol-2543/paper-22.pdf.-27p.

2. Е.М. Лаврищева, А.К.Петренко, С.В.Зеленов, С.В.Козин. Технология сборочного создания экспериментального варианта ядра OS Linux с обеспечением качества для применения в прикладных и технических системах.- OS DAY-15октября 2021.

3. Лаврищева Е.М., Пакулин Н.В., Рыжов А.Г., Зеленов С.В. Анализ методов оценки надежности оборудования и систем. Практика применения методов. Труды ИСП РАН, том5. DOI:10.15514/ISPRAS-2018-30(3), 2018. 10.15514/ISPRAS-2018-30(3), http://0x1.tv/20180517F 9(in rus).

4. Зеленов С.В., Лаврищева Е.М. ИСП РАН -2019: систем. Труды ИСП РАН 2020, Том 32 № 5. С.151-163.

5. Е.М.Лаврищева, И.Б.Петров. Теория моделирования технических и математических задач предметных областей знаний. - Труды ИСП РАН 2020, Том 32 № 6. -Евроазиатское Научное Объединение. 2021.№ 1- (1).-с.35-43.

6. Лаврищева Е.М., Петров И.Б., Петренко А.К. Парадигмы моделирования и программирования задач и данных предметных областей знаний. DirectMeduio. Москва-Берлин, 2021.- 495с.

Основные положения технологии моделирования и программирования сложных вариабельных систем

Основу технологии моделирования ПТС и ОС составляет метод сборки ресурсов через интерфейс связи разнородных компонентов повторного использования (КПИ) с возможным преобразованием типов несовпадающих типов данных. Интерфейс между КПИ включает набор формальных средств организации обмена данными между модульными элементами Метод сборки основан на математических формализмах спецификации связей (по данным и по управлению) между разнородными модульными элементами и генерации интерфейсных модулей-посредников для каждой пары связанных элементов. Связывание КПИ задается операторами вызова в ЯП CALL/RPC/RMI с набором фактических и формальных параметров. Разработан стандарт конфигурационной сборки ресурсов Интернет IEEE 828 (configuration): 2012. Стандарт обеспечивает создание конфигурационной структуры системы по моделям и характериситмкам систем.

Технология сборки — это метод, средства и инструменты сборки информационных, программных, операционных ресурсов с помощью операций сборки:

link разнородных модулей систем CORBA, АПРОП;

make, cmake компиляторных программ системы BSD, GNU, MSBuild;

assemble, config. build системных, сервисных и вычислительных ресурсов EuroGrid,

Etics; integration *config* сервисных SOA, SCA в Semantic Web;

weaver обработка данных BEA WebLogic Oracle, SAP .Net.

Способы сборки такими операциями ресурсов опубликованы в работах:

1. Лаврищева Е.М., Петренко А.К. Технология сборки интеллектуальных и информационных ресурсов Интернет // Научный сервис в сети Интернет: труды XXI Всероссийской научной конференции (23-28 сентября 2019 г., г. Новороссийск). — М.: ИПМ им. М.В. Келдыша, 2019. — С. 469-488. — URL: http://keldysh.ru/abrau/2019/theses/93.pdf doi:10.20948/abrau-2019-93.

2. E.M. Lavrischeva, A.K. Petrenko and B.A. Pozin. Technology of Assembly of Intellectual and Information Resources Internet.- CEUR-WS.org/2019.-vol-2543/paper-22.pdf.-27p.

3. Лаврищева Е.М., Петров И.Б., Петренко А.К. Парадигмы моделирования и программирования задач и данных предметных областей знаний,.DirectMedio Москва, Берлин, 2021.- 495с.

6.1. Проведение конфигурационной сборки в Grid и Etics в рамках E-Science

Сборки компиляторных программ в ЯП (C++, Fortran, Go, Perl, PHP, Pyrhon, Java) с помощью операций make, cmake в системах BSD, GNU Unix и MS по интерфейсному посреднику в API (Application Programming Interface) исполняемого файла в машинном коде (из библиотек модулей Microsoft, Unix и др.), а также интеграцию интеграции

скомпилированных модулей с интерфейсом ABI (Application Binary Interface) в промежуточный язык MSIL байт-кода в среде .NET (*рис. 1*).

Процесс сборки ресурсов в ЯП с интерфейсом в API и ABI через make выполняет генератор сборочных скриптов GNU Build system в средах MSBuild (.NET), Apache Ant (Java), Apache Maven трансляторов с Java, C#, Scala; Scons (C, C++, Java, Fortran, Tex) и др.

ABI Interface содержит:

– набор инструкций процессора к регистровому файлу, стеку и памяти;

– размер и расположение базовых ТД, с которыми работает процессор компьютера;

– бинарные форматы файлов, библиотек и исполняемых файлов.

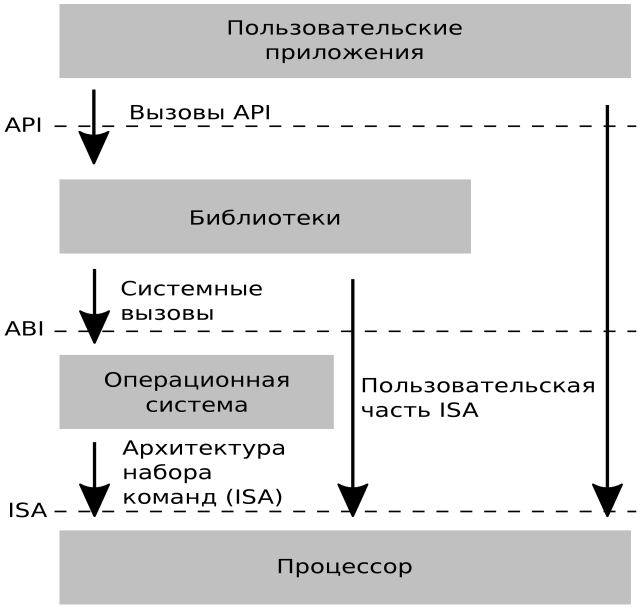


Рис. 1. Обработка приложений с интерфейсом ABI и API

Операции связи программ в API и ABI для C++ и Fortran (*рис. 2*):

– add executable (exec_name source1 source2 …) # создать исполняемый файл из файлов исходного кода source1, source2, и т.д.;

– add library (lib_name source1 source2 …) # создать библиотеку lib_name из файлов исходного кода source1, source2, и т. д.

– target include_directories (target PUBLIC dir1 dir2 …) # включить директории dir1, dir2, … в список поиска заголовочных файлов при сборке исполняемого файла из библиотеки;

– target_link_libraries(target lib1 lib2 …) # прилинковать (включить в сборку) элементы библиотек lib1, lib2 для target.

main.cpp

```
#include <iostream>

extern "C" void fort(void);
extern "C" void cube(float* x, float* y);

int main() {
  fort();
  std::cout << "C++\n";
  float x = 2.5, y = 0;
  cube(&x, &y);
  std::cout << x << ' ' << y << '\n';
}
```

fort.f90

```
subroutine fort() bind(C)
  implicit none
  write(*,*) "Fortran"
  return
end

subroutine cube(x,y) bind(C)
  implicit none
  real*4 x, y
  y = x * x * x
  return
end
```

Запуск

```
gfortran -c fort.f90 -o fort.o
g++ main.cpp fort.o –lgfortran
./a.out
```

Вывод

```
 Fortran
C++
2.5 15.625
```

Рис. 2. Пример сборки модулей в C++ и Fortran

Конфигурационная сборка ресурсов в системах GRID и ETICS.

Grid обеспечивает взаимодействие ресурсов через вызовы RPC/RMI в программах на ЯП с помощью операций ресурсами и устанавливают связь ресурсов между собой при работе с Cloud Computing и Big Data. Система GRID обеспечивает обработку и управление программными, сервисными и др. Web-ресурсами глобальной сети работают с данными разного объема средствами ETICS и GRID (рис. 3). Ресурсы описываются в разных современных ЯП. Базовые сущности систем, пакет и приложений, а также связи описываются в CIM (Common Information Model).

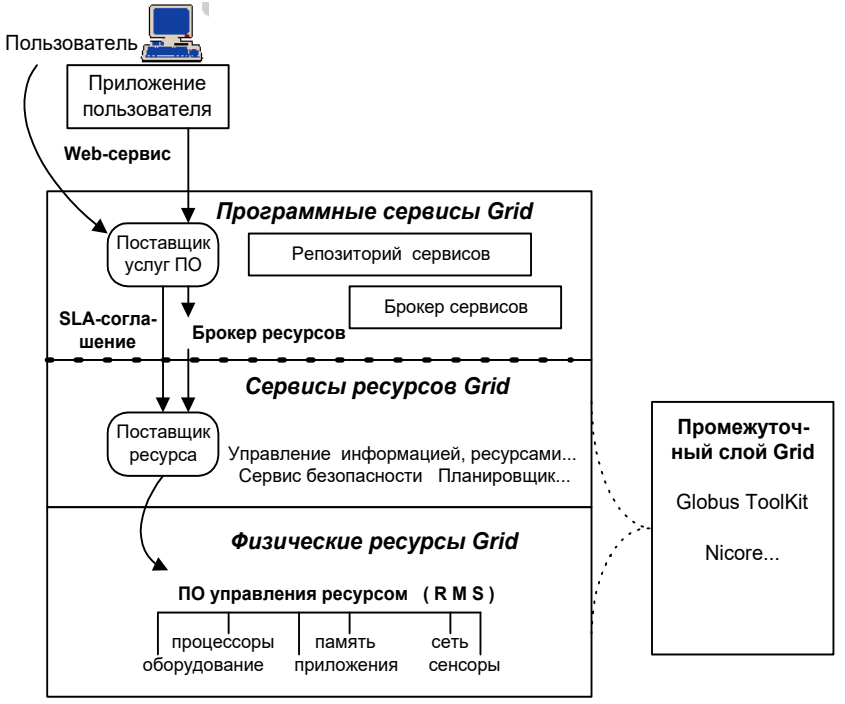


Рис. 3. Общая структура функционирования GRID

В системе ETICS (рис. 4) используется стандартизованное описание ТД для главных объектов: *Проект, Подсистема и Компонент. Проект.* Подсистема может содержать только Компоненты. В них используется модель данных CIM, задающая связь между разными объектами.

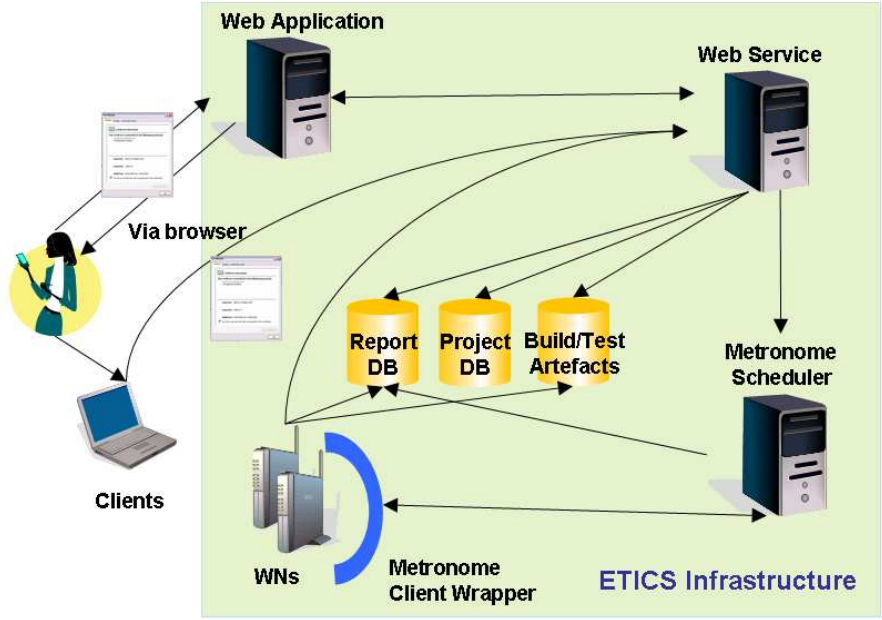


Рис. 4. Архитектура ETICS

ETICS Web Application для управления конфгурациєю проектов, зарегистрированных в хранилище метаданих ETICS;

ETICS Web Service содержит сервисы для функционирования Веб систем и обратывает запросы клиента к Веб-приложению с задачами сборки, тестирования и обработки Metronome; Metronome Execution Engine использует систему *Condor* для выполнения задач на рабочих узлах (worker nodes) для выполнения на разних мульти-платформенных средах сборки и тестирования;

ETICS Database после конфигурации накапливает данные в БД ETICS и установить зависимости между проектами и пакетами (packages) с управляемым качеством. Данные накапливаются в БД и называются Хранилищем *метаданных*.

Модель данных типа CIM, позволяет вводить формальные сущности приложений, описывать объекты и связи между ними, а также передавать результаты их выполнения по запросам. Сохранение и ведение данных базируется на модели реляционного типа в MySQL.

Описание модели данных основано на следующих базовых положениях:

1) каждый компонент содержит описание сведений (имя, лицензия, URL репозитория и т. п.) и глобального уникального идентификатора — ID (GUID);

2) объект конфигурации содержит информацию о версиях, связи с репозиторием, GUID, платформе фреймворка и связь с конфигурацией системы;

3) каждый объект проходит проверку (checkout) скомпилированного элемента, тестовых команд и GUIDs, а также связь с конфигурацией;

4) при определении конфигурации и платформы в каждом объекте появляется GUID, его свойства, среда выполнения и зависимости, которые могут объявляться статически или

динамически. Статическая зависимость — это взаимоотношение между двумя конфигурациями, динамическая зависимость — взаимоотношение между конфигурацией и модулем.

ETICS по функциям сборки близок концепции современных фабрик программ и базируется на наборах характеристик, услуг и процедур изготовления пакетов. Последние могут объединяться плагинами с описанием услуг для потребителей и поставщиков, обеспечивать управление заданиями с рабочих мест, а также давать доступ к OCAM, архитектуре CPU, компиляторам с ЯП и средствам спецификации зависимостей между разными пакетами и их тестами при сборке программ и их развертывании. Множество функциональных плагинов обеспечивает проверку контрактов, тестов выполнения разных элементов систем, генерацию документации, ведения готовых КПИ в оперативном или статическом репозитории ETICS.

Главная задача системы ETICS состоит в преобразовании некоторых компонентов систем для альтернативной платформы гетерогенной среды компьютеров с 16-, 32-разрядных платформ на 64-разрядную платформу среды Grid. Вопросами поддержки интерфейса ПО занимается *UNICORE* (UNiform Interface to Computing Resources) (www.unicore.org).

Обеспечение безопасности и защиты данных ресурсов и систем осуществляет инструмент WSRF (Web Service Recourse Framework). Кроме того, в среду системы Grid входят системные компоненты, которые обеспечивают моделирование физических экспериментов, проведение Cloud вычислений и обработку данных больших объемов:

Globus Toolkit (GGF) управляет сервисными SOA и SCA ресурсами.

Condor (www.cs.wisc.edu/condor) выполняет задания с интенсивными вычислениями, не требующими вмешательства человека в этот процесс.

WebFlow (http://www.npac.syr.edu/users/haupt/WebFlow/) обеспечивает публикацию ресурсов, многократное использование вычислительных модулей и создание распределенных приложений с помощью Web-браузера.

Nimrod/G (www.csse.monash.edu.au/~davida/nimrod/references.htm) предоставляет декларативный параметрический язык моделирования физических экспериментов, динамичное выявление требуемых ресурсов и их рассылку по сети Grid;

Gridbus Data Grid Service Broker развивает модель брокера ресурсов вычислительного Grid для использования в распределенных сетях, ориентированных на данные в удаленных Хранилищах сети.

GRACE (Grid Architecture for Computational Economy) — развитие *Nimrod/G* для динамического сотрудничества с владельцами ресурсов и выбора ресурсов за *оптимальную стоимость*.

Система GRID включает *Grid-порталы* и *Grid Port Toolkit* для управления созданием порталов и Веб-приложений разными пользователями.

НАЦИОНАЛЬНАЯ АКАДЕМИЯ НАУК УКРАИНЫ
ИНСТИТУТ ПРОГРАММНЫХ СИСТЕМ

ETICS

Базовые концепции и обзор возможностей

Семинар №2

Стадии ЖЦ ПС, поддерживаемые в ETICS
Базовые знания:
- **особенности систем сборочного типа**
- **процесс управления конфигурацией**
- **механизмы управления версиями в VCS-системах**

Место ETICS в электронной науке
Основные концепции ETICS
- **модель данных ETICS**
- **структура проекта**
- **объекты конфигурации**

Web-портал ETICS в режиме «Гость»

Коваль Галина Ивановна, Лаврищева Екатерина Михайловна

1

Базовые положения общего сборщика современных ресурсов

В качестве общего вывода по рассмотрению операций сборки можно сделать вывод о том, что приведенные операции сборки (*link, make, config, assembling, building, weaver, integration*) ресурсов в системных средах Интернет являются базовыми средствами и можно сделать общий «сборщик» готовых ресурсов для прикладных, сервисных, информационных и технических систем в сетевой Глобальной сети Интернет.

Рассмотренные операции конфигурационной сборки (link, make, config, assembling, weaver) ресурсов в системных средах Интернет (Corba, VS.Net, BSN, GNU, Grid, Etics) имеют схожие операционные способы обработки разных вариантов ресурсов: модулей в ЯП, исходных и выходных текстов компиляторных программ, сервисных и системных ресурсов Веб среды Интернет, технических средств компьютеров имеют похожие способы сборки, которые повторяются в разных общесистемных средах. Как бы не назывались способы сборки, семантика сборки остается одинаковой. Операция сборки зависит от данных, которые используются при обмене данными между связываемыми разнородными ресурсами.

Представлены средства генерации общих типов данных стандарта ISO/IEC 11404 GPD-2007 требуют создания набора новых примитивных функций для нестандартных типов данных (портфелей, контейнеров, шаблонов, указателей, неструктурных данных, больших данных и др.) и использоваться в названных способах сборки разнородных ресурсов Интернет.

Для создания общего «сборщика» готовых ресурсов в прикладные и технические (макроконвейерные) системы в Интернет, необходимо на базе конфигурационной сборки реализовать следующие задачи:

1. Определить формальный вариант описания операции сборки ресурсов

2. Создать библиотеку примитивных функций для всех систем Интернет и дорабатывать примитивные функции для новые ТД (контейнеров, шаблонов, указателей и др.) согласно стандарта GPD -2007.

3. Создать анализатор операций сборщика и взаимодействия компиляторных, системных, прикладных, технических, операционных систем с учетом всех типов ресурсов.

4. Сделать анализатор соответствия типов ресурсов и взаимодействия с другим ресурсом через интерфейс и осуществлять обращение к соответствующим программам преобразования обмениваемых данных для генерации соответствующего преобразования по теории преобразования типов данных стандарта GPD.

5. Реализовать конфигурационную сборку, выполняя все процессы, определенные в стандарте IEEE 828 Configuration:2012.

6. Выдавать сведения о результатах сборки и завершения работы сборщика.

7. Обеспечить размещение ресурсов в библиотеках и хранилищах Интернет из разных предметных областей знаний (медицины, биологии, генетики, математики и др.).

Техническим результатом способа сборки общего сборщика является:

– простота поиска ресурсов в хранилищах Интернет и снижение затрат на разработку за счет готовых правильных многоразовых ресурсов и настройку их на конкретные условия применения;

– повышение качества и производительности создаваемых из ресурсов Веб-приложений и систем за счет системных операций замены отдельных более правильных ресурсов и изменения конфигурации (архитектуры) варианта конфигурационного файла с получением качественных решений функциональных задач приложений в разных предметных областях знаний.

Технология создания вариантов ПТС, СПС и ядра OS Linux:

E.M. Lavrischeva, V.S. Mutilin, A.G. Ryzhov. Designing variability models for software, operating systems and their families, Proceedings of the Institute for System Programming of RAS-2017, Том 29, N5.-с.93-110. DOI10.15514./ispRAS-2017-29(5)-6.

