Е.М. Лаврищева
В.Н. Грищенко

Сборка модулей, объектов и компонентов в языках программирования

Институт системного программирования
им. В.П. Иванникова РАН

Е.М. Лаврищева
В.Н. Грищенко

Сборка модулей, объектов и компонентов в языках программирования

Вторая редакция книги
Лаврищева Е.М., Грищенко В.Н.
Связь разноязыковых модулей в ОС ЕС.–
М.: Финансы и статистика.- 1982.

Москва 2022

ББК 32.073
Л13

Лаврищева Е. М., Грищенко В. Н.

Сборка модулей, объектов и компонентов в языках программирования.— М., ИСП РАН.— 2022 — 174 с.

Книга посвящена методам объединения, сборки модульных элементов, записанных в языках программирования ОС ЕС и интеллектуальных ресурсов в Web. Цель работы – научить программистов методу сборки модулей в разных языках программирования для создания сложных программных систем и комплексов путем их конструирования из готовых ресурсов с использованием разработанной библиотеки межъязыкового интерфейса. От пользователей требуется знание языков программирования высокого уровня для описания функций прикладных систем модульными элементами в современных языках программирования в средах IBM 360 (ОС ЕС), VS.Net, Java.Net, Corba и др. Приводятся функции библиотеки межъязыкового интерфейса 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 языках программирования (Java, Ruby, Basic, C, C++ и др.) в средах Интернет (IBM, VS.Net, Corba, Grid, Etic, BSD, Oracle BD