С.В. Козин. Конфигурационная сборка варианта ядра Linux для прикладных систем. Труды ИСПРАН-2018.

Процессы технологии создания вариантов от ядра OS Linux

1) Процесс анализ функции ядра с их компиляцией для получения кода версии компонента или системы с исправлением обнаруженных ошибок.

2) Компиляция выбранных формально описанных функций из базового ядра OS Linux для получения выходного кода функции и проверка ошибок в синтаксисе описания.

3) Поиск функциональных элементов ядра OS и представления их в классах эквивалентности с верификацией, тестированием MC/DC, исходя из двух понятий: *решение* и *условие*. *Решение* определяется формулой, состоящей из условий и передач управления для выполнения. *Условие* задается логикой, влияющей на значение принимаемого решения при отборе функциональных элементов для их связей.

4) Сборка (связывание) отдельных элементов ОС с помощью операторов make, config с проверкой интерфейса собираемых функциональных элементов и данных, обмениваемых между ними, из внешних библиотек Интернет.

5) Конфигурация структуры варианта ОС с помощью операции config для получения файла конфигурации архитектуры с проверкой наличия зависимостей и несуществующих переменных в выходном коде.

6) Тестирование используемых элементов и собранного из них варианта ОС с поиском ошибок средствами LDV и CPAchecker с оценкой правильности элементов ядра OS Linux.

7) Оценивание элементов конфигурационной структуры нового варианта ядра OS на надежность, безопасность и защиту информации.

Описание Функциональных элементов OS Linux

OS Linux содержит более 10 000 переменных и большое множество функциональных системных компонентов, обеспечивающих обработку разного рода задач для управления функционированием прикладных систем и устройств. Из базового ядра ОС выбираются необходимые компоненты **OS**, помещаются в модель характеристик, включая функциональные, интерфейсные элементы и данные. Они тестируются на правильность с помощью наборов тестов и операций связи linking с другими элементами модели. Затем проверенные элементы собираются в конфигурационную структуру с помощью стандартной операции сборки *config/make* для получения нового варианта ядра **OS**.

Стандартная операция config имеет разновидности для ручной, автоматической, графической сборки вариантов **OS Linux**: oldconfig; defconfig; menuconfig; xconfig; gconfig и др.

Ядро OS Linux - трехуровневая. На верхнем уровне располагаются интерфейсы системных вызовов базовых функций (чтение и запись). Следующий уровень содержит архитектурно-независимый код ядра OS, общий для всех процессорных архитектур Linux. Затем располагается архитектурно-зависимый код, образующий BSP (Board Support Package) для разных архитектур платформ.

Основными компонентами ядра OS Linux являются:

– интерфейсы системных вызовов функций ядра;

– процессы, потоки, планировщик, компоновщик и управление;

– физическая, виртуальная память и файловая система VFS со средствами коммутации;

– сетевой протокол IP (Internet Protocol) и транспортный протокол TCP (Transmission Control Protocol) и др.;

– драйверы устройств и конкретные аппаратные устройства;

– гипервизор, с помощью которого строиться вариант OS для прикладных применений.

Стандартный набор пакетов ядра Linux

Для функционирования системы должны быть установлены следующие пакеты:

Linux-4.18.5 API Headers, Man-pages-4.16, Glibc-2.28, Zlib-1.2.11, File-5.34, Readline-7.0, M4-1.4.18, Bc-1.07.1, Binutils-2.31.1, GMP-6.1.2, MPFR-4.0.1, MPC-1.1.0, Shadow-4.6, GCC-8.2.0, Bzip2-1.0.6, Pkg-config-0.29.2, Ncurses-6.1, Attr-2.4.48, Acl-2.2.53, Libcap-2.25, Sed-4.5, Psmisc-23.1, Iana-Etc-2.30, Bison-3.0.5, Flex-2.6.4, Grep-3.1, Bash-4.4.18, Libtool-2.4.6, GDBM-1.17, Gperf-3.1, Expat-2.2.6, netutils-1.9.4, Perl-5.28.0, XML::Parser-2.44, Intltool-0.51.0, Autoconf-2.69, Automake-1.16.1, Xz-5.2.4, Kmod-25, Gettext-0.19.8.1, Libelf-0.173, Libffi-3.2.1, OpenSSL-1.1.0i, Python-3.7.0, Ninja-1.8.2, Meson-0.47.1, Procps-ng-3.3.15, E2fsprogs-1.44.3, Coreutils-8.30, Check-0.12.0, Diffutils-3.6, Gawk-4.2.1, Findutils-4.6.0, Groff-1.22.3, GRUB-2.02, Less-530, Gzip-1.9, IPRoute2-4.18.0, Kbd-2.0.4, Libpipeline-1.5.0, Make-4.2.1, Patch-2.7.6, Sysklogd-1.5.1, Sysvinit-2.90, Eudev-3.2.5, Util-linux-2.32.1, Man-DB-2.8.4, Tar-1.30, Texinfo-6.5, Vim-8.1.

При создании ядра OS определяются USB-драйверы (ehci_hcd, ohci_hcd и uhci_hcd), использовались модули загрузки ehci_hcd, ohci_hcd и uhci_hcd, обеспечивающие обнаружение ошибочных ситуаций и их исправления. GRUB записывает данные на первую физическую дорожку жесткого диска. Действующие программные элементы OS получают доступ к модулям GRUB в загрузочном разделе /boot/grub/ трека /boot/grub/grub.cfg.

Системные средства описания структур систем и устройств в OS

HAL (Hardware Abstraction layer) для поддержки нескольких аппаратных архитектур (процессы, семафоры и др.):

API (Application Programming Interface) для описания интерфейсов;

IPC (Inter process Communication) для коммуникации технических и программных средств безопасности SELinux Национального агентства по безопасности NSA, ориентированных на безопасность компонентов OS;

etc /udev /rules.d/70-persistent - средства проверки файлов /net.rules имен сетевых устройств, оборудования, драйверов и схем их именования с помощью Udev;

dev /cdrom и /dev/dvd - описание символических ссылок для устройств CD-ROM или DVD-ROM для устройств USB и FireWire;

Sys / class или / sys / block и / sys / class / video4linux / videoX - каталоги раздела видеоустройств;

ifconfig.xyz - спецификация интерфейсов с помощью сетевых сценариев etc / sysconfig /.

спецификация исходных файлов функциональных элементов в ЯП (Fortran, Cobol, Ada, C, C++, Basic, Smalltalk, …).

Системные средства создания и описания экспериментального варианта ядра OS Linux

OS Linux (около 6 гигабайт ГБ) содержит дисковый раздел для хранения всех исходных архивов и компиляции пакетов. При настройке жесткого диска для использования OS Linux наиболее удобной установкой является начальная загрузка жесткого диска хоста. Если на хосте уже есть один IDE-диск, который рассматривается как /dev/hda, IDE-диск рассматривается hdb или hdc, в зависимости от настройки хоста. Затем монтируется диск. Единственное различие состоит в том, что целевой диск, использованный на хосте, как вторичный диск /dev/hdb или /dev/hdc, рассматриваться hda для компиляции пакетов в разделе диска с подкачкой (swap) компонентов рабочих станций Linux и серверов путем обращения к большой памяти и эмуляции памяти на запоминающем устройстве.

Большинство встроенных запоминающих устройств flash и DOC-девайсы, имеют ограниченные циклы стирания и записи. При создании ядра уменьшается память приложений до минимального набора двоичных файлов. Диск разбивается специальной утилитой cfdisk или fdisk, на котором будет создан новый раздел /dev /sda для основного диска Integrated Drive Electronics (IDE). После создания этого раздела формируется файловый вариант версии OS Linux с помощью средств языка Kconfig:

– ext2 для небольших разделов, которые обновляются не часто и диск разбивается на дополнительные участки;

– ext3 апгрейд для ext2, который включает журнал для восстановления статуса раздела после его отключения;

– ext4 версия семейства файлов типа ext, включающая несколько новых возможностей - наносекундные временные метки, создание и использование очень больших файлов (16 ТБ) для повышения скорости.

При выборе функций и списка типов данных, хранящихся в файловой системе OS Linux библиотеки, формировалось ПО для устройств и систем переменной конфигурации в XML. При создании нового ядра OS USB-драйверы (ehci_hcd, ohci_hcd и uhci_hcd) вводились модули, загружаемые в правильном порядке; модуль ehci_hcd загрузки ohci_hcd и uhci_hcd с проверкой ошибочных ситуаций во время загрузки; GRUB и записи данных на первую физическую дорожку жесткого диска с доступом к модулям GRUB в загрузочном разделе /boot/grub/. Создаваемый раздел OS делался размером около 100 МБ с информацией для проведения конфигурационной сборки нового варианта ядра OS Linux.

Операции Kconfig, CDL для описания моделей ядра OS

Kconfig разработан для ядра Linux и применяется с 2002 года. Он предоставляет конфигуратор xconfig, позволяющий выбирать характеристики в графическом интерфейсе. В

настоящее время это язык поддерживается и развивается языке CDL (Component Definition Language). Оператор Kconfig был разработан для ядра Linux и применяется с 2002 года. Он предоставляет конфигуратор xconfig, позволяющий выбирать характеристики в графическом интерфейсе. В настоящее время это язык поддерживается и развивается языке CDL (Component Definition Language). Язык Kconfig использовался для описания операционных систем (и системного ПО) и при конфигурации ядра Linux с 2002 г. Язык CDL использовался для систем реального времени eCos и встроенных систем. Оба языка поддерживаются методом FODA (Feature-Oriented Domain Analysis). Исследования этих языков проведен Митулиным в рамках проекта РФФИ. В зарубежной литературе R. Zippel описывал модели характеристик систем средствами Kconfig так:

kconfig- language.txt, available in the kernel tree at www.kemel.org, seen 2012-04/10.] и CDL [cite B. Veer and J. Dallaway, “The eCos component writer’s guide,” seen Mar. 2010 at ecos.sourceware.org/ecos/docs-latest/ cdl-guide/cdl-guide.html.]. Обе модели используются в открытых операционных системах (IBM, VS.MS, Intel, Linux и др.).

Kconfig использовался для описания вариантных версий OS Linux и десяти других проектов с открытым кодом. Язык CDL как часть eCos — операционной системы реального времени для встроенных устройств. Как OS Linux так и eCos имеют большой набор опций конфигурации с тысячами характеристик, что удобно при управлении вариантностью. Он g позволяет правильно опись зависимость между характеристиками и инструментами Kconfig. В выражениях зависимости между характеристиками включаются любые другие характеристики с помощью конструкции SELECT для оценки правильности. Однако в зарубежных статьях отмечается, что конструкция SELECT является причиной значительной части ошибок в конфигурации [cite R. Lotufo, S. She, T. Berger, K. Czamecki, and A. Wasowski, “Evolution of the Linux kernel variability model,” in SPLC, 2010.]. Ошибки в описаниях Kconfig могут привести к невозможности выбора некоторых опций для покрытия некоторых участков кода (Y. Xiong, A. Hubaux, S. She, and K. Czamecki, “Generating range fixes for software configuration,” in ICSE, 2012.

Оба языка конфигурации **Kconfig и CDL** поддерживают сравнительно небольшой набор операций анализа по сравнению с тем, что имеется в академических языках [D. Benavides, S. Segura, and A. Ruiz-Cortes, “Automated analysis of feature models 20 years later: A literature review,” Information Systems, vol. 35, no. 6, pp. 615-636, 2010.]. С учетом приведенных замечаний средства языков Kconfig и CDL развиваются при проведении конфигурационной сборки операциями по стандарту IEEE 828 Configuration-2012: make, cmake, config, assembly, build, waver.

Проблемы анализа конфигураций ОС Linux

Конфигурирование структур ОС и систем требует автоматизированного тестирование вариантов ОС с поиском ошибок. Для корректной проверки, инструменты должны производить обработку одной конкретной конфигурации, поэтому программисты должны вручную извлечь множество конфигурационых элементов, чтобы гарантировать, что все конфигурационно-условные части кода проверены. Основные действия для выполнения статического анализа следующие:

1) В процессе сборки версии OS Linux выполняются правила выполнения функций Makefile, способствующих компиляции соответствующих С-файлов и обработки параметров конфигурирования файлов и загружаемых модулей ядра (LKM).

2) Генерация вариантов проводится путем выбора конфигурационных опций из модели вариабельности, описанной на языке Kconfig. Каждая опция Kconfig имеет тип (bool, tristate, string, int), который определяет выбор значений для входа. Kconfig.

3) Kbuild скрипты Makefile или Kbuild файлы запускают процесс сборки. Данные скрипты написаны на языке make и распределены по коду ядра. Большая часть Makefile файлов в Linux находятся в подкаталогах опций, и о отвечают за установку make переменных obj-y, obj-m, и obj-n, тем самым сообщая top-Makefile файлу список файлов, которые должны

быть построены либо статически, либо как загружают модуль ядра, или же не должны быть построены.

4) Kconfig гереводит описание в пропозиционной логике два типа конфигурационных переменных, один из которых заканчивается суффиксом _MODULE и передается для С-препроцессора и Makefile-файлов. Этот процесс вводит логические ограничения, которые могут задать неточности и осуществлять конфигурационно-зависимый дефект.

Проведение верификации моделей и диаграмм характеристик

К инструментам проведения верификации х моделей относятся:

Feature Model Plugin (FMP) или FeaturePlugin. Инструмент реализован в Eclipse и обеспечивает моделирование моделей характеристик МХ и конфигурирования СПС на их основе. Использует BDD-решатель для определения количества возможных комбинаций характеристик в модели характеристик, но не поддерживает атрибутные модели характеристик (attributed feature models) http://gsd.uwaterloo.ca/projects/fmp-plugin.

xFeature основан на XML инструменте моделирования характеристик и поддерживает моделирование СПС без анализа модели характеристик. Доступный по адресу – http://www.pnp-software.com/XFeature/ Download.html.

CaptainFeature инструмент для моделирования характеристик с помощью нотации FODA для построения модели МХ с анализом моделей:

https://ngfe1049srsid5978550sourceforge.ngfe1049srsid5978550net/ngfe1049srsid5978550 projects/ngfe1049srsid5978550 captainfeature/.

Requiline - инструмент инженерии требований для эффективного управления линиями программных продуктов. Включает функцию контроля checker и используется в таких средах: Microsoft.NET, Oracle DBMS. Доступен по адресу http://www-lufgi3.informatik.rwth-aachen.de/TOOLS/requiline/ Product/download.php при наличии заявки по e-mail.

Pure::Variants -клиент поддержки моделирования характеристик, конфигурирование и базовые операции анализа через Prolog решателя с ограничениями constraint solver. Доступен по адресу – http://www.software-acumen.com/downloads/.

AHEAD Tool Suite (ATS) - набор инструментов для разработки линий продуктов, модуляризацию характеристик и их компоновку. Может выполнять определенные операции анализа модели характеристикс помощью SAT-решателей и сохранять ее в виде грамматики. Доступен по адресу –http://www.cs.utexas.edu/~schwartz/ATS.html.

Из представленных инструментов фактически только FeaturePlugin поддерживает практически анализ модели характеристик по адресу – http://gp.uwaterloo.ca/fmp2rsmVerifier/demo.htm – на примере (demo) использования верификатора FeaturePlugin. Анализ этих инструментов показывает, что наиболее подходящий инструмент для работы с ОС – CaptainFeature.

6.2. Стандартные средства для проведения конфигурационной сборки:

POSIX (Portable Operating System Interface) -2012, включающий набор стандартов, описывающих интерфейсы между ОС, прикладными программами в API, библиотекой в ЯП (C, C++) и набор приложений с интерфейсами в среде UNIX OS и Intel OS.

FHS (File system Hierarchy Standard) Version 3.0 - стандарт иерархической файловой структуры, унифицирующий местонахождение файлов и каталогов в UNIX OS и BD файл f /etc/passwd.

LSB (Linux Standard Base) Version 5.0 (2015) -LSB Core: Bash, BC, Binutils, Coreutils, Diffutils, File, Findutils, Gawk, Grep, Gzip, M4, Man-DB, Ncurses, Procps, Psmisc, Sed, Shadow, Tar, Util-linux, Zlib, SB Runtime Languages, Perl.

На конфигураторе Интернет реализуется сборка КПИ, Reuses и сервисных и информационных ресурсов. преобразование исходного кода в код для выполнения на компьютере. Одним из шагов *сборки* является процесс компиляции исходного кода, где файлы превращаются в промежуточный код или в код выполнения, например, в среде VS.Net. Для сложных программ после компиляции устанавливается связь (нахождение реального положения всех внешних функций), которая выполняется специальной программой — связник. Процесс связи представляет собой замену относительных адресов функций внешних библиотек в реальные адреса, используемые в программе при выполнении. В этом случае конфигуратор является системой автоматизации и предоставления интерфейсов элементов для сборки ПС из набора ГОР в среде СПС.

К операциям управления конфигурации вариантов относятся функции:

– идентификация элементов конфигурации и данных;

– определение функций управления процессом изменений;

– модели и методы вариантности;

– метрики оценки надежности и защиты данных.

В процессе управления конфигурацией собираются данные для проведения процедур:

– управление сборкой ГОР и инструментов построения ПТС;

– управление процессами как гарантирование соблюдения плана развития ПТС;

– управление средой — управление программным и аппаратным обеспечением;

– взаимодействия членов команды по отслеживанию дефектов и оценки характеристик надежности.

6.3. Проведение сборки экспериментального варианта ядра OS Linux

Основной раздел OS Linux - Binutils. Он устанавливается первым. Glibc выполняет различные функциональные тесты компоновщика и определяет, какие программные функции включены или отключены. Сбой тестового набора позволяет выявить ошибки при установке некоторых функций ОС. Раздел Binutils устанавливает свой ассемблер, компоновщик, заголовки API Linux и дает возможность стандартной библиотеке Glibc взаимодействовать с включенными функциональными элементами экспериментального ядра OS Linux. В этот вариант входят компилятор, бинарные инструменты и заголовки ядра. Glibc использует компилятор, относящийся к параметру host, переданному скрипту configure (компилятор i686-lfs-linux-gnu-gcc).

Configure building осуществляется файлом config / make каталога glibc-build механизма сборки. Во время вторичной установки Binutils используется переключатель конфигурации — with-lib-path для управления поиском библиотеки ld. При вторичной установки GCC вносятся изменения для запуска динамического компоновщика из встроенного в него каталога хост-системы /lib и последующего выхода из хоста. При управлении каталогом инструментов и файлами в новую OS Linux добавляются LFS файлы /etc/passwd на хост-системе.

OS Linux содержит нескольких узлов устройств, консольных и null-устройств, узлы которых находятся на жестком диске. Они доступны запуску udevd OS Linux с init = /bin/bash. Заполнение каталога /dev устройствами путем подключения виртуальной файловой системы tmpfs каталога /dev. Запуск устройств и приборов выполняется во время загрузки Udev и вручную устанавливается, и заполняется /dev и связь мониторинга каталога хоста /dev. Bind mount обеспечивает монтирование зеркала каталога с помощью точек монтирования.

Для выполнения конфигурационной сборки версии варианта ОС, LFS делает вход в среду chroot каталога с доступом 755. При этом вносятся два изменения: один в каталог root, а другой – во временные файлы. Первое изменение обеспечивает гарантированный вход в

/root, второе изменение вносится в директорий /tmp и /var/tmp без удаления файлов другого пользователя (так называемым «sticky bit»). Дерево каталога, сформированное с помощью FHS, содержит каталоги - /usr/local/games и /usr/share/games, с помощью символических ссылок для замены реальных файлов и списка смонтированных файловых систем в файле /etc/mtab.

Варианты конфигурирования ОС в средах BSD и Grid

В ходе сборки ядра OS Linux применялась утилита *make*, использующая специальные make-файлы, в которых указаны зависимости файлов друг от друга и правила их использования. На основе информации о времени последнего изменения каждого файла *make* определяет и запускает необходимые программы. Операция *make* выступает в роли инструмента управления сборкой для получения конфигурационной структуры в JavaEE:

config – общий способ конфигурирования, в котором задаются операторы конфигурации последовательно одному с соответствующей реализацией;

oldconfig - создаётся автоматически файл конфигурации, основываясь на текущей конфигурации ядра;

defconfig - создаётся автоматически файл конфигурацииэлементов основываясь на их значениях по умолчанию.

menuconfig - псевдографический интерфейс ручной конфигурации, не требует последовательного ввода значений параметров с терминала;

xconfig - графический (X, QT) интерфейс ручной конфигурации, не требует последовательного ввода значений параметров;

gconfig - графический (GTK+) интерфейс ручной конфигурации, не требует последовательного ввода значений параметров в среде GNOME и др.

Стандартная конфигурация, основанная на операции *make в* JavaEE*,* может быть выполнена и через *config*, после чего осуществляется команда *make* в случае установки ОС на многоядерную машину с заданием количество потоков для ускорения процесса сборки.

Сборка с помощью операции build в Grid. Это процесс получения продукта из исходного кода компонентов после проведения компиляции программных компонентов к выходному коду. Операция сборки *build* ядра ОС Linux выполняется утилитой *make*, использующей специальные *make-файлы,* в которых указаны зависимости файлов друг от друга и правила для их удовлетворения. Оператор *make* реализован в BSD (make) и GNU (cmake). Они позволяют собирать исполняемые файлы из библиотеки программных ресурсов - OS Linux. Порядок сборки ресурсов задается в файле *Makefile*, в котором указываются правила (зависимость, tab) и действия над элементами библиотеки ОС средствами системы GNU или Visual Studio (примеры 1 и 2).

В создаваемый вариант ядра входят стандартные элементы базового ядра OS Linux:

– функции сжатия данных для арифметических вычислений рациональных чисел с плавающей точкой;

– функции чтения аудио-файлов, компиляторов в языках C, C++, Basic, Python систему защиты паролей, интерфейсы pkg, текстовый редактор, операции работы с сетью и определения требуемой информации из библиотеки баз данных, вызова генератора хэш функций, XML-парсера, openSSL и некоторых других функций;

– выбора операторов сборки - make BSD / cmake GNU для вызов исполняемых файлов из библиотеки и сборки ресурсов в ЯП в общую формальную структуру (см. рис.1, 2).

Операции Make in системе GNU

```
edit : main.o kbd.o command.o display.o \
insert.o search.o files.o utils.o
cc -o edit main.o kbd.o command.o display.o \
insert.o search.o files.o utils.o
```

```
main.o : main.c defs.h
cc -c main.c
kbd.o : kbd.c defs.h command.h
cc -c kbd.c
command.o : command.c defs.h command.h
cc -c command.c
display.o : display.c defs.h buffer.h
cc -c display.c
insert.o : insert.c defs.h buffer.h
cc -c insert.c
search.o : search.c defs.h buffer.h
cc -c search.c
files.o : files.c defs.h buffer.h command.h
cc -c files.c
utils.o : utils.c defs.h
cc -c utils.c
clean :
rm edit main.o kbd.o command.o display.o \
insert.o search.o files.o utils.o
```

Рис.1 Пример Makefile в GNU.

Операция Make в Visual Studio.VS

Генерация CMakeLists.txt к Makefile в VS включает следующие операции:

- add_executable(exec_name source1 source2 …) # создает исполняемый файл exec_name из файлов исходного кода source1, source2, и т.д.

- add_library(lib_name sout rce1 source2 …) # создает библиотеку lib_name из файлов исходного кода компонентов source1, source2, и т.д.

- target_include_directories(target PUBLIC dir1 dir2 …) # включает директории dir1, dir2,

- в список поиска заголовочных файлов при сборке (файла или библиотеки) target.

target_link_libraries(target lib1 lib2 …) # выполняет link сборки библиотек lib1, lib2.

cmake выполняет исполняемый файл myExec cmake_minimum_required (VERSION 2.6), project (MyProject).

- add_executable(myExec source1.cpp source2.cpp).

Рис.2. Реализация операции cmake в Visual Studio.

Приведенные операции на рис. 1, 2 обеспечивают конфигурационную сборку компонентов в моделируемой общесистемной средах Интернет, которые записаны в разных ЯП (C, C++, Basic, Java, Pyton, Ada и др.). Элементы модели системы проверяются на правильность (верификацию, тестирование) задания их функциональности при сборке make / config и данных согласно стандарту IEEE 828-1996-2012 (Configuration) с файлами версии OS Linux в выходном файле.

Набор инструментов сборки в JAVA Интернет

Apache Ant инструмент сборки компонентов в JAVA является платформо-независимый аналогом UNIX-утилиты make и использется среде языка JAVA, приспособленный для JAVA-объектов. Разница между Ant и Make состоит в том, что Ant использует язык XML для описания процесса сборки и его зависимостей, а Make имеет свой

собственный формат файл Makefile. XML-файл (или build.xml) осуществляет сборку исполняемых программ.

Apache Maven — программный инструмент для управления (*management*) JAVA-проектами сборкой *building* ПО аналогично Apache Ant, но с более простой *build-моделью* конфигурации, которая базируется на формате XML. Движок ядра инструмента динамически загружает плагины из репозитория и обеспечивает доступ ко многим версиям разных JAVA-систем с открытым кодом Apache. В системе Java, BSD для сборки вариантов конфигурации используется стандартная операция *config*, с помощью которой формируются файлы конфигурационного типа;

oldconfig – основанной на текущей конфигурации ядра;

defconfig – основанной на стандартной операции по умолчанию;

menuconfig – основанной на псевдографическом интерфейсе без ввода значений параметров на терминале;

xconfig – на основе графического (X, QT) интерфейса ручной конфигурации;

gconfig - графический (GTK+) интерфейс ручной конфигурации, не требующкго последовательного ввода значений параметров в среде GNOME и др.

Таким образом, создан экспериментальный вариант ядра OS Linux (см. Козин С.В., Конфигурационная сборка ядра OS Linux для прикладных систем. Труды ИСП РАН. М.: 2018, Том 29. Вып.4.-с.217-230.-М.: 2018, Том 30. Вып.6.-с.161- 170.) и представлен в Приложении.

6.4. Современные средства обеспечения гарантоспособной надежности ПТС и ОС

В классической теории надежности в 1986г. появился термин dependability (*гарантоспособность)*, который обозначает отказоустойчивость, живучесть, информационная безопасность (целостность, конфиденциальность). Представитель технического комитета IFIP «Гарантоспособные вычисления и отказоустойчивость» J.C. Laprie в 1992г. написал книгу «Гарантоспособность. Основные определения и терминология» и стандарт «Критерии оценки безопасности информационных технологий (ITSEC)», принятых в Германии, Франции, Великобритании, Италии и США. Эти критерии постоянно совершенствуются и согласуются с разными странами-участницами IFIP. Так, департамент компьютерных наук университета Вирджинии (Department of Computer Science University of Virginia) занимается повышением живучести информационных систем (ИС) в критических инфраструктурах.

Центр надежных и высокоэффективных вычислений Иллинойского университета (Center for

Reliable and High - Performance Computing University of Illinois) модифицировал программный продукт Chameleon до уровня функциональной надежности КС с сетевой структурой. В нем Fault – Tolerance Manager обеспечивает адаптацию гарантоспособных средств в разных вычислительных средах (однородной, гетерогенной и кластеризованной). Департамент компьютерных наук университета в Теннесси (Department of Computer Science University of Tennessee) разрабатывает средства для отказоустойчивых сетевых вычислений NetSolve, усовершенствуя клиент-серверную структуру Интернет. В ней модульность продукта давала возможность вести гарантоспособные вычисления на двух уровнях: внутреннем и межсерверном.

Ведущими учреждениями в области гарантоспособности США являются: компания IBM с Исследовательским центром им. Дж. Уотсона; Лаборатория им. Ч. Дрейпера Массачусетского технологического института; Международный Стэнфордский научно-исследовательский институт; корпорация Boing; Лаборатория автоматики и системного анализа (LAAS) Тулузского национального научно-исследовательского центра, компания Schneider Electric, SRC и CSR Англия, Siemens Германии и другие.

В США создана в 22.05.1998г. президентская директивы PDD-63 «Программы гарантирования всесторонней защиты инфраструктур» DIAP (Defense – wide Information Assurance Program). Эта программа активно поддерживается оборонными ведомствами, Германией и НАТО. К ней присоединились Канада, Евросоюз, Японии и др. страны IFIP.

Наиболее известными работами в гарантоспособности являются:

Avizienis A., Laprie J.-C., Randell B., Landwehr C., Dobson I.E (Basic Concepts and Taxonomy of Dependable and Secure Computing / A. Avizienis, J.-C. Laprie, B. Randell [et al.] // IEEE Trans. on Dependable and Secure Computing. – 2004. – Vol. 1, N 1. – P. 11 – 33 и др.

С позиций этих работ для проведения гарантированных достоверных вычислений КС должны предоставляться надежные гарантоспособные сервисные услуги, ориентированные на безотказность, защиту и ремонтопригодность без катастрофических последствий; обеспечение конфидициальности (confidentiality), живучести и качества систем и устройств.

В рамках проекта РФФИ-206 проведены исследования гарантоспособности с учетом возникающих атак, аварий в процессе вычислений ПТС, выполненные ИСП РАН:

– Липаев В.В. Качества программного обеспечения.-Синтег.-2011. IX international conference “Strategy of Quality in Industry and Education”. – Varna, Bulgary, 2013. – Vol. 1. – 396p.

– Зеленов С.В. Лаврищева Е.М. Модельный подход к обеспечению безопасности и надежности Web-систем. Труды ИСП РАН 2020, Том 32 № 5. С.151-163. Представлен вероятностный метод Маркова и приводится пример системы торможения системного устройства с учетом приведенных средств гарантоспособности и авиационного стандарта ARP-4761.

Таким образом, в рамках данного проекта РФФИ опробованы методы и средства моделирования прикладных и операционных систем, в частности создание экспериментальной версии варианта ядра OS Linux с анализом возникающими аварий, контратак при выполнении отдельных компонентов данного ядра и защиты данных при вычислении задач, работающих с большими объемами данных (Cloud Computing, Big Data).

Задачи гарантоспособных сервисных, прикладных и операционных систем

Устойчивость компьютерных систем определяется их способностью обнаруживать и парировать отказы и сбои, часть из которых не была предусмотрена на этапе проектирования этих систем. Особенно остро такая задача стоит при построении Веб-систем и облачных систем, которые характеризуются глобальной распределённостью компонентов, их гетерогенностью и высокой сложностью. При построении этих систем практически невозможно предусмотреть все возможные исключительные ситуации (ошибки, отказы и сбои) и реализовать методы их парирования.

Сервис-ориентированные системы задают компонентную интеграцию сервисов, предполагающую их параллельно последовательное функционирование при решении комплексной задачи. Основными проблемами обеспечения надежности таких систем являются:

1) *нестабильность функционального состава* и структуры СОС;

2) *многовариантность результатов обслуижвания*, которая проявляется в том, что при вызове Web-сервиса может быть получен корректный результат обслуживания; некорректный (неочевидныйошибочный) результат, очевидный ошибочный результат (явное сообщение о возникшей ошибке), а также отсутствие любого из перечисленных выше результатов в течение установленного времени ожидания;

3) *неопределенность характеристик гарантоспособности* отдельных компонентов (т.е. Web-сервисов) – доступности, безотказности, готовности, достоверности и др.;

4) *неопределенность характеристик оперативности* Web-сервисов, прежде всего, времени обработки запросов и *непредсказуемость изменения сетевой задержки*,

индивидуальной для каждого клиента, что обусловлено глобальностью, недостаточно высоким качеством и надежностью сети Интернет;

5) *сложность эффективного применения традиционных средств повышения гарантоспособности и оперативности* из-за невозможности точного диагностирования возникающих отказов, а также отсутствия достоверной априорной информации о значении надежностных характеристиках Web-компонентов и оперативности обслуживания.

Таким образом, можно констатировать, что, несмотря на существенный прогресс в информационных технологиях на сегодня отсутствуют устоявшиеся методы и технологии измерения, оценки и прогнозирования характеристик Web-сервисов и сервис-ориентированных систем в условиях их использования множеством независимых клиентов, неравномерности поступления запросов, отсутствие информации о текущем состоянии (загруженности, готовности и др.) удаленных Web-компонентов.

Для достоверной оценки надежности гарантоспособности и оперативности отдельных Web-сервисов фактически нужно прогнозировать характеристики интегрируемых сервисных систем, а также синтеза таких систем с требуемыми характеристиками. Современные собираемые системы должны обладать способностью отрабатывать отказы, сбои и потенциально-опасные события, обусловленные разнообразными причинами, включая изменяющиеся характеристики среды взаимодействия, отказы и ошибки интеграции, разрывы сетевых соединений и др.

Возможность создавать качественную инфраструктуру ПТС является важным условием развития науки, обороны, медицинского обслуживания, улучшения качества повседневной жизни, расширения рынка Web-услуг, эффективного функционирования предприятий и государственного аппарата (например, программы NEC, NESSI, NECTISE, EPSRC, DTI, EC, FP7 и др.

Анализ видов последствий и критичности отказов

Одним из методов оценки безопасности программно-аппаратных систем является анализ видов и последствий отказов (ГОСТ 27.310-95).

Анализ видов и последствий отказов является анализом “снизу вверх”, когда анализ стартует с некоторого сбоя в отдельном компоненте и определяет, к каким последствиям может привести этот сбой. При проведении этого анализа:

– выявляют возможные сбои компонентов и системы в целом, изучают их причины, механизмы и условия возникновения и развития;

– определяют возможные неблагоприятные последствия возникновения выявленных отказов, проводят качественный анализ тяжести последствий отказов и/или количественную оценку их критичности.

Критичность отказов оценивают с использованием показателей, учитывающих для каждого анализируемого отказа:

– вероятность его возникновения за время эксплуатации;

– условные вероятности наступления всех возможных неблагоприятных последствий отказа;

– размер возможного ущерба в результате наступления каждого из ожидаемых последствий отказов.

Инструментальная поддержка анализа рисков

В рамках проекта РФФИ в ИСП РАН реализованы следующие методы анализа надежности технических систем:

Анализ дерева неисправностей методами логико-вероятностного анализа.

Марковский анализ методами теории марковских процессов.

Анализ видов последствий и критичности отказов.

Анализ надежности технической системы на основе формального описания системы на языке ADL. В рамках *анализа дерева неисправностей* для заданного отказа целевого компонента автоматически строится дерево неисправностей и автоматически вычисляются минимальные сечения и вероятность возникновения заданного отказа; атомарные отказы подкомпонентов автоматически ранжируются по мере значимости относительно величины вклада в вероятность возникновения заданного отказа.

В рамках *марковского анализа* автоматически строится марковская цепь и составляются уравнения Колмогорова-Чепмена, формулируется задача Коши для численного ее решения и вычисляется вероятность отказа целевой компоненты в заданный момент времени. Решение полученной задачи Коши и нахождение вероятностей отказов данной (под)системы в заданные моменты времени осуществляется численно методом Рунге-Кутта.

А*нализ видов последствий и критичности отказов* для заданной комбинации отказов целевых компонентов вычисляются возможные причины и их вероятности; комбинации отказов целевых компонентов автоматически вычисляются последствия и критичность системы в целом.

Поддерживаются две разновидности анализа видов и последствий отказов:

1. исходными сбоями считаются внутренние ошибки компонентов;
2. исходными сбоями считаются нерабочие состояния компонентов.

В первом случае (когда исходными сбоями считаются внутренние ошибки компонентов) таблица видов и последствий отказов соодержит следующие данные:

- Item: компонент, в котором произошел исходный сбой;
- Initial failure mode: исходный сбой (событие) в данном компоненте;
- End effect: набор финальных последствий данного исходного сбоя;
- Sev: тяжесть данного последствия для системы (по 10-балльной шкале);
- P: вероятность достижения данного последствия в результате возникновения данного исходного сбоя.

Во втором случае (когда исходными сбоями считаются нерабочие состояния компонентов) таблица видов и последствий отказов соодержит следующие данные:

- Item: компонент, в котором произошел исходный сбой;
- Initial failure mode: исходный сбой (нерабочее состояние) в данном компоненте;
- Potential cause: возможные причины данного сбоя;
- Cause: описание непосредственной причины сбоя;
- P: абсолютная вероятность возникновения данного сбоя;
- Occ: рейтинг вероятности возникновения данного сбоя (по 10-балльной шкале);
- End effect: набор финальных последствий данного исходного сбоя;
- Sev: тяжесть данного последствия для системы (по 10-балльной шкале);
- P (cond): условная вероятность достижения данного последствия в результате возникновения данного исходного сбоя.

Обнаружение ошибок, дефектов, отказов и сбоев Веб-службами Интернет

Общепринятая концепция надежности компьютерных систем определяет следующие угрозы их функционированию: ошибки (errors), дефекты (faults, bugs) и отказы/сбои (failures).

Дефект в компьютерной системе, например, дефект в программном обеспечении, допущенный вследствие ошибки программиста, может привести к отказу или сбою системы. Отказ/сбой системы диагностируется, когда ошибочный результат распространяется за пределы интерфейса системы. Дефект – это гипотетическая причина, приведшая к получению ошибочного результата. Обычно различают три группы дефектов компьютерных систем: дефекты проектирования и разработки, физические неисправности и ошибки взаимодействия.

Основными этапами взаимодействия веб-сервисов являются:

1) привязка (binding) веб-службы;

2) вызов (invocation) веб-службы;

3) обмен сообщениями SOAP между клиентом и веб-сдужбой;

4) обработка веб-сдужбой запросов клиентов и подготовка результата.

Предварительно был составлен список из 18 различных ошибок/дефектов, специфичных для сервис-ориентированных систем, возникающих на всех указанных этапах (см. Таблицу 1), засев и анализ проявления которых предлагается выполнить в процессе выполнения лабораторной работы. Эти ошибки могут быть условно разделены на три категории:

1) сбои сети и отказы удаленной веб-службы

2) внутренние ошибки и сбои веб-службы и 3) ошибки привязки на стороне клиента.

Указанные ошибки/дефекты являются общими (не специфичными для конкретного приложения) и могут проявляться в любой веб-службе во время её работы.

По своему усмотрению слушатели могут дополнить представленный список ошибок, дефектов и сбоев. Сообщения о возникших исключительных ситуациях (exception messages), генерируемые программной системой, являются проявлением симптомов возникновения/активации ошибки, сбоя или дефекта.

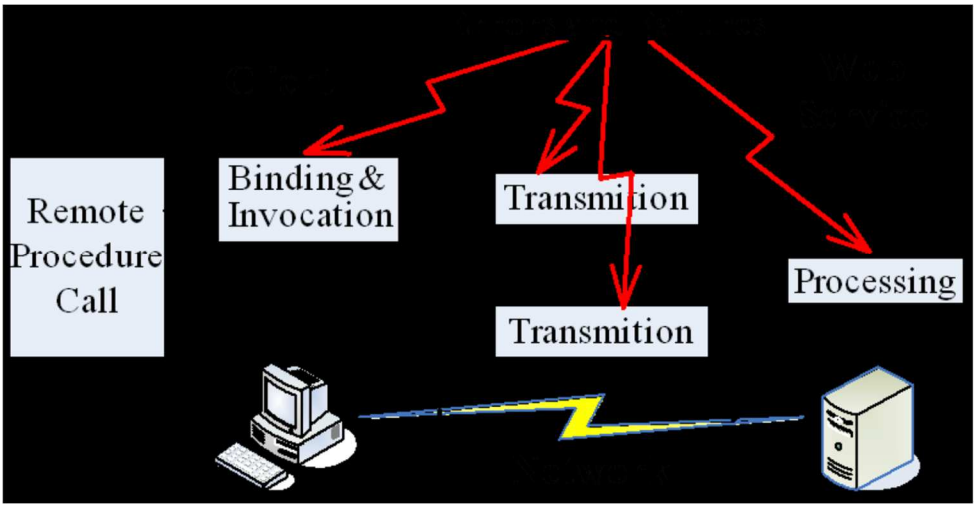


Рис. 3. Потенциальные. места возникновения ошибок, отказов и сбоев в ПТС

Таблица 1. Перечень ошибок, дефектов и сбоев для их засева в тестовую веб-службу

№ Type of error/failure Error/failure domain

1. Network connection break-off Network and system failures
2. Domain Name System is down
3. Loss of request/response packet
4. Remote host unavailable
5. Application Server is down
6. Suspension of WS during transaction Service errors and failures
7. System run-time error
8. Application run-time error
9. Error causing user-defined exception
10. Error in Target Name Space Client-side binding errors
11. Error in Web Service name

12. Error in service port name
13. Error in service operation's name
14. Output parameter type mismatch
15. Input parameter type mismatch
16. Error in name of input parameter
17. Mismatching of number of input params
18. WS style mismatching ("RPC" or "Doc")

К традиционным сетевым отказам и сбоям относятся службы DNS. Безотказная работа ПТС обеспечивается Веб-службами и средствами серверов WebSphere, Apache Tomcat и IIS. Ошибки могут возникать и на стороне клиента при раннем связывании прикладных компонентов. К ним относятся: «Ошибка в пространстве имен», «Ошибка в имени веб-службы», несоответствий между WSDL описанием веб-службы и фактическим интерфейсом вызова и др. Сбои и отказы Веб-служб могут быть возникать во время их выполнения (run-time errors): «переполнение стека», «нехватка памяти», «деление на ноль», «несоответствие типа операнда», «выход индекса за пределы массива» и др.

В таблице 1 приведены ошибки несоответствие типа операнда, зацикливание процесса, а также ошибки исключения. Так отказы (6, 7, 8) могут быть симулированы с помощью внедрения соответствующих ошибок в программный код сервиса. Ошибки связывания (10-18) реализуются на стороне клиента с помощью передачи веб-сервису ошибочных значений параметров вызова. Сетевые отказы и сбои имитируются фильтрацией межсетевого экрана сообщений DNS-сервера, принудительного завершения работы сервера приложений и закрытия сетевых подключений на стороне клиента и Веб-службы.

Трассирование исключительных ситуаций

На рисунке 4 показан фрагмент Java кода, который следует за критическим разделом программы и служит для перехвата возможных исключительных ситуаций, а также отображения результата трассировки стека исключений в случае возникновения ошибок. Такой код реализуется на стороне клиента и веб-сервиса. К результатам трассировки стека исключений относятся ошибки: «несовпадение типов операндов», обнаруженных Веб-службой (на рис.4, 5). Представленная цепочка исключений имеет четыре вложенных вызова (начинающихся с предлога "at").

Для сравнения, трассировка стека исключений при возникновении аналогичной ошибки при взаимодействии с веб-сервисом, реализованным с помощью IBM WSDK, имеет 63 вложенных вызова.

```
    ...
  catch (Exception e) {
 e.printStackTrace();
}
```

Рис. 4. Пример программного кода JAVA для перехвата и трассировки стека исключений

```
java.rmi.ServerException: JAXRPC.TIE.04:
Internal Server Error (JAXRPCTIE01: java.lang.
NumberFormatException: For input string:
"578ER")
at com.sun.xml.rpc.client.dii.BasicCall.
invoke(BasicCall.java:497)
at ai.c1.xai12.wstest.InvoceWS.invoce
(InvoceWS.java:125)
at ai.c1.xai12.wstest.InvoceWS.
invoceByVector(InvoceWS.java:75)
at wstest.Main.main(Main.java:42)
```

Рис. 5. Пример трассирования стека исключений при ошибках («Несовпадение типов операндов») на стороне клиента, использующего реализацию JAX-RPC в среде Sun Microsystems.

6.5. Начальные подходы обеспечения надежности и качества ПТС

В рамках проектов РФФИ №19-01-00206 в ИСП РАН проведен анализ работ по обеспечению надежности технических и программных систем (ПТС):

Лаврищева Е.М., Пакулин Н.В., Рыжов А.Г., Зеленов С.В. Анализ методов оценки надежности оборудования и систем. Труды ИСП РАН.-М.: 2018, Том 30. Вып.3.-с.99-120. Приведены результаты исследований высоконадежных (гарантоспособных) средств и применения их при создании ПТС.

Наиболее весомые практические результаты по обеспечению надежности и качества получены автором по договору с МНИИПА (1980-1987) в рамках ВПК под руководством Липаева В.В. Была разработана технология сборочного программирования и обеспечения высокого качества создаваемых ПТС для авиационной, космической, морской и систем специального назначения. При этом к основным задачам обеспечения качества создаваемых ПТС относились:

– тестирование каждого средства и систем из них на тестовых наборах со сбором сведений об ошибках;

– интеграция методом сборки отдельных связанных системных ресурсов в систему, комплекс и проведения интеграционного тестирования;

– оценивание надежности и качества созданных ПТС для авиационной и космической областей;

– анализ рисков ПТС и отказов аппаратных средств бортовых компьютеров и проведение мероприятий по выпуску качественных продуктов.

Такие задачи выполнялись и в проекте РФФИ 206 и приведены результаты оценки надежности некоторых технических и бортовых систем для космоса и авиации. Полученные научно-технологические результаты основными разработчиками ПТС в проектах ВПК по созданию качественных комплексов ПРОТВА, РУЗА, ЯУЗА и набор технических устройств, ПТС были оценены государственной комиссией СМ СССР и присвоена Государственная премия на тему «Технология разработки бортовых систем».

В настоящее время проблемы обеспечения безотказности, отказоустойчивости информационной безопасности реализуются в проектах ИСП РАН, в том числе и РФФИ 206 для информационных и интеллектуальных ресурсов, рассмотренных выше. Этому способствует главное направление директора ИСП - борьба с авариями, катастрофами и контратаками в рамках военной тематики и свободных коммуникационных средств Интернет.

Статистика инцидентов, аварий и катастроф, вызванных отказами КС появились в научно-техническом центре LAAS (Франция), SRC и CSR (Великобритания), фирма Siemens

(ФРГ), Schneider Electric (Франция), Boing Westinghouse (США) и др. Такие задачи имеют большую значимость в банковских и медицинских системах, а также в электронной коммерции, ГИС и системах мониторинга окружающей среды, приложений распределенного хранения и обработки данных.

На данный момент разработка ПТС основывается на применении сервисных и системных компонентов SOA, SCA, Web-сервисов, Semantic- Web и Client-Server Architecture. Основу технологий КС и ПТС составляет *гарантоспособная надежность*, включающая следующие положения:

– безотказность (reliability);

– готовность (availability);

– достоверность (high confidence);

– приспособленность к обслуживанию или ремонтопригодность (maintainability);

– безопасность (security), конфиденциальность (confidentiality) и целостность (integrity);

– аварийная безопасность (safety).

При этом *отказоустойчивость* (*fault-tolerance*) обеспечивает прогнозирование, сбор данных об отказах и механизмы восстановления работоспособности после отказов и оценку надежности. отдельная составляющая КС.

С 2004 года всемирный Комите IEEE издает журнал *Dependable and SecurityComputing*, в котором есть разделы: Dependable E-services, Internet-commerce и банковские технологии с гарантированной готовностью (High, Continuous, Dependable Availability Technologies).

Современные информационно-вычислительные системы основываются на принципах сервис- архитектур и компонентной интеграции SOA, SCA Web-сервисов.

Теория надежности Веб систем развивается с учетом современных достижений в области элементной базы и технологий математического моделирования систем, теории надежности и конфигурирования сложных ПТС из готовых информационных, сетевых и сервисных ресурсов Интернет (См. статья Лаврищева, Петренко, Позин) и проведения оценки рисков, прогнозирования неисправностей, отказоустойчивости, защиты, безопасности и качества.

Особое внимание в проекте РФФИ 206 уделялось вопросам моделирования сложных Веб-систем и сайтов с помощью *сервисных и сервис-компонентных ресурсов Интернет*, накопленных в Библиотеках и Хранилищах данных Интернет (Статья с Рыжовым А.Г. Подход к моделированию сайтов и систем…). В этой статье используются сервисные ресурсы Интернет для создания Веб-систем и сайтов в среде Интернет и проведен анализ причин и источников возникновения исключительных ситуаций во время сборки ПТС и ОС и восстановления систем после исправления ошибок. К таким механизмам отказоустойчивости относятся:

• backward error recovery (возврат системы к предыдущему безошибочному состоянию);

• forward error recovery (переход системы в одно из безошибочных состояний).

Эти механизмы зависят от логики приложения и использования средств обработки исключительных ситуаций. В связи с тем, что создание систем проводилось в среде IBM WebSphere SDK, Apach Axis, Microsoft .Net и др.), то требовалось проведение анализа механизмов обработки исключений для обеспечения устойчивой работы Веб-систем, задаваемых оператором Link, config, и анализом возникновения ошибок (errors), дефектов (faults, bugs), отказов и сбоев (failures).

Отказ/сбой в ПТС диагностируется, когда ошибочный результат распространяется за пределы интерфейса системы. Дефект в КС или дефект в ПО, допущенный вследствие ошибки программиста, может привести к отказу или сбою Веб-системы. Обычно различают

три группы дефектов КС: дефекты проектирования и разработки, физические неисправности и ошибки взаимодействия.

В заключительном отчете по РФФИ 206 проводились проверки на надежность и качество созданного вариант OS Linux, а также и других систем предметных областей знаний. Опыт показывает, что в Веб-системах могут возникнуть:

- сбои сети и отказы удаленной веб-службы;
- внутренние ошибки и сбои веб-службы;
- ошибки привязки на стороне клиента или сервера.

Указанные ошибки/дефекты являются общими и могут проявляться в любой веб-службе во время её работы. К традиционным сетевым отказам и сбоям относятся недоступность службы DNS или же потери и искажения сетевых пакетов. Кроме того, безотказная работа веб-службы зависит от безотказного функционирования ПО, такого как веб-сервер, сервер приложений и система управления базами данных. Такие сбои могут возникать при принудительном и неожиданном прекращении работы сервера приложений WebSphere, Apache Tomcat и IIS и др.

К ошибкам связывания, сборки сервисов относятся наборы тестов робастности и реализуются на стороне клиента с передачей веб-сервису ошибочных значений параметров вызова операциями link, Config. Трассировка и тестирование создаваемых систем проводились с помощью стандартного инструментария JavaEE или других языков.

В качестве технологий реализации систем с помощью Веб-сервисов и сервера приложений использовались инструменты:

- IBM WSDK + WebSphere;
- Apach Axis + Tomcat;
- Apach Axis + Glassfish;
- Microsoft .Net + IIS, Java EE.

В качестве интегрированной среды разработки могут быть использованы Eclipse IDE, Netbeans IDE (для Java веб-сервисов), а также Microsoft Visual Studio (для.Net Веб-сервисов).

Исследование уязвимостей NVD и CVE Баз данных

Созданная Веб-система из готовых сервисных ресурсов и компонентов может работать с БД Интернет, где размещаются большие данные, к которым идет обращение из других систем и сайтов. В БД могут содержаться сведения об уязвимостях к кибератакам, информационным вторжениям, нарушения доступности к сервисным услугам, целостности и конфиденциальности данных. Требуется провести обнаружение аварийных ситуаций, диагностирование выявленных исключительных ситуаций и уязвимостей в Веб-системе. Причиной возникновения неадекватных ситуаций могут быть:

- сбои в сети и ошибки удаленной службы при выполнении операторов связывания типа link;
- сбои и ошибки внутреннего типа при вычислении некоторой функции отдельного компонента;
- ошибки взаимодействия между клиентом и сервером по передаваемым и обрабатываемым данным.

Ошибки на стороне клиента происходят из-за изменений параметров или передаваемых данных и несоответствия типов описания данных в WSDL и/или IDL.

Ошибки связывания реализуются на стороне клиента при передаче ошибочных значений передаваемых данных операциями и вызова. В случае возникновения ошибок обмениваемых данных проводится трассировка стека исключительных ситуаций на стороне клиента и Веб-системы. Эту трассировку проводит, например, IBM WSDK в процессе взаимодействии с веб-сервисом в виде: catch (Exception e) {e.printStackTrace(); }.

Пример трассировки стека исключений, соответствующего перехвата ошибки («Несовпадение типов операндов») на стороне клиента, использующего реализацию JAX-RPC от Sun Microsystems привеведен на рис.6.

```
java.rmi.ServerException: JAXRPC.TIE.04:
Internal Server Error (JAXRPCTIE01: java.lang.
NumberFormatException: For input string:
"578ER")
at com.sun.xml.rpc.client.dii.BasicCall.
invoke(BasicCall.java:497)
at ai.c1.xai12.wstest.InvoceWS.invoce
(InvoceWS.java:125)
at ai.c1.xai12.wstest.InvoceWS.
invoceByVector(InvoceWS.java:75)
at wstest.Main.main(Main.java:42)
```

Рис. 6. Пример трассировки стека исключений

Информацию об уязвимости получается из общедоступных источников или специализированных БД уязвимости (БДУ), которые предоставляют информацию об уязвимости, возникшей в связи с атаками, нарушениями несанкционированного доступа к защищенным данным БД и др. Поставщиком уязвимостей в БД является общий словарь CVE в виде XML-формата или SGL-дампов (www.osvdl.org). На основе обнаруженных ошибок, исключительных ситуаций и отказов проводится оценка надежности ПО Веб-систем, изготовленных из сервисных ресурсов Интернет средствами Web-служб.

Заключение. По проекту РФФИ 206 -2021

В проекте РФФИ 206-2021 представлен метод моделирования и программирования технических, программных и операционных систем из готовых КПИ, Reuses, а также Веб-систем и сайтов, собираемых из информационных и сервисных ресурсов Интернет. Ресурсы содержатся в Web-services, в библиотеках и хранилищах Интернет и требуют проверки на безопасность и надежность функционирования ПТС в глобальной среде Интернет.Участниками проекта была разработана версия экспериментального ядро OS Linux, широко обсуждаемой в современных предметных областях знаний (медицина, физика, химия и др.) и технических системах в международных конференциях VAMOS (2004-2017) и OS DAY (2014-2021), на которой сделан доклад по технологии сборки экспериментального варианта ядра OS Linux, сделанного по данному проекту. К докладу был интерес многих представителей конференции и одобрение OS DAY-2021. Созданная версия варианта ядра OS Linux проверялась на надежность и безопасность с анализом уязвимостей ресурсов и систем в среде общесистемных средств Интернет (VS.MS, CORBA, BCD, BSU, JAVAEE, GRID, ETICS и др.) к кибер-атакам, нарушению целостности и конфиденциальности данных ПО ОС и прикладных систем с оценкой надежности и качества версии систем и Legasy Systems VAMOS. Отдельные аспекты средств моделирования и программирования вариантных ПТС и ОС представлены со средствами обеспечения надежности, качества и безопасности ресурсов, ОС, ПС и Веб-систем из них.

Описаны стандартные сервисные ресурсы SOA, SCA, SOAP для создания ПТС методом сборки из готовых информационных и сервисных ресурсов для некоторой предметной области знаний. Эти сервисные ресурсы отрабатываются в средах: IBM WSDK + WebSphere, Apach Axis + Tomcat, + Glassfish; – Microsoft .Net + IIS и проверяются на правильность и конфигурируются согласно стандарта IEEE 828 Configuration-2012. Созданные ПТС из готовых ресурсов Интернет проверялись на отказоустойчивости, безопасности, защиту данных и оценку надежности и качества. В качестве примера

разработана версия экспериментального варианта ядра ОС Linux, которая проверена на надежность и качество для применения в предметных областях знаний.

Доклады и статьи по тематике проекта РФФИ №19-01-00206-2021

1. Lavrischeva E.M., Mutilin V.S., Ryzhov A.G. Designing variability models for software, operating systems and their families. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 5, 2017, pp. 93-110. DOI: 10.15514/ISPRAS-2017-29(5)-6.

Abstract. The complexity of existing Legacy systems and the difficulty of amending it led to the development of the new concept of variability of systems specified by a model of the characteristics of FM (Feature Model). In the paper, we discuss the approaches to formal definition of FM and creating on its basis variants of program systems (PS), operating systems (OS) and families of program systems (FPS) for PS and OS. We give methods of manufacturing of PS in the Product Family/Product Lines, the conveyor of K.Czarnecki for assembling of artifacts in the space of problems and solutions, logical-mathematical modeling of PS from the functional and interface objects by Object-Components Method (OCM), extraction of the functional elements from OS kernel to FM for the generation of new variants of the OS. We discuss approaches for formalization of variability of legacy and new PS and their FPS. The new concept of management of variability systems with help OCM is defined. The approach to verify models of the FM, PS, FPS and OS and to configuration of functional and interface objects for obtaining the variants of the resulting product are proposed. We elaborate the characteristics for the testing process of variants of the PS, OS andFPS.

Keywords: variability model; software systems; family of systems; configuration; variant; functional, interface element; requirement; management.

2. Теория графового моделирования сложных систем из модулей для прикладных областей, Лаврищева Е.М. Журнал Высшая школа (рус.) июль (14)2019, ISSN 2409-1677.-с.24-44.

Аннотация. Представлены математические основы графового моделирования прикладных систем. В вершинах графа располагаются функциональные элементы (модули) систем, а дуги задают связи между ними. На основе графа из модулей определяются сложные структуры (программа, комплекс, агрегат и др.) с помощью математических операций (объединения, соединения, разности и др.) и они представляются матрицей смежности и достижимости. Приводятся формальное связывание, сборка (link) модулей и их интерфейсов с операциями обмена данными между ними модулями и преобразования неэквивалентных данных в среде ОС ЕС IBM (1976-1987). Описан формальный переход от модулей к объектам и компонентам согласно парадигмы ОКМ, а также появления стандартной операции config (IEEE 828-2012, Configuration Managment) для получения выходного конфигурационного файла прикладной системы. Предложен теоретический аппарат формального преобразования неэквивалентных данных из класса общих типов данных – генерированных, неструктурных и агрегатных стандарта ISO/IEC 11404-2007, GDT к более простым фундаментальным. Сделан вывод о перспективном развитии теории графов и механизмов сборки применительно к прикладным областям (медицина, биология, генетика и др.).

Ключевые слова: теория графов; матрица смежности, достижимости; математические операции; объединение, проекция, следование; конфигурационная сборка; преобразование данных.

Graph theory modeling complex systems modules for the application areas

Annotation.The basics of graph modeling of applied systems are presented. In the vertices of graph are located the functional elements (modules) of the systems, and the arcs define the connections between them. On the basis of graph from modules are represented the program structures by mathematical operations (unions, connections, differences, etc.) and mathematical matrix of the adjacency and reach ability. Given the characteristics of graph structures, complexes,

units, and systems created from the modules of the graph. The method of assembling the system on the graph of multilingual modules in the first programming languages – LP (Algol-60, Fortran, Cobol, PL/1, Smalltalk, etc.) for the IBM OS (1976-1985) is given. A scientific theory is determined by the OKM associate objects graph using objects of the interface containing operations for data exchange among objects of the graph. Transition to components, services and modeling of complex program structures on the graph with the advanced operations of assembly, association (make, config, assembling, weaver, etc.), functions of transformation of the exchanged data according to the ISO/IEC 11404-2007 GDT standard in the class of the applied systems working in the environment of Big Data, Cloud Computing Internet and quality assurance of this systems are shown.

Keywords: graph theory; adjacency matrix, reach ability; mathematical operations; assembling; configuration; nostructured big data; transformation.

3. Технология сборки интеллектуальных и информационных ресурсов Интернет.- Е.М. Лаврищева, А.К. Петренко. XXI Всероссийская научная конференция (23-28 сентября 2019 г., г. Новороссийск. — М.: ИПМ им. М.В.Келдыша, 2019. — С. 469-488. — http://keldysh.ru/abrau/2019/theses/93.pdf, doi:10.20948/abrau-2019-93. Некоторые аспекты технологии сборки отражены и в презентации доклада на конференции ИСП РАН-2019 (http://0x1.tv/20191206AB).

Аннотация. Предлагается технология конфигурационной сборки интеллектуальных (Reuses) и информационных (data) ресурсов, которые разрабатываются для ТПС и накапливаются в библиотеках и хранилищах Интернет, в прикладные web-системы разного назначения. Представлены средства описания ресурсов, их верификация, тестирование, сборка и обеспечение безопасности и качества их взаимодействия. Обмениваемые между ресурсами данные трансформируются к требуемым форматам данных серверной среды Интернет с учетом стандарта ISO/IEC 11404 GDT и в соответствии с исходными данными компонентов повторного использования (КПИ), готовых сервисных и системных ресурсов Интернет. Приведена технология проверки правильности и оценивания показателей качества вариантов Веб-систем.

Ключевые слова: компонент, сервис, ресурс, система, веб-система, брокер сервисов, надежность, качество.

Technology of Assembly of intellectual and information resources Semantic Web Internet. E.M. Lavrischeva, A.K. Petrenko http://keldysh.ru/abrau/2019/theses/93.pdf, doi:10.20948/abrau-2019-93

Annotation. Technology offers intelligent configuration of the Assembly (Reuses) and information (data) resources developed and accumulated in the libraries and depositories of the Internet, applied web-based systems. Means of description, verification, testing, Assembly and security of interaction of resources are presented. The transmitted data is generated to the required data formats of the server environment taking into account the ISO/IEC 11404 GDT standard and converted to the source data of the client. Proposed configuration Assembly variation of the system of CPI and assessment of the required indicators of the quality of the resulting web system.

Keywords: component, service, resource, system, web-system, service broker, reliability, quality, Internet.

4. E.M. Lavrischeva, A.K. Petrenko and B.A.Pozin. Technology of Assembly of intellectual and information resources Internet.-CEUR-WS.org/2019.-vol-2543/paper-22.pdf.-27p.

Annotation. The technology of Assembly of intellectual (Reuses) and information (data) resources which are developed and saved up in libraries and storages of the Internet, in applied web-systems of different function is offered. The intellectual tasks of subject areas of knowledge in the Semantic Web in mind service, software and system resources used in creating Web systems are presented. Methods of assembling resources (link, make, assemble, weaver, kconfig) are considered

for modern environments. The problem of data exchange and generation to the formats of the global network environment according to ISO/IEC 11404 GDT is defined. The concept of a network resource collector is proposed as a generalization of ways to assemble resources with functional security, resource protection and quality assessment of Web-systems. Perspective paradigms of programming of different areas of knowledge are defined.

Keywords: resource, service, web system, configuration, functionality, security, reliability, quality.

5. Методы оценки надежности программных и технических систем. Е.М. Лаврищева, С.В. Зеленов, Н.В. Пакулин, А.Г. Рыжов, С.В.Морозов, О.А. Тарлапан. Труды ИСП РАН, том 31, вып. 5, 2019 г., стр. 95-108. DOI: 10.15514/ISPRAS-2019-31(5)-7 https://videonauka.ru/stati/30-metodika-prepodavaniya-tekhnicheskikh-distsiplin/238-analiz-metodov-otsenki-nadezhnosti-oborudovaniya-i-sistem-i-praktika-primeneniya-etikh-metodov

Аннотация. Определяются основные методы обеспечения и оценки надежности и безопасности программно-технических систем (ПТС) на процессах жизненного цикла (ЖЦ) и сбором сведений о возникающих в них ошибках, дефектах и отказах для последующих изменений. Рассмотрена стандартная модель надежности и дана характеристика базовых показателей, среди которых показатель надежности, функциональность и безопасность составляют основу измерения надежности. Приведена классификация моделей надежности, дана характеристика моделей оценочного типов, используемых при проверке показателей надежности компонентов ПТС. Показаны экспериментальные результаты применения оценочных моделей надежности к разным размерам программных компонентов ПТС и представлена оценка результатов измерения показателя надежности на этих компонентах с учетом плотности дефектов, интенсивности отказов и восстанавливаемости. Отмечается важность обеспечения надежности и безопасности систем в рамках новых стандартов (dependability and safety) в рамках интеллектуальных систем и Интернет вещей.

* Выполняется по проекту РФФИ 19-01-00206.

Ключевые слова: надежность, ошибка, дефект, отказ, дефект, безопасность, тестирование, надежность, риск, умные компьютеры.

6. Lavrischeva E.M., Zelenov S.V., Pakulin N.V. Methods for assessing the reliability of software and hardware systems. *Proceedings of the Institute for System Programming of the RAS (Proceedings of ISP RAS)*. 2019;31(5):95-108. (In Russ.) https://doi.org/10.15514/ISPRAS-2019-31(5)-7. http://0x1.tv/20180517F 9(in rus)

Annotation. The basic methods of ensuring and assessing the reliability and safety of software and hardware systems (PTS) on the life cycle processes (LC) and collecting information about errors, defects and failures arising in them for subsequent changes are determined. The standard model of reliability is considered and the characteristic of basic indicators among which the indicator of reliability, functionality and safety make a basis of measurement of reliability is given. Classification of models of reliability is given, the characteristic of models of the estimated types used at check of indicators of reliability of components of PTS is given. The experimental results of the application of reliability evaluation models to different sizes of software components of PTS are shown and the evaluation of the measurement results of the reliability index on these components taking into account the density of defects, failure rate and recoverability is presented. The importance of ensuring the reliability and safety of systems within the new standards (dependability and safety) in intellectual systems and Internet Thinks.

* Performed by the RFBR project 19-01-00206.

Keywords: reliability, error, defect, failure, defect, security, fault, completeness, testing, reliability, risks, smart computers

7. Лаврищева Е.М., Мутилин В.С., Козин С.В., Рыжов А.Г. Моделирование прикладных и информационных систем из готовых сервисных ресурсов Интернет. ***Труды Института системного программирования РАН*****. 2019;31(1):7-24. https://doi.org/10.15514/ISPRAS-2019-31(1)-1**

Аннотация. Рассматривается подход к созданию сложных (информационных и прикладных) систем из готовых ресурсов (модулей, компонентов, КПИ, сервисов, reuses и др.). В основе подхода создания систем, веб–систем лежит компонентная модель (КМ), включающая функциональные, системные, сервисные и интерфейсные готовые ресурсы (ГОР), и алгебра компонентов для выполнения разных операций над ГОР. Прикладные функции компонентов, КПИ описываются в языках программирования (ЯП), интерфейсы в языках IDL и WSDL, а сервисные компоненты создаются в SOA, SCA IBMSphere или выбираются из Интернет библиотек, как готовые. Исходные компоненты верифицируются сохранятся в репозитории ГОР и интефейсов. Представлен метод сборки ГОР: Link в среде IBM, MS.VS; make в BSD, config в JavaEE; стандарт IEEE 828–96–2012 (Configuration), как итог всех сборок. ГОР тестируются на множестве тестовых данных, проверяется их правильность и надежность работы. На готовый сконфигурированный продукт формируется сертификат качества на основе стандартов ISO/IEC 9000 (1–4) quality. Даны перспективы развития средств обеспечения безопасности и качества веб–систем.

Ключевые слова: компонент; сервис; ресурс; система; веб–система; сборка; надежность; качество.

8. Лаврищева Е.М., Зеленов С.В. Модельный подход к обеспечению безопасности и надежности Web-сервисов. ***Труды Института системного программирования РАН*****. 2020;32(5):153-166. https://doi.org/10.15514/ISPRAS-2020-32(5)-12**

Аннотация. Дается анализ проблематики надежности и безопасности в мировой практике и в нашей стране. Рассмотрены аспекты моделирования программно-технических систем (ПТС) из сервисных ресурсов и готовых КПИ с обеспечением надежности и безопасности. Приводятся сформировавшиеся базовые и теоретические основы проблемы моделирования, опыта использования современныхсервисных средств SOA, SCA, SOAP в ПТС и Web-системах с обеспечением их надежности и безопасности в Интернет. Отмечается, что ПТС и Web-системы создаются методом сборки всовременных средах: IBM WSDK + WebSphere, Apache Axis + Tomcat; Microsoft .Net + IIS и др. Должны проводиться верификация и тестирование систем для поиска ошибок, возникающих приисключительных ситуациях (кибератаках, запрещенных доступах к БД и др.). Описаны методы анализа таких ситуаций и применения методов надежности и безопасности для обеспечения устойчивой ибезотказной работы сервисных компонентов ПТС в информационной среде Интернет.

*** Выполняется по проекту РФФИ 19-01-00206.**

Ключевые слова: безопасность; надежность; качество; сервисный ресурс; сервисные службы; Web-системы; анализ уязвимости; ошибочные ситуации; оценка качества.

9. Lavrischeva E.M., Zelenov S.V. Model-Based Approach to Ensuring Reliability and Security of Web-services. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 5, 2020, pp. 153-166 Model-Based Approach to Ensuring Reliability and Security of Web-services

Abstract. In the paper, we analyze problems of reliability and security in the world practice and in Russia. We consider aspects of modeling software/hardware systems from service resources and ready-made reuses with ensuring reliability and security. We present the formed basic and theoretical foundations of the modeling problem, the experience of using modern service tools SOA, SCA, SOAP in software/hardware systems and Web systems to ensure their reliability and security on the Internet. We note that software/hardware system sand Web systems are created by the assembly build method in modern environments: IBM WSDK +WebSphere, Apache Axis + Tomcat; Microsoft .Net + IIS, etc. Verification and testing of systems should be conducted for

searching of errors that occur in exceptional cases (cyber-attacks, forbidden access to the database, etc.). We describe methods for analyzing such situations and applying reliability and security methods to ensure stable and trouble-free operation of software/hardware systems service components in the Internet in formation environment.

Keywords: security; reliability; quality; service resource; service services; Web systems; vulnerability, analysis; erroneous situations; quality assessment .

10. Лаврищева Е.М., Петров И.Б. Моделирование технических и математических задач прикладных областей знаний на ЭВМ. *Труды Института системного программирования РАН*. 2020;32(6):167-182. https://doi.org/10.15514/ISPRAS-2020-32(6)-13.

Аннотация. Рассмотрено моделирование технических задач и задач прикладной математики, их алгоритмизация и программирование. Дается характеристика численного моделирования технических задач и прикладной математики: физико-технических экспериментов, энергетических, баллистических и сейсмических методов И.В Курчатова, начиная с математических методов 17-20 вв., первых ЭВМ и компьютеров. Дан анализ первых технических задач и задач прикладной математики, их моделирование, алгоритмизация и программирование с помощью граф-схемного языка А.А.Ляпунова, адресного языка и языков программирования. Представлены численные методы, реализованные под руководством А.А. Дородницына, А.А.Самарского, О.М. Белоцерковского и других ученых на современных суперкомпьютерах. Приведены примеры математического моделирования биологической задачи лечения глаз и предмета «Вычислительной геометрии» в Интернет.

Ключевые слова: математика, модель, моделирование, алгоритм, граф, области знаний, биология, геометрия.

Работа поддержана грантом РФФИ № 19-01-00206

11. Е.М. Лаврищева, А.К.Петренко, С.В.Зеленов, С.В.Козин. Технология сборочного создания экспериментального варианта ядра OS Linux с обеспечением качества для применения в прикладных и технических системах .- Системный администратор, №11 (228), 2021.

Аннотация. Предлагается технология сборки экспериментального варианта ядра OS Linus с надежным и безопасным функционированием в технических устройствах и прикладных системах. Разработка варианта ядра проводилась в рамках проектов РФФИ №352 и №206 в период 2016-2021гг. Представлена технология сборочного создания вариантов OS Linus и обеспечения безопасности и надежности функционирования варианта ядра в системах и устройствах. Описаны операции config, make конфигурационной сборки ядра ОС IEEE 828 Configuration - 1996, 2012, средства верификации, тестирования, безопасности, защиты и оценки надежности отдельных элементов и варианта ядра OS Linus. Приведены примеры методов обеспечения надежности, защиты и безопасности на некоторых технических средствах с членами названных проектов РФФИ.

Ключевые слова: сборка, конфигурация, операции сборки, link, config, make, assembly, функциональные элементы OS, модули, объекты, компоненты, библиотеки.

12. Онтология моделирования домена жц стандарта iso/iec 12207. Лаврищева Е.М.

Аннотация. Рассмотрен онтологический подход к представлению модели жизненного цикла стандарта ISO/IEC 12207-2007, включая описание основных, организационных и процессов поддержки. Для процесса тестирования приведено онтологическое описание понятий, концептов и задач в терминах системы Protégé, результатом которой является общепринятый язык XML, по которому можно получить машинную программу. Эти онтологические описания процессов жизненного цикла являются базисом их автоматизации на любой платформе компьютера, особенно это касается процесса тестирования по

программе, которой проведено проверка программ в языке Ruby.Описание процессов и реализации представлены в инструментально-технологическом комплексе на веб-сайте http://7dragons.ru/ru сайта ИСП РАН.

Ключевые слова: онтология, модель, понятия предметной области, процессы, основные, поддержки, организационные, тестирование, реализация, выходной язык XML.

13. Лаврищева Е.М., Петров И.Б., Петренко А.К. Парадигмы моделирования и программирования задач и данных предметных областей знаний. DirectMedia. Под редакцией: Аветисян А.И., Баксанского О.Е., Горбунов-Посадова М.М.Москва- Берлин, 2021.- 495с. ISBN 978-5-4499-1889-5.

Монография посвящена парадигмам моделирования и программирования современных предметных областей знаний (физика, математика, биология, химия, медицина и др.), начиная с появления первых аналитических машин и ЭВМ для численного решения технических задач и задач предметных областей знаний. Описываются сформировавшиеся подходы к моделированию математических задач на первых ЭВМ (А. А. Самарского, Е. Л. Ющенко, В. М. Глушкова, О. М. Белоцерковского, О. М. Поспелова, И. Б. Петрова и дискретных программно-технических систем (ПТС) с участием В. М. Глушкова, В. Н. Коваля и др. Дана характеристика зарубежных парадигм программирования задач разных предметных областей знаний (ООР, VDM, UML, Z, RAISE, RSL, Agaile и др.), а также отечественных парадигм модульного, синтезирующего, композиционного, аспектного, агентного программирования и инженерии приложений и доменов (DSL, OWL, OCM и др.).

Уровень и роль работ по проекту РФФИ №19-01-00206-2021

Созданная теория моделировании и программирования сложных программных, операционных и Веб-систем из готовых ресурсов - ГОР (reuses, модулей, объектов, компонентов, сервисов и др.) с оценкой надежности и качества отдельных программных элементов и ПТС из них находится на уровне Европейских и международных работ по проблематике VAMOS, Reusebility (2007-2020). Поднят уровень интеллектуализации и информатизации объектов предметных областей знаний, используемых для создания Веб-систем и сайтов применительно к Big Data and Cloud Computing с использованием стандартных операций сборки: link/make/config/make/cmake/build/waver, функционирующих в системах IBMSphere, .Net, BSD, Grid, JAVAEE, ETICS и основанных на проведении конфигурационной структуры по стандарту ISO/IEC 828-92-2012 -configuration) для формирования отдельных вариантов ПТС и ОС. Сборка определена с общих позиций и предлагается как заявка на патент. Конфигурационная структура верифицируется, тестируется и оценивается безопасность, надежность и качество. Многие результаты исследований и реализаций по данного проекту опубликованы и доступны для применения в конкретной деятельности по проектированию программных и информационных систем. Разработана по заявке Интуит.ру 2 версия книги –Лаврищева Е.М., Грищенко В.Н. «Связь ращноязыковых модулей Ос ЕС ЭВМ.- М.: Финансы и статистика .-1980.-137с.

Аннотация. Книга посвящена методам объединения, сборки модульных элементов, записанных в языках программирования (ЯП) ОС ЕС и интеллектуальных ресурсов в WWW3C. Цель работы – научить программистов методу сборки модулей в разных ЯП для создания сложных программных систем и комплексов путем их конструирования из готовых ресурсов с использованием разработанной библиотеки межъязыкового интерфейса (БМИ) и развитием ее BigData. От пользователей требуется знание ЯП высокого уровня для описания функций прикладных систем модульными элементами в современных ЯП в средах IBM 360 (ОС ЕС), VS.Net, Java.Net, Corba и др. Приводятся функции БМИ для преобразования передаваемых простых, сложных, неструктурированных данных от одного ресурса к другому с учетом современных общих типов данных GDT (General Data Types) стандарта ISO/IEC 11404-2012 и метода сборки ресурсов, записанных в современных ЯП (Algol, Cobol, Smalltalk, Fortran, Java, Ruby, Basic, C, C++ и др.) в средах Интернет (IBM, VS.Net, Corba, Grid, Etic, BSD, Oracle BD и др.) для получения систем в медицине, биологии, физических экспериментах, нано приборах и роботах.

В книге представлены разделы для проведения сборки и обработки ОС, ресурсов, работающих с новыми неструктурными данными и преобразования к формату ЭВМ; средств онтологии ЖЦ ПТС согласно стандарту ISO/IEC 12207-2002, 2007 и вычислительной геометрии. Lfyj jписан экспериментального варианта OS Linux, выполненого с участием студентов МФТИ (см.презентацию).

Публикации по тематике проектов РФФИ на конференциях:

– «Научный сервис - 2019 Доклад Моделирование сложных систем - http://0x1.tv/20191206AB);

-Актуальные проблемы систем и программного обеспечения» - ASPSSE-2019 http://ceur-ws.org/Vol-2514/ ; -p.235-247. ISPRAS OPEN -2019 http://0x1.tv/20191206AB);

http://0x1.tv/20180517F; http://0x1.tv/20181122AF.

– XXI Всероссийской научной конференции (23-28 сентября 2019 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2019. — С. 469-488. — URL: http://keldysh.ru/abrau/2019/theses/93.pdf doi:10.20948/abrau-2019-93. Размещена презентация к докладу;

– Научный журнал «Austrua Science", №28/2019, с.12-29; Теории графового моделирования систем в Международных журналах «Computer and Information Sciences» – vol.12. №4 2019 и TMLAI Vol 12 No 6 Dec 2019 – "The Theory Graph Modeling and Programming Paradigms of Systems from Modules to the Application Areas";

Доклады и публикации являются перспективными в области теории программирования систем и ТПС. Они вызывают интерес у специалистов разных стран мирового сообщества по видео в Интернете

http://0x1.tv/20191206AB;

http://0x1.tv/20180517F;

http://0x1.tv/20181122AF.

https://videonauka.ru/stati/30-metodika-prepodavaniya-tekhnicheskikh-distsiplin/238-analiz-metodov-otsenki-nadezhnosti-oborudovaniya-i-sistem-i-praktika-primeneniya-etikh-metodov.

– Лаврищева Е.М. Программная инженерия и технология программирования сложных изменяемых систем. Учебник. 2-издание. Москва – Юрайт, 2018.- 431с. (www.urait.bibloo – 0nline.ru).

– Е.М.Лаврищева, И.Б.Петров. Теория моделирования технических и математических задач предметных областей знаний. - Евроазиатское Научное Объединение. 2021.№ 1- (1).- с.35-43.

Соглашения: https://drive.google.com/open?id=1Sind_wVjT5v5lP4HiR65oViDAqc0I9GG

Тексты статей: https://drive.google.com/open?id=1YF506xO2Zp05phnYbzxIxM_iyDXg1AGg

Публикации в зарубежных журналах по тематике технологии и инженерии программирования за период проектов РФФИ 352 – 2016-2018 и 206- 2019-2021.

1. E.M.Lavrischeva. Science of computer programs and systems in XX-XXI centuries: Past, Present, Future.-European Journal Mathematics and Computer Science.-2016. Vol.5 N 1.- p.67-87. 2018.-ISSN 2059-9951. www.idpublications.org.

2. E.M. Lavrischeva. Scientific Basis of System Programming.- Journal of Software Engineering and Applications (JSEA), Vol. 11 No. 8 of August issue, 2018.-N 11.-p.408-434, ISSN online 1945-3124, ISSN Print 1945-3116.

3. E. Lavrischeva. Software Engineering: New disciplines and e-learning them for development of Applied Systems. Europe Journal of Engineering and Technology, 2015.-N3. p. 36-67. ISSN2056- 5860, www.idpublications.org.

4. Lavrishcheva E. M. The theory of object-component modeling of software systems. Preprint of the RAS, No. 29 (48 p.). 2016.- ISBN 078-5-91474-025-9.13.

5. Lavrishcheva E. M. Ryzhov A. G. Application theory of general data types of the standard ISO/IEC 11404 GDT in relation to Big Data. - The conference " Actual problems in science and ways of their development",27december 2016, http://euroasia-science.ru/ p. 99-110.

6. Лаврищева Е.М. Рыжов А.Г. Применение теория общих типов данных стандарта ISO/IEC 11404 GDT применительно к Big Data.- The conference "Actual problems in science and ways their development", 27 december 2016, http://euroasia- science.ru/ p.99-110.

7. Лаврищева Е.М., Рыжов А.Г. Подход к моделированию систем и сайтов из готовых ресурсов. Научный сервис в сети Интернет: труды XX Всероссийской научной конференции (17-22 сентября 2018 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2018. — С. 321-345. —URL: ttp://keldysh.ru/abrau/2018/theses/50.pdf doi:10.20948/abrau-2018-50

8. Лаврищева Е.М., Слабоспитская О.А. Коллекция CASE-Tools для сборки вариабельных систем из готовых программных ресурсов.- Научно-практическая конференция «Технология разработки информационных систем». ТРИС Таганрог 15, 6-12 сентября 2015.-Геленджик. Изд-во ЮФУ Таганрог, 2015.- с.147-179.

9. Лаврищева Е.М. Теория графового моделирования сложных систем из модульных элементов для прикладных областей.- Международный научный журнал «Austria-science»

2015. 1 часть №28/201. с.12-29. Universitätsstraße 16-18, 6020 Insbruck, Австрия. Сайт: http://austria-science.info

10. Лаврищева Е.М. Теория графового моделирования сложных систем из модулей для прикладных областей.- Научно-практический журнал, Высшая школа. №14, 2019.с.27-46: www.ran-nauka.ru © ООО «Инфинити», 2019. ISSN 2409-1677.

11. Ekaterina Lavrischeva. The theory graph modeling systems from quality modules of the application areas, Journal "Actual Problems of System and Software Engineering, 2019.- http://ceur-ws.org/ceur-author-agreement-ccby.pdf, Paper: 135.- p.235-247.

12. Ekaterina Lavrischeva. The theory graph modeling systems from quality modules of the application areas, Journal "Actual Problems of System and Software Engineering, 2019.- http://ceur-ws.org/ceur-author-agreement-ccby.pdf,Paper: 135..-p.235-247.22.

13. Е.М. Лаврищева, И.Б. Петров. Теория моделирования технических и математических задач предметных областей знаний. Евроазиатское Научное Объединение. 2021.№ 1- (1).-с.35-49.

14. Е.М.Лаврищева, И.Б.Петров. Моделирование технических и математических задач прикладных областей знаний на ЭВМ. Труды ИСП РАН 2020, Том 32 № 6 с.167-182.

15. Лаврищева Е.М. Теория графового моделирования сложных систем из модульных элементов для прикладных областей .- Austria-science»1 часть №28/2019.- p.12-30. http://austria-science.info

Базовые публикации в 2021

1. Лаврищева Е.М., Петренко А.К., Зеленов С.В., Козин С.В., Технология сборочного создания экспериментального варианта ядра OS Linux с обеспечением качества для применения в прикладных и технических системах.- Наука и технологии.-Системный администратор.- 2021. Ноябрь.- с.80-89.

2. Е.М. Лаврищева, И.Б. Петров. Теория моделирования технических и математических задач предметных областей знаний. Евроазиатское научное объединение. 2021. № 1 (1). С. 35–43.

3. Lavrischeva E. M. Petrenko A. K. Kozin V. P. "Technology of assembly creation of an experimental version OS Linux kernels with quality assurance for applied and subject areas of knowledge".EuroazianScience Assosiation? www.ESA.ru. november-2021.

5.Лаврищева Е.М., Гртщенко В.Н. Метод сборки модулей, объектов, компонентов в языках программирования.- ИСП РАН.-Редакция 2.2022.с.174.

4. Монография - Е.М.Лаврищева, И. Б. Петров, А. К. Петренко. Парадигмы моделирования и программирования задач предметных областей знаний: монография / DirectMeduio. Москва- Берлин, 2021.- 495с.

Монография посвящена парадигмам моделирования и программирования современных предметных областей знаний (физика, математика, биология, химия, медицина и др.), начиная с появления первых аналитических машин и ЭВМ для численного решения технических задач и задач предметных областей знаний. Описываются сформировавшиеся подходы к моделированию математических задач на первых ЭВМ (А. А. Самарского, Е. Л. Ющенко, В. М. Глушкова, О. М. Белоцерковского, О. М. Поспелова, И. Б. Петрова и дискретных программно-технических систем (ПТС) с участием В. М. Глушкова, В. Н. Коваля и др. Дана характеристика зарубежных парадигм программирования задач разных предметных областей знаний (ООР, VDM, UML, Z, RAISE, RSL, Agaile и др.), а также отечественных парадигм модульного, синтезирующего, композиционного, аспектного, агентного программирования и инженерии приложений и доменов (DSL, OWL, ОСМ и др.).

В период появления первых ЭВМ и языков программирования (ЯП) — Algol-58, 60; Fortran, Cobol, PL/1, Snobol, Lisp и др. описана программирующая программа трансляторов с ЯП (ТА1-ТА5) для перевода исходных схем программ в ЯП в код первой ЭВМ. Обсуждаются сформировавшиеся методы трансляции, как средств автоматизации программирования. Показано развитие схем к моделированию структур алгоритмов графовым и сетевым нейронным способом. Описана сборка разнородных модулей в ЯП, работающих с разными данными из класса фундаментальных типов данных (ФДТ) и межъязыкового преобразования совместных данных в среде IBM-360 (1976–1982).

Описана парадигма графового математического моделирования задач предметных областей знаний с применением математических операций и аппарата матриц смежности, инцидентности и средств обеспечения правильности (Model Checking, V&V, Testing функций задач), обеспечения безопасности и надежности функционирования задач в Интернет. Представлено математическое моделирование задач прикладной математики средствами компьютеров (Петров И. Б. МФТИ) для биологии и вычислительной геометрии.

Определены задачи интеллектуализации и информатизации знаний в рамках Европейского проекта E-science, Grid и Semantic Web глобальной сети Интернет. Приведены парадигмы моделирования космических задач, вычислительной геометрии и прикладного домена ЖЦ с использованием интеллектуальных методов Data Mining и др.

Описана парадигма моделирования сложных вариабельных прикладных систем с использованием моделей систем Msys, модели характеристических свойств функций систем MF (Model Feature) и моделей знаний Mining, сформовавшихся в мировом сообществе в рамках промышленных технологий в разных странах. Приведен вариант ядра OS Linux с использованием Variability Mining на примере магистерских работ МФТИ и операций конфигурационной сборки в ОС Интернет, выполненных участниками в рамках проекта РФФИ № 16-01-00352.

Описан общий сборщик интеллектуальных, информационных, программных и технических ресурсов Интернет в ЯП пятого поколения (C++, Java, Python, Ruby и др.) в общесистемных средах (JavaEE, IBMSphere, OS Linux, Intel, Grid, Visial Studio и др.) в Веб-системы и сайты глобальной сети. Приведены алгоритмы преобразования общих типов данных (GDT) ISO/IEC 11404 GDT и неструктурированных типов данных Big Data, Cloud Computing Semantic Web Интернет. Приводятся парадигмы DSL, инженерии SE, фабрик Чернецкого и Бея, основанных на методе конвейерной сборки интеллектуальных ресурсов Интернет для ПТС.

Приводится характеристика зарубежных и отечественных парадигм моделирования, методов синтеза частиц нано и ДНК в МФТИ; Нейро учебников для обучения студентов университетов (МФТИ, МГУ и др.) современным парадигмам моделирования, инженерии и программирования ПТС для разных предметных областей знаний.

После конференции при правительтве по онтологии ЖЦ в 2018г. был выдан программмный сертификат Лаврищевой Е.М.

Свидетельство

«Государственная регистрация программы для ЭВМ № 2018615442 Государственная служба по интеллектуальной собственности. от 28.03.2018. Лаврищева Е.М. “Реализация метода онтологического моделирования домена ЖЦ стандарта ISO/IEC 12207“.

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2018615442

«Реализация метода онтологического моделирования домена ЖЦ стандарта ISO/IEC 12207»

Правообладатель: *Федеральное государственное бюджетное учреждение науки Институт системного программирования им. В.П. Иванникова Российской академии наук (RU)*

Автор: *Лаврищева Екатерина Михайловна (RU)*

Заявка № 2018612585
Дата поступления 20 марта 2018 г.
Дата государственной регистрации в Реестре программ для ЭВМ *08 мая 2018 г.*

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

Доклады по тематике Проектов РФФИ 352 и 206

1. Доклад- Лаврищева Е.М.Технология сборочного создания экспериментального варианта ядра OS Linux с обеспечением качества для применения в прикладных и технических системах//Петренко А.К., Зеленов С.В..- Восьмая научно-практичекая конференция OS DAY "Российские аппаратные платформы и операционнные системы".- Москва 14 октября 2021. -15-00

2. Доклад - Лаврищева Е.М. Научный отчет по теории моделирования и программирования качественных сложных систем по проектам РФФИ 352 и 206.- Открытая конференции ИСП РАН им. В.П. Иванникова. Секция 2 «Технологии анализа, моделирования и трансформации программ".- 2 декабря,2021-15-30.

Публикации по Проекту РФФИ 206-2019-2021 на анг.

1. E. M. Lavrischeva. The theory graph modeling systems from quality modules of the application areas. http://ceur-ws.org/Vol-2514/ 2019.-p.235-247.

2. E.M. Lavrischeva, A.K. Petrenko and B.A.Pozin.Technology of Assembly of intellectual and information resources Internet, E-library, SEUR-2019.

3. E.M. Lavrischeva, A.K. Petrenko .Technology of Assembly of intellectual and information resources Semantic Web Internet/ http://keldysh.ru/abrau/2019/theses/93.pdf doi:10.20948/abrau-2019-93.

4. E. M. Lavrischeva , Graph theory modeling complex systems module elements for the application areas.- Scientific Journal «Austrua Science” info, №28/2019, p.12-29. http://austrua science/info/

5.Lavrischeva E.M. Zelenov S.V., Pakulin N.V. Ryzhov A.G. Morozov S.V. Tarlapan O.A. Proceeding of the ISP RAS, Volume 31, ussue. 5, 2019, pp. 95-108. DOI: 10.15514/ISPRAS-2019-31(5)-7/, http://0x1.tv/20180517F; http://0x1.tv/20181122AF.

5. E.M. Lavrischeva.The Theory Graph Modeling and Programming Paradigm Systems from Modules to the Application Areas – DOI: https://doi.org/10.14738/tmlai.74.6782, Vol 12 No 6 (2019): Transactions on Machine Learning and Artificial Intelligence.

8. E. M. Lavrischeva. The Theory Graph Modeling and Programming Paradigms of Systems from Modules to the Application Areas.- Journal. Computer and Information Sciences – vol.12, №4. 2019 (ISSN: 1913- 8989, E-ISSN: 1913-8997)

9. Lavrischeva E.M. Graph theory modeling complex systems modules for the application areas. http://ran-nauka.ru/wp-content/uploads/2014/09/Nauka-14-2019.pdf.-p.24-44.10.

Публикации по проекту РФФИ 19-01-206-2021

1. Лаврищева Е.М., Петренко А.К. Технология сборки интеллектуальных и информационных ресурсов Интернет // Научный сервис в сети Интернет: труды XXI Всероссийской научной конференции (23-28 сентября 2019 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2019. — С. 469-488. — URL: http://keldysh.ru/abrau/2019/theses/93.pdf doi:10.20948/abrau-2019-93

2. Лаврищева Е.М. Теория графового моделирования сложных систем из модулей для прикладных областей.- Научно-практический журнал, Высшая школа. №14/ 2019.-.с.27-46: www.ran-nauka.ru © ООО «Инфинити», 2019. ISSN 2409-1677.

3. Лаврищева Е.М. Тория графового моделирования систем из модульных элементов для прикладных областей знаний.- Austria-science»1 часть №28/2019.- p.12-30. http://austria-science.info.

4. E.M. Lavrischeva, A.K. Petrenko and B.A.Pozin. Technology of Assembly of Intellectual and Information Resources Internet.-CEUR-WS-2019.org/vol-2543/ipaper-22.pdf.-27p.

5. Лаврищева Е.М. Тория графового моделирования систем из модульных элементов для прикладных областей знаний.- Austria-science»1 часть №28/2019.- p.12-30. http://austria-science.info.

6. Лаврищева Е.М., Зеленов С.В., Рыжов А.Г.. Теория моделирование программно-техническихсистем с обеспечением безопасности и надежности в среде Web-services Интернет. Труды Института системного программирования РАН, Том 29.№

7. Екатерина Михайловна Лаврищева, Сергей Вадимович Зеленов. Модельный подход к обеспечению безопасности и надежности Web-сервисов. Труды ИСП РАН 2020, Том 32 № 5. С.151-163.

8. Е.М. Лаврищева, И.Б. Петров. Моделирование технических и математических задач прикладных областей знаний на ЭВМ. Труды ИСП РАН 2020, Том 32 № 6.- с.167-182.

9. Е.М.Лаврищева, И.Б.Петров. Теория моделирования технических и математических задач предметных областей знаний. Евроазиатское Научное Объединение. 2021.№ 1- (1).- с.35-43.

10. Екатерина Михайловна Лаврищева, Сергей Вадимович Зеленов. Модельный подход к обеспечению безопасности и надежности Web-сервисов. Труды ИСП РАН 2020, Том 32 № 5. С.151-163.

11. Е.М.Лаврищева, И.Б.Петров. Моделирование технических и математических задач прикладных областей знаний на ЭВМ. Труды ИСП РАН 2020, Том 32 № 6.- с.167-182.

12. Е.М.Лаврищева, И.Б.Петров.Теория моделирования технических и математических задач предметных областей знаний. Евроазиатское Научное Объединение. 2021.№ 1- (1).-с.35-43.

13. Лаврищева Е.М., Петренко А.К., Зеленов С.В., Козин С.В. Технология сборочного создания экспериментального варианта ядра OS Linux с обеспечением качества и безопасности для применения в прикладных и технических системах.- Наука и технологии.-Системный администратор.- 2021. Ноябрь.- с.80-89.

14. Lavrischeva E. M. Petrenko A. K. Kozin V. P. "Technology of assembly creation of an experimental version OS Linux Kernels with quality assurance for applied and subject areas of knowledge". Euroazian Science Assosiation. www.ESA.ru, November-2021.-«Евразийское Научное Объединение» • № 11 (81) • Ноябрь, 2021.-94-105. DOI:10.5281/zenodo.5796926.

15. Лаврищева, Е. М. Парадигмы моделирования и программирования задач предметных областей знаний / Е. М. Лаврищева, И. Б. Петров, А. К. Петренко; под ред. А. И. Аветисяна, О. Е. Баксанского, М. М. Горбунов-Посадова; Институт системного программирования им. Иванникова. – Москва ; Берлин : Директ-Медиа, 2021.– 496 с. URL: https://biblioclub.ru/index.php?page=book&id=602516.

16. Лаврищева Е.М., Грищенко В.Н. Сборка модулей, объектов и компонентов в языках программирования.- 2-редакция книги Связь разноязыковых модулей в ОС ЕС.1982.-Москва.-ИСП РАН.- 19.08.2022.

17. Лаврищева Е.М. Наука моделирования и программирования задач математики, информатикии техники из информационных, интеллектуальных и сервисных ресурсов. Курс лекций для студентов МФТИ и ВНЗ.-Москва.2022.- Отдел оперативной полиграфии «Ф Физтехполиграф», 2022.

Общий список публикаций по тематике проекта РФФИ 19-01-00206 (2019-2021)

1. Лаврищева Е.М., Грищенко В.Н. Связь разноязыковых модулей в ОС ЕС. — М.: Финансы и статистика. — 1982. — 136 с.

2. Лаврищева Е.М. Интерфейс в программировании // Проблемы программирования. — Киев: 2006. — №2. — С.126–139; Formal fundamentals of component interoperability in programming / Lavrischeva K.M. // Cybernetics and Systems Analysis. — 2010. — Volume 46. — Issue 4. — P. 639–652. — doi:10.1007/s10559-010-9240-z

3. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование. — Киев: Наук. думка, 1991. — 236 с.

4. Липаев В.В., Позин Б.А., Штрик А.А. Технология сборочного программирования. — М.: Радио и связь, 1992.

5. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование. Основы индустрии программных продуктов. — Киев: Наук. Думка, 2009. — 371 с.

6. Ершов А.П. Опыт интегрального подхода к актуальной проблеме ПО // Кибернетика. — 1984. — С. 11–21;

7. Ершов А.Н. Научные основы доказательного программирования // Вестник АН СССР. — 1984. — № 10. — С. 9–19.

8. Web Services Description Language (WSDL) 1.1//W3C Note 15 March 2001. — URL: http://www.w3.org/TR/wsdl.*Semantic Web. — URL:* http://www.w3.org/2001/sw/ .

9. Reference Model for Service Oriented Architecture 1.0. — URL: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html

10. Web Services Resource 1.2 (WS-Resource). — URL: http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf.

11. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition) // W3C Recommendation. 27 April 2007. — URL: http://www.w3.org/TR/SOAP12-part1.

12. Semantic Web programming / S. Hebeler, M. Fisher, R.Blace, A/Peter-Lopes. — Willey Publishing Inc., 2008. — URL: http://semwebprogramming.org.

13. Лаврищева Е.М. Теория и практика фабрик программ // Кибернетика и системный анализ. — No. 6. — 2011. — С. 145–158; Theory and practice of software factories / K.M. Lavrischeva // Cybernetics and Systems Analysis. — 2011. — Volume 47. — Issue 6. — P. 961–972.

14. E.M. Lavrischeva. Classification of software engineering disciplines // Cybernetics and Systems Analysis. — 2008. — Volume 44. — Issue 6. — P. 791–796.

15. Лаврищева Е.М. Теория объектно-компонентного моделирования программных систем // Препринт ИСП РАН 29, 2016. — С. 1–52. — URL: ISBN978-5-9-474-25..

16. Иванников В.П., Дышлевый К.В., Мажелей С.Г и др. Распределенные объектно-ориентированные среды // Труды Института системного программирования РАН. — Том 1. — 2000. — С. 100–121.

17. Лаврищева Е.М. Рыжов А.Г. Применение теория общих типов данных стандарта ISO/IEC 11404 GPD применительно к Big Data // Conference "Actual problems in science and ways their development", 27 December 2016, http://euroasia-science.ru/. — С. 99–110.

18. Е. М. Лаврищева, Л. Е. Карпов, А. Н. Томилин, Системная поддержки решения бизнес-задач в глобальной информационной сети, Научный сервис в сети Интернет: Труды XVII Всероссийской научной конференции (19–24 сентября 2016 г., г. Новороссийск). — М.: ИПМ им. М.В. Келдыша, 2015. — с.101–127.

19. Лаврищева Е. М., Карпов Л. Е., Томилин А. Н. Семантические ресурсы для разработки онтологии научной и инженерной предметных областей // Научный сервис в сети Интернет: Труды XVIII Всероссийской научной конференции (19–24 сентября 2016 г., г. Новороссийск). — М.: ИПМ им. М.В. Келдыша, 2016. — с.126–138. — doi:10.20948/abrau-2016-8.

20. Лаврищева Е. М., Карпов Л. Е., Томилин А. Н. Подходы к представлению научных знаний в Интернет науке // Сб. XIX Всероссийский научной конференции «Научный сервис в сети Интернет», Новороссийск, 18–23 сентября 2017. — С. 310–326.

21. Лаврищева Е.М. Компонентная теория и коллекция технологий для разработки индустриальных приложений из готовых ресурсов // Труды 4-ой научно-практической конференции «Актуальные проблемы системной и программной инженерии», АПСПИ-2015, 20–21мая 2015. — С. 101–119.

22. Лаврищева Е.М. Программная инженерия. Тема 1. Теория Программирования. 50 с.; Тема 2. Технология программирования, 48; Тема 3. Базовые основы программной инженерии. — 52с. / /Методические пособия, Москва, МФТИ, 2016.

23. Лаврищева Е.М., Петренко А.К. Моделирование систем и их семейств // Труды ИСП РАН. — М.: 2016. — Том 28. — Вып. 6. — С.180 –190.

24. Кулямин В.В., Лаврищева Е.М., Мутилин В.С., Петренко А.К. Верификация и анализ вариабельных операционных систем // Труды ИСП РАН. — М.: 2017. — Том 28. — Вып. 3. — С.189–209.

25. Лаврищева Е.М. Программная инженерия. Парадигмы, Технологии, CASE-средства программирования, 2 изд. — М.: Юрайт, 2016. — 280 с.

26. Лаврищева Е.М. Программная инженерия и технология программирования сложных систем // М.: Юрайт, 2017. — 431с.

27. Лаврищева Е.М., Колесник А.Л., Стеняшин А.Ю. Объектно-компонентное проектирование программных систем // Теоретические и прикладные вопросы. — Весник КНУ, серия физ.-мат. наук. — Киев, 2013. — №4. — С. 150–164; Ekaterina Lavrischeva, Andrey Stenyashin, Andrii Kolesnyk. Object-Component Development of Application and Systems. Theory and Practice // Journal of Software Engineering and Applications. — 2014. — Vol. 7. — No. 9. — doi:10.4236/jsea.2014.79070.

28. Ekaterina M. Lavrischeva. Assembling Paradigms of Programming in Software Engineering // Journal of Software Engineering and Applications. — 2016. — Vol. 9. — No. 6. — P. 296–317. — doi:10.4236/jsea.2016.96021.

29. Lavrischeva Ekaterina. Ontological Approach to the Formal Specification of the Standard Life Cycle // Science and Information Conference-2015, July 28–30, London, UK, www.conference.thesai.org. — P. 965–972.

30. MacGregor S.D., Sykes D.A. Practical Guide to Testing Object-oriented software. — Addison-Wesley, 2001.

31. Sayyad A. S., Ingram J., Menzies T., Ammar H. Scalable product line configuration: a straw to break the camel's back // Proc. of IEEE/ACM 28-th International Conference on Automated Software Engineering (ASE 2013). — IEEE, 2013. — P. 465–474. — doi:10.1109/ASE.2013.6693104.

32. Лаврищева Е.М., Рыжов А.В. Подход к созданию систем и сайтов из готовых ресурсов // Научный сервис в сети Интернет: труды XX Всероссийской научной конференции (17–22 сентября 2018 г., г. Новороссийск). — М.: ИПМ им. М.В. Келдыша, 2018. — С. 321–346. — ISBN 978-5-98354-046-0.

33. E.M. Lavrischeva, A.K.Petrenko. Informatics-70. Computerization aspects of programming software and informatic systems technologies // Proc. ISP RAS. — 2018. — Vol. 1. — Issue 2. — P. 3–4.

34. Лаврищева Е.М., Аветисян А.И., Петренко А.К. Информатика. ЭВМ-70. Анализ и аспекты развития. — Доклад на конференции ISPRAS-2018. (http://0x1.tv/20181122AF).

35. E.M. Lavrischeva. The Scientific Basis of Software Engineering // International Journal of Applied and Natural Sciences (IJANS). — Vol. 7. — Issue 5. — Aug–Sep 2018. — P. 15–32. — ISSN(P): 2319-4014; ISSN(E): 2319-4022.

36. Лаврищева Е.М., Пакулин Н.В., Рыжов А.Г., Зеленов С.В. Анализ методов оценки надежности оборудования и систем. Практика применения методов // Труды ИСП РАН. — Том 30. — Вып. 3. — 2018. — С. 99–120. — doi:10.15514/ISPRAS-2018-30(3)-8. http://0x1.tv/20180517F

37. Лаврищева Е.М. Современные системы искусственного интеллекта // Симпозиум «Искусственный интеллект: различные подходы к его воплощению (компьютерно-сетевые, нейросетевые, квантовые технологии и другие)», Москва, 13 октября 2018. — Федеральное государственное образовательное учреждение при Правительстве Российской Федерации. Колледж информатики и программирования. — Презентация доклада Лаврищевой Е.М. в https://videonauka.ru/stati/30-metodika-prepodavaniya-tekhnicheskikh-distsiplin/237-informatika-i-evm-70-analiz-i-aspekty-razvitiya.

38. Лаврищева Е.М. Теория графового моделирования сложных систем из модулей для прикладных областей.- Научно-практический журнал «Высшая школа, июль (14) 2019.- с.27-46.

41. Лаврищева Е.М., Петренко А.К. Технология сборки интеллектуальных и информационных ресурсов Интернет // Научный сервис в сети Интернет: труды XXI Всероссийской научной конференции (23-28 сентября 2019 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2019. —С. 469-488. — URL: http://keldysh.ru/abrau/2019/theses/93.pdf doi:10.20948/abrau- 2019-93

42. Лаврищева Е.М. Теория графового моделирования сложных систем из модулей для прикладных областей.- Научно-практический журнал, Высшая школа. №14/ 2019.-.с.27-46: www.ran-nauka.ru © ООО «Инфинити», 2019. ISSN 2409-1677.

43. Лаврищева Е.М. Тория графового моделирования систем из модульных элементов для прикладных областей знаний.- Austria-science»1 часть №28/2019.- p.12-30. http://austria-science.info.

44. E.M. Lavrischeva, A.K. Petrenko and B.A.Pozin. Technology of Assembly of Intellectual and Information Resources Internet.-CEUR-WS-2019.org/vol-2543/ipaper-22.pdf.-27p.

45. Лаврищева Е.М. Теория графового моделирования систем из модульных элементов для прикладных областей знаний.- Austria-science»1 часть №28/2019.- p.12-30. http://austria-science.info.

46. Лаврищева Е.М., Зеленов С.В., Рыжов А.Г.. Теория моделирование программно-техническихсистем с обеспечением безопасности и надежности в среде Web-services Интернет. Труды Институт системного программирования РАН, Том 29. №6.-р. 221-231.

47. Екатерина Михайловна Лаврищева, Сергей Вадимович Зеленов. Модельный подход к обеспечению безопасности и надежности Web-сервисов. Труды ИСП РАН 2020, Том 32 № 5. С.151-163.

48. Е.М.Лаврищева, И.Б.Петров. Моделирование технических и математических задач прикладных областей знаний на ЭВМ. Труды ИСП РАН 2020, Том 32 № 6.- с.167-182.

49. Е.М.Лаврищева, И.Б.Петров. Теория моделирования технических и математических задач предметных областей знаний. Евроазиатское Научное Объединение. 2021.№ 1- (1).-с.35-43.

50. Лаврищева Е.М., Петренко А.К., Зеленов С.В., Козин С.В.. Технология сборочного создания экспериментального варианта ядра OS Linux с обеспечением качества и безопасности для применения в прикладных и технических системах.- Наука и технологии.-Системный администратор.- 2021. Ноябрь.- с.80-89.

51 Lavrischeva E. M. Petrenko A. K. Kozin V. P. "Technology of assembly creation of an experimental version OS Linux kernels with quality assurance for applied and subject areas of knowledge". Euroazian Science Assosiation. www.ESA.ru, november-2021.- p.94-105. DOI: 10.5281/zenodo.5796926.

52. Лаврищева, Е. М. Парадигмы моделирования и программирования задач предметных областей знаний / Е. М. Лаврищева, И. Б. Петров, А. К. Петренко; под ред. А. И. Аветисяна, О. Е. Баксанского, М. М. Горбунов-Посадова; Институт системного программирования им. Иванникова. – Москва ; Берлин : Директ-Медиа, 2021.– 496 с. URL: https://biblioclub.ru/index.php?page=book&id=602516.

53. Лаврищева Е.М., Грищенко В.Н. Сборка модулей, объектов и компонентов в языках программирования. - 2-редакция книги Связь разноязыковых модулей в ОС ЕС.1982.- Москва. -ИСП РАН.- 19.08.2022.

57. Лаврищева Е.М. Наука моделирования и программирования задач математики, информатикии техники из информационных, интеллектуальных и сервисных ресурсов. Курс лекций студентов МФТИ и ВНЗ.-Москва.2022.- Отдел оперативной полиграфии «Ф Физтехполиграф», 2022.

Публикации по Проекту РФФИ 206-2019-2021 на анг.

1. E. M. Lavrischeva. The theory graph modeling systems from quality modules of the application areas. http://ceur-ws.org/Vol-2514/ 2019.-p.235-247.

2. E.M. Lavrischeva, A.K. Petrenko and B.A.Pozin.Technology of Assembly of intellectual and information resources Internet, E-library, SEUR-2019.

3. E.M. Lavrischeva, A.K. Petrenko .Technology of Assembly of intellectual and information resources Semantic Web Internet/ http://keldysh.ru/abrau/2019/theses/93.pdf doi:10.20948/abrau-2019-93.

4. E. M. Lavrischeva. Graph theory modeling complex systems module elements for the application areas.- Scientific Journal «Austrua Science" info, №28/2019, p.12-29. http://austrua science/info.

5. Lavrischeva E.M. Zelenov S.V., Pakulin N.V. Ryzhov A.G. Proceeding of the ISP RAS, Volume 31, ussue. 5, 2019, pp. 95-108. DOI: 10.15514/ISPRAS-2019-31(5)-7/ http://0x1.tv/20180517F ; http://0x1.tv/20181122AF.

6. E.M. Lavrischeva.The Theory Graph Modeling and Programming Paradigm Systems from Modules to the Application Areas – DOI: https://doi.org/10.14738/tmlai.74.6782, Vol 12 No 6 (2019): Transactions on Machine Learning and Artificial Intelligence.

7. E. M. Lavrischeva. The Theory Graph Modeling and Programming Paradigms of Systems from Modules to the Application Areas.- Journal. Computer and Information Sciences – vol.12, №4 2019 (ISSN: 1913- 8989, E-ISSN: 1913-8997)

8. Lavrischeva E.M. Graph theory modeling complex systems modules for the application areas http://ran-nauka.ru/wp-content/uploads/2014/09/Nauka-14-2019.pdf.-p.24-44.

9. E.M. Lavrischeva, A.K. Petrenko and B.A.Pozin. Technology of Assembly of Intellectual and Information Resources Internet.-CEUR-WS-2019.org/vol-2543/ipaper-22.pdf.-27p.

10. Lavrischeva E. M. Petrenko A. K. Kozin V. P. "Technology of assembly creation of an experimental version OS Linux kernels with quality assurance for applied and subject areas of knowledge". Euroazian Science Assosiation. www.ESA.ru, november-2021.- p.94-105. DOI: 10.5281/zenodo.5796926.

11. Учебник Е.М.Лаврищева Е.М., Петров И.Б. «Технология моделирования и программирования технических и прикладных систем из ресурсов Интернет.-МФТИ.-2021.306с.

12. Учебник Е.М. Лаврищева.-Технология и инженерия разработки вариабельных программных и прикладных систем И интернет.- Долгопрудный.- www.perehletoff.ru/-440с.

13. Лаврищева Е.М. Наука моделирования и программирования задач математики, информатики, и техники из информационных, интеллектуальных и сервисных ресурсов. Учебно-методическое пособие.—Министерство науки и высшего образования РФ. Московский физико-технический институт. (Национальный исследовательский университет).-Москва, МФТИ.- 2023.-71с. –rio@mipt.ru

14. Lavrischeva E.M. of The pole Cybernetitics, Computer Science and Atifischal Intellect in collektion of resurces and assembly computer Systems.-Practica Orientited Science: UAE-RUSSIA-INDIA.-Materials of International University Scienific Forum.-july 17, 22/-UAE 2022.-65-69p-.

Приложение 1. Основные научные публикации по теме «Методы формального моделирования программных, технических и операционных систем»

1. Pohl K., Böckle G., van der Linden F. J. Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag, 2005. DOI: 10.1007/3-540-28901-1.
2. Bachmann F., Clements P. Variability in software product lines. CMU/SEI Technical Report CMU/SEI-2005-TR-012, 2005.
3. Lotufo R., She S., Berger T., Czarnecki K., Wąsowski A. Evolution of the ux kernel variability model. Proc. of SPLC'10, LNCS 6287:136-150, Springer, 2010. DOI: 10.1007/978-3-642-15579-6_10.
4. Чарнецки К., Азенакер У. Порождающее программирование. Методы, инструменты, применение. – Изд. Дом №Питер» - М.: СПб.-Харьков-Минск.- 2005.-730 с.
5. Лаврищева Е.М., Грищенко В.Н. Методы и средства объектно-компонентного программирования.- Кибернетика и системный анализ. - 2003.-№1, с. 39-55.
6. Berger T., She S., Lotufo R., Wąsowski A., Czarnecki K. A study of variability models and languages in the systems software domain. IEEE Transactions on Software Engineering, 39(12):1611-1640, 2013. DOI: 10.1109/TSE.2013.34.
7. Kang K., Cohen S., Hess J., Novak W., Peterson S. Feature-oriented domain analysis (FODA) feasibility study. CMU/SEI Technical Report CMU/SEI-90-TR-21, 1990.
8. Zippel R. et al. Kconfig language. https://www.kernel.org/doc/Documentation/kbuild/kconfig-language.txt.
9. Лаврищева Е.М. Теория объектно-компонентного моделирования программных систем // www.ispras.ru/preprints/docs/prep_29_2016.pdf, Lavrischeva.
10. Лаврищева Е. М., Коваль Г.И., Слабоспицькая О.О., Колесник А.Л. Особенности процессов управления созданием семейств программных систем. Проблемы программирования, (3):40-49, 2009.
11. Лаврищева Е.М., Слабоспицькая О.А., Коваль Г.И., Колесник А.А. Теоретические аспекты управления вариабельностью в семействах ПС. Весник КНУ, серия физ.– мат. наук, 2011, № 1.- с. 151-158.
12. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование. Основы индустрии программных продуктов. – К.: Наук. Думка, 2009 – 371 с.
13. Кулямин В.В. Лаврищева Е.М., Мутилин В.С., Петренко А.К. Верификация и анализ вариабельных операционных систем. Труды ИСП. РАН. Том 28. Вып.3.с.189-209.
14. Лаврищева Е.М. Петренко А.К. Моделирование систем и их семейств. Труды ИСП РАН, 2016, том 28. вып. 6, с. 180 -190.
15. Лаврищева Е.М., Слабоспицкая О.А. Подход к построению объектно-компонентной модели семейства программных продуктов (укр.) // Проблемы программирования, 2013.–№3 – с.14–26.
16. Ekaterina Lavrischeva, Andrey Stenyashin, Andriy Kolesnyk, Object-Component Development of Application and Systems. Theory and Practice, Journal of Software Engineering and Applications, 2014, 7, Published Online August 2014 in SciRes http://www.scirp.org/journal/jsea.
17. Лаврищева Е.М., Колесник А.Л., Стеняшин А.Ю. Объектно-компонентное проектированием программных систем. Теоретические и прикладные вопросы – Весник КНУ, серия физ.-мат. наук. –2013. –№4. – С. 150–164.
18. Ekaterina Lavrischeva, Andrey Stenyashin, Andriy Kolesnyk, Object-Component Development of Application and Systems. Theory and Practice, Journal of Software

Engineering and Applications, 2014, 7, Published Online August 2014 in SciRes http://www.scirp.org/journal/jsea.
19. Kästner C., Dreiling A., Ostermann K., Variability Mining with LEADT, In Proc. Int'l Conf. Generative Programming and Component Engineering (GPCE), pp. 157–166, 2009.
20. Колесник А.Л. Модели и методы разработки семейства Вариантных программных систем. – Автореф. дис, КНУ, 2013. –22с. (укр).
21. Melo J., Flesborg E., Brabrand C., Wąsowski A. A quantitative analysis of variability warnings in Linux. Proceedings of the 10-th International Workshop on Variability Modelling of Software-intensive Systems, pp. 3-8. ACM, 2016. DOI: 10.1145/2866614.2866615.
22. Колесник А.Л. Поддержка процессу управления вариабельностью в семействах программных систем // Проблемы программирования. – 2012, № 2-3.- с.182-191.
23. Deming E. New economics for manufactures, governments and education, 1993.
24. Коваль Г.І. Байесовские сети – способ оценивания и прогнозирования качества програмного забезпечення // Проблеми програмування. – 2005. - №2. – С.15-23.
25. Lauenroth K., Toehning S., Pohl K. Model checking of domain artifacts in product line engineering. Proc. Intl. Conf. on Automated Software Engineering (ASE), pp. 269-280. IEEE, 2009.DOI: 10.1109/ASE.2009.16.
26. Лаврищева Е.М. Программная инженерия. Парадигмы, Технологии, CASE-средства программирования.2 изд. – М: Юрайт, 2016. – 280 с.
27. Sayyad A. S., Ingram J., Menzies T., Ammar H. Scalable product line configuration: a straw to break the camel's back. Proc. of IEEE/ACM 28-th International Conference on
28. Automated Software Engineering (ASE 2013), pp. 465-474. IEEE, 2013.DOI: 10.1109/ASE.2013.6693104.
29. Колесник А.Л. Модели и методы разработки семейств вариантных программных систем.- Автореф.- КНУ.- 2013. 22 с.
30. MacGregor S.D., Sykes D.A. Practical Guide to testing Object-oriented Software.-2001, Addison-Wesley Professional.
31. Андон Ф.И. Основы инженерии качества программных систем. Андон Ф.И. [и др.]. – К: Академпериодика, 2007. – 680 с.

Приложение 2. Дополнительная литература к разделу 5

1. Эммерих В. Конструирование распределенных объектов. Методы и средства программирования интероперабельных объектов в архитектурах OMG/CORBA, Microsoft COM и JAVA RMI.-Мир.-2002.
2. Лаврищева Е.М., Рыжов А.Г. Подход к моделированию систем и сайтов из готовых ресурсов, Конференция Абрау-18, "Научный сервис в сети интернет". М.: ИПМ им.М.В.Келдыша, с..137-151.
3. Лаврищева Е.М. Теория объектно-компонентного моделирования программных систем // www.ispras.ru/preprints/docs/prep_29_2016.
4. Лаврищева Е. М., Карпов Л. Е., Томилин А. Н., Семантические ресурсы для разработки онтологии научной и инженерной предметных областей, Сб. "Научный сервис в сети Интернет" 19-24 сентября 2016, Абрау , с.126-138.
5. Лаврищева Е.М., Грищенко В.Н. Сборочное программирование. Основы индустрии программных продуктов. – К.: Наук. Думка, 2009 – 371 с.
6. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition) W3C Recommendation 27 April 2007 http://www.w3.org/TR/SOAP12-part1/
7. Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001 http://www.w3.org/TR/wsdl
8. Влад Боркус. Web-сервисы: современные стандарты. Аналитический обзор.- vlad_borkus@pcweek.ru//-Vjcrdf? 2004-2005/
9. SPARQL Query Language for RDF W3C. http://www.w3.org/TR/rdf-sparql-query/
10. Reference Model for Service Oriented Architecture 1.0 http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html
11. Web Services Resource 1.2 (WS-Resource) http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf
12. Web Services Resource Properties 1.2 (WS-ResourceProperties) http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf
13. http://docs.oasis-open.org/dita/v1.2/spec/DITA1.2-spec.html/.
14. Кулямин В.В. Лаврищева Е.М., Мутилин В.С., Петренко А.К. Верификация и анализ вариабельных операционных систем. Труды ИСП. РАН. Том 28. Вып.3.с.189-209.
15. Лаврищева Е.М. Петренко А.К. Моделирование систем и их семейств. *Труды ИСП РАН*, 2016, том 28, вып. 6, 2016, стр. 49-64. ISSN 2220-6426 (Online), ISSN 2079-8156 (Print). DOI: 10.15514/ISPRAS-2016-28(6)-4.
16. Ekaterina Lavrischeva, Andrey Stenyashin, Andriy Kolesnyk, Object-Component Development of Application and Systems. Theory and Practice, Journal of Software Engineering and Applications, 2014, 7, Published Online August 2014 in SciRes http://www.scirp.org/journal/jsea.
17. Лаврищева Е.М. Программная инженерия. Парадигмы, Технологии, CASE-средства программирования.2 изд. – М: Юрайт, 2016. – 280 с.
18. Ларищева Е.М. Программная инженерия и технология программирования сложных систем. М: Юрайт, 2018. – 432 с.
19. Лаврищева Е.М. Теория объектно-компонентного моделирования вариабельных программных систем. Доклад на Международной научно-практической конференции «Теория активных систем (ТАС-2016), 16-17 ноября 2016.- ИПУ РАН им. В.А.Трапезникова.
20. Доклад Лаврищева Е.М. Научные основы построения программных и информационных систем. Е-обучение теории и методам, XII Научно-практическая конференция «Современные информационные технологии и ИТ-образование», 25-26

ноября 2016, МГУ им. М.В.Ломоносова.
21. Лаврищева Е.М. Петренко А.К. Моделирование систем и их семейств. *Труды ИСП РАН*, 2016, том 28, вып. 6, 2016, стр. 49-64. ISSN 2220-6426 (Online), ISSN 2079-8156 (Print). DOI: 10.15514/ISPRAS-2016-28(6)-4.
22. E.M. Lavrischeva, V.S. Mutilin, A.G. Ryzhov. Designing variability models for software, operating systems and their families, 2017, Сборник ИСП РАН, 2017, issue 5, 2017, pp.93-109.- DOI: 10.15514/ISPRAS- 2017(5).
23. Е.М. Лаврищева, В.С. Мутилин, А.Г. Рыжов. Аспекты моделирования вариабельных программных и операционных систем.- Сб. XIX Всероссийский научной конференции «Научный сервис в сети Интернет», Новороссийск, 18-23 сентября 2017.
24. E.M. Lavrischeva, V.S. Mutilin, A.G. Ryzhov. Designing variability models for software, operating systems and their families, 2017, Сборник ИСП РАН, 2017, issue 5, 2017, pp.93-109.- DOI: 10.15514/ISPRAS- 2017(5).
25. Lavrischeva E.M.. Development of the theory programs and systems in the USSR. History and modern theory .- Sorucom-2017, IEEE Springer-2017. P.31-47.
26. Лаврищева Е.М., Пакулин Н.В., Рыжов А.Г., Зеленов С.В. Анализ методов оценки надежности оборудования и систем. Практика применения методов. Труды ИСП РАН, том5 DOI: 10.15514/ISPRAS-2018-30(3), 2018.-р.41-53.
27. Лаврищева Е.М. Фундаментальные основы программной инженерии.- Сборник научных трудов 5-международной конференции «Актуальные проблемы системной и программной инженерии». - Сборник трудов АПСПИ-2017, 14-16 октября 2017.- с.163-177.
28. Lavrischeva E.M., Petrov I.B. Ways of Development of Computer Technologies to Perspective Nano, Agenda - Future Technologies Conference (FTC) 2017, 29-30.- November 2017| Vancouver, Canada, p. 539-549.
29. Lavrischeva E.M., Petrenko A.K. Informatics. Formation of computer software and technologies of software systems. Trudy ISP RAN/Proc. ISP RAS, 2018.
30. Лаврищева Е.М., Рыжов А.Г. Подход к моделированию систем и сайтов из готовых ресурсов. Научный сервис в сети Интернет: труды XX Всероссийской научной конференции (17-22 сентября 2018 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2018. — С. 321-345. — URL: http://keldysh.ru/abrau/2018/theses/50.pdf doi:10.20948/abrau-2018-50
31. Лаврищева Е.М., Петренко А.К. Технология сборки интеллектуальных и информационных ресурсов Интернет // Научный сервис в сети Интернет: труды XXI Всероссийской научной конференции (23-28 сентября 2019 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2019. С. 469-488. — URL: http://keldysh.ru/abrau/2019/theses/93.pdf doi:10.20948/abrau-2019-93
32. Лаврищева Е.М. Теория графового моделирования сложных систем из модулей для прикладных областей.- Научно-практический журнал, Высшая школа. №14/ 2019.- .с.27-46: www.ran-nauka.ru © ООО «Инфинити», 2019. ISSN 2409-1677.
33. Лаврищева Е.М. Теория графового моделирования систем из модульных элементов для прикладных областей знаний.- Austria-science»1 часть №28/2019.- р.12-30. http://austria-science.info.
34. E.M. Lavrischeva, A.K. Petrenko and A.Pozin. Technology of Assembly of Intellectual and Information Resources Internet.-CEUR-WS-2019.org/vol-2543/ipaper-22.pdf.-27p.
35. Лаврищева Е.М., Зеленов С.В., Рыжов А.Г.. Теория моделирование программно-технических систем с обеспечением безопасности и надежности в среде Web-services Интернет. Труды Институт системного программирования РАН, Том 29.-№31.
36. Екатерина Михайловна Лаврищева, Сергей Вадимович Зеленов. Модельный подход к обеспечению безопасности и надежности Web-сервисов. Труды ИСП РАН 2020, Том 32 № 5, С.151-163.

37. Е.М.Лаврищева, И.Б.Петров. Моделирование технических и математических задач прикладных областей знаний на ЭВМ. Труды ИСП РАН 2020, Том 32 № 6.- с.167-182.
38. Е.М.Лаврищева, И.Б.Петров. Теория моделирования технических и математических задач предметных областей знаний. Евроазиатское Научное Объединение. 2021.№ 1-(1).-с.35-43.

Приложение 3. Дополнительная литература к разделам 4-6.

1. E. M. Lavrischeva. The theory graph modeling systems from quality modules of the application areas. http://ceur-ws.org/Vol-2514/ 2019.-p.235-247.
2. E.M. Lavrischeva, A.K. Petrenko and B.A.Pozin.T echnology of Assembly of intellectual and information resources Internet, E-library, SEUR-2019.
3. E.M. Lavrischeva, A.K. Petrenko .Technology of Assembly of intellectual and information resources Semantic Web Internet/ http://keldysh.ru/abrau/2019/theses/93.pdf doi:10.20948/abrau-2019-93.
4. E. M. Lavrischeva, Graph theory modeling complex systems module elements for the application areas.- Scientific Journal «Austrua Science" info, №28/2019, p.12-29. http://austrua science/info/
5. Lavrischeva E.M. Zelenov S.V., Pakulin N.V. Ryzhov A.G. Morozov S.V. Tarlapan O.A. Proceeding of the ISP RAS, Volume 31, ussue. 5, 2019, pp. 95-108. DOI: 10.15514/ISPRAS-2019-31(5)-7/ http://0x1.tv/20180517F; http://0x1.tv/20181122AF.
6. 6. E.M. Lavrischeva.The Theory Graph Modeling and Programming Paradigm Systems from Modules to the Application Areas – DOI: https://doi.org/10.14738/tmlai.74.6782, Vol 12 No 6 (2019): Transactions on Machine Learning and Artificial Intelligence.
7. E. M. Lavrischeva. The Theory Graph Modeling and Programming Paradigms of Systems from Modules to the Application Areas.- Journal. Computer and Information Sciences – vol.12, №4 2019 (ISSN: 1913- 8989, E-ISSN: 1913-8997)
8. Lavrischeva E.M. Graph theory modeling complex systems modules for the application areas.http://ran-nauka.ru/wp-content/uploads/2014/09/Nauka-14-2019.pdf.-p.24-44.
9. E.M. Lavrischeva, A.K. Petrenko, and B.A.Pozin. Technology of Assembly of Intellectual and Information Resources Internet.-CEUR-WS.org/vol-2543/2019.paper-22.pdf.-27p.
10. E. M. Lavrischeva, Doctor of phys.-math. Sci., Professor of MIPT, scientifically specialist ISPRAS. The theory graph modeling systems from quality modules of the application https://areas.-seur.org.- 2020.-112-122.

Оглавление

Предисловие ... 3

Раздел 1. Форма 501. Проект № 16-01-00352-2016. Теория и методы разработки вариабельных программных и операционных систем ... 6

Подраздел I проекта РФФИ 352-2016. Теория разработки вариабельных программных и операционных систем ... 9

1. Общие положения теории и методов моделирования прикладных систем ... 9
1.1. Современные подходы к моделированию сложных систем ... 10
1.2. Подходы к моделированию изменяемых систем (SPLE, ОКМ, Grid, Etics) ... 11
1.3. Вариабельность систем и управление ... 17
1.4. Стандарт конфигурационной сборки прикладных систем ... 19
1.5. Основные положения компонентного программирования ... 20
1.6. Веб-сервисы для создания прикладных систем и сайтов в Интернет ... 22

Заключение к заявке Проекта РФФИ 352-2016 ... 25

Публикации по I подразделу проекта РФФИ 352-2016 ... 26

Подраздел II проекта РФФИ 352-2017. Общие подходы к моделированию вариабельных ОС ... 28
2.1. Сущность подхода Data Mining, Variability Mining при извлечении артефактов из ОС ... 29
2.2. Анализ генерации вариантов конфигурации ОС, ПС ... 31
2.3. Верификация моделей вариабельности ПС и ОС ... 34
2.4. Средства статической верификации ядра системы Linux ... 38

Заключение к подразделу II проекта РФФИ 352 -2017 и литература ... 39

Публикации по подразделу II Проекта РФФИ 352-2017 ... 40

Подраздел III проекта РФФИ 352-2018. Метод анализа технологии программирования и ОС ... 42
Введение ... 42
3.1. Подходы к созданию технологий программирования производственных продуктов ... 44
3.2. Зарубежные ТЛ SPLE (Software Product Line Engineering) ... 46
3.3. Типовые AppFab (фабрики программ) на современных ОС платформах Интернет ... 48
3.4. Парадигмы технология сборки систем из готовых ресурсов – ГОР, КПИ, Reuses ... 56
3.5. Конфигурационная сборка программных артефактов, ПС и СПС ... 61
Заключение по подразделу III проекта РФФИ 35202018 ... 64
Литература к п. 3.1 - 3.5 ... 65

3.6. Научный сервис и Е-наука Интернет для создания предметных областей знаний ... 68
3.6.1 Сетевые технологии в W3C Интернет ... 72
3.6.2. Сервисно-ориентированная архитектура SOA и SCA ... 74
3.6.3. Среда взаимодействия систем WCF ... 77
3.6.4. Технологии и языки Grid - OGSA ... 80
Заключение по SOA SCA и системным сервисам ... 82
Литература к подразделу 3.6 ... 82
3.7. Средства Интернет для использования сервисных ресурсов ... 83
3.7.1. Онтологии для представления существующих знаний ... 86
3.7.2. Элементы онтологии для задания доменов и систем ... 88
3.7.3. Описание домена – Вычислительная геометрия ... 88
3.7.4. Разработка онтологии стандарта жизненного цикла 12207 ... 92
Заключение по подразделу 3.7 ... 98
Литература к 3.6 - 3.7 ... 98

Общее заключение по Методам и подходам информатизации в проекте РФФИ 352 ... 99

Раздел 2. Форма 502. Развернутый научный отчет по проекту 16-01-00352-2017 (№ 35202018). Методы формального моделирования программных, технических и операционных систем ... 104

1. Анализ теоретических основ формального описания программ и объектов для сложных систем ... 105

1.1. Теория программ для первых ЭВМ ... 106

1.2. Теория технологии программирования синтез, сборка, доказательство ... 106
1.2.1. Фундаментальные и общие типы данных в программных ресурсах ... 108

1.3. Теория объектно-компонентного моделирования (ОКМ) изменяемых систем 108

2. Методы верификации, конфигурирования архитектуры их модулей и тестирования копонентов и систем из них ... 109
2.1. Верификация моделей MF и систем ... 109
2.2. Управление вариабельностью моделей и систем ... 110
2.3. Конфигурационная сборка ГОР, КПИ в ПС и СПС ... 111
2.4. Тестирование ПС семейства ... 113

2.5. Вариабельность в пространстве проблем и решений для ПС и ПТС ... 115

3. Подходы к созданию вариантов ядра OS LINUX для предметных областей знаний ... 117
3.1. Моделирование вариабельности с помощью диаграмм характеристик ... 120
3.2. Вариабельные зарубежные и отечественные системы ... 121
3.3. Электронная наука для представления знаний о задачах систем ... 129
3.4. Управление знаниями ... 131
3.5. Средства создания моделей знаний ... 132

3.5.1. Виды онтологий 134
3.5.2. Web-средства, Grid в E-Sciences для представления областей знаний 135
3.5.3. Веб-сервисы OGSA - Grid и Etics 137
3.5.4. Инструменты разработки Веб- систем средствами Grid 140
Заключение 141
Литература к 3.1-3.5 раздела 2 РФФИ-352- 2017 141

Научные работы по реализации задач проекта РФФИ 352- 2017г. 142

Основные результаты по проекту РФФИ-352 в 2017 году 145

Раздел 3. Форма 503. Проект РФФИ 16-01-00352 (2018). Теория и методы моделирования сложных прикладных систем и обработки данных 146

1.1. Основные задачи проекта 3 147
1.2. Новые направление информатизации - интеллектуализация знаний 149
1.3. Разработаны средства обработки фундаментальных и общих типов данных ресурсов 150
1.3.1. Отечественный метод преобразования ТД разнородных ресурсов на основе интерфейсов 150
1.3.2. Неструктурированные большие данные- Big Data 153
1.3.3. Формальное описание изменяемых прикладных систем объектно-компонентным методом ОКМ 154
1.3.4. Взаимодействия и верификации программ и систем 155
1.3.5. Управление вариабельностью моделей и систем 157
1.4. Моделирования систем из компонентов 158
1.5. Создание ПС и СПС из системных и прикладных сервисов 160
1.6. Проектирование веб-систем из сервисных компонентов в Интернет 162
1.7. Конфигурирование ресурсов Веб-систем по стандарту IEEE 829-1996-2012 166
1.8. Моделирование варианта ядра ОС Linex для прикладных областей знаний 167
1.8.1. Описание эксперимeтального варианта ядра OS Linux 169
1.8.2. Конфигурирование архитектуры систем и ОС в Grid, Etics, BCD 172
1.8.3. Апробация и создание экспериментальногоь варианта OS Linux (toolchain) 174
1.9. Описание средств системы LEADT для извлечения знаний 179
1.10. Средства масштабируемости клиента и сервера для веб-систем и сайтов 181
1.11. Обеспечение качества ПО, прикладных систем и Веб-систем 182

Заключение по подразделу проекта РФФИ 352-2018 183

Литература к разделу 3 185

Раздел 4. Форма 504. Проект РФФИ 19-01-00206 (2019-2021). Модели, методы и средства обеспечения надежности и качества технических и программных средств 196

Введение 196

1. Научные основы технологии и инженерии программирования систем. 198
1.1. Математическая теория моделирования прикладных систем Г. Буча (1987) 202
1.2. Методы математической спецификации задач предметных областей знаний ..203
1.3. Метод графового моделирования программных структур 206
1.3.1. Матричное представление графов программных структур 209
1.3.2. Математические операции над элементами графа 212
1.3.3. Характеристика простых и сложных графовых структур 214
1.3.4. Операции связывания (сборки) модульных структур 217
1.3.5. Подход к графовому моделированию сложных структур ОКМ 219
1.3.6. Теория преобразования данных в методе сборки 223
1.3.7. Операции изоморфного отображения передаваемых данных 224
1.3.8. Трансформация данных с учетом стандарта ISO/IEC 11404 GDT-2012 225

Список литературы к 1.1. – 1.3.7 226

2. Технология сборки ПС, ПТС с обеспечением качества 228
2.1. Способы сборка систем из модульных элементов через сборшик ресурсов Интернет 228
2.2. Создание Общего сборщика разнородных ресурсов Интернет 234
2.2.1. Теоретический базис преобразования ТД FDT, GDT, GLD 234
2.3. Технологии программирования (ТП) и путь ее развития после 1992 237
2.4. Описание задач сборщика интеллектуальных и информационных ресурсов в Интернет 240
2.5. Конфигурационная сборка сервисных ресурсов Web-систем 241
2.6. Задачи сборка экспериментального варианта ядра OS Linux 242
2.7. Парадигмы программирования задач предметных областей знаний 243
2.8. Обеспечение надежности и качества создаваемых систем 244

3. Средства Интернет для представления знаний 245
3.1. Средства создания моделей знаний 245
3.2. Общее представление знаний онтология прикладных областей 246
3.3. Базовые ресурсы Семантик Веб 247
3.4. Характеристика сервисных и семантических ресурсов Интнернет 253
3.5. Моделирования технических и программных систем в рамках проекта РФФИ.... 254
3.6 Интеллектуальные и информационные ресурсы Интернет 256
3.7. Сервисные и информационные ресурсы при сборке Веб-систем 258
3.8 Обеспечение безопасности, надежности и качества ПТС 259
3.8.1. Процессы ЖЦ для разработки ПТС 261
3.8.2. Модели и методы качества ПТС и их оценивание по стандартам 262
3.8.3. Марковский анализ безопасности и надежности ТС (Зеленов С.) 270
3.8.4. Прогнозирование и распределение надежности по компонентам 273
3.8.5. Заключение по проекту РФФИ 206 -2019 274
Список литературы к под разделу РФФИ 206-2019 275

Доклады и статьи по тематике проекта РФФИ №19-01-00206 -2019 276

Раздел 5. Форма 505. Проект РФФИ 19-01-00206-2020. Методы оценки надежности и безопасности технических и программных систем 283

5.1. Методы надежности оборудования и систем 284
5.2. Определение понятия надежности и безопасности систем 284
5.3. Базовые понятия моделей надежности и безопасности 287
5.4. Классификация моделей надежности ПО 298
5.5. Оценка надежности систем реального времени 301
5.6. Инженерия надёжности ПО 303
5.7. Проведение по моделям оценки надежности ПО 306
5.8. Технологические модули (ТМ) оценки надежности систем 308
Выводы 311
Литература к разделу 5 проекта РФФИ 206-2020 311
5.9. Принципы обеспечения надежности компонентов Веб- систем 312
5.10 Функция надежности модели Дж. Мусы в управлении качеством ПС 315

5.11. Технология сборки интеллектуальных и информационных ресурсов с обеспечением качества и безопасности 319
5.12. Интеллектуализация знаний 320
5.13. Технология сборки сервисных ресурсов в Интернет 326
5.13.1. Индустриальные системы сборки ресурсов 326
5.13.2. CASE инструменты и стандарты автоматизации сборки программ 327
5.13.3. Метод к преобразованию типов данных при сборке ресурсов 329
5.12.4. Общие теоретические задачи преобразования ТД стандарта GPD 338
5.13.5. Задачи общего сборщика ресурсов систем и индустрии фабрик программ 341
5.13.6. Заключение по технологии сборки ресурсов в Интернет 343

Раздел 6. Форма 506. Проект РФФИ 19-01-00206 (2021). Методы и средства обеспечения надежности, безопасности и защиты технических, программных и операционных систем 345

6.1. Проведение конфигурационной сборки в Grid и Etics в рамках E-Science 348
6.2. Стандартные средства для проведения конфигурационной сборки: 358
6.3. Проведение сборки экспериментального варианта ядра OS Linux 359
6.4. Современные средства обеспечения гарантоспособной надежности ПТС и ОС 362
6.5. Начальные подходы обеспечения надежности и качества ПТС 368

Заключение. По проекту РФФИ 206 -2021 371

Доклады и статьи по тематике проекта РФФИ №19-01-00206-2021 372

Приложение 1. Основные научные публикации по теме «Методы формального моделирования программных, технических и операционных систем» 389

Приложение 2. Дополнительная литература к разделу 5 391

Приложение 3. Дополнительная литература к разделам 4-6 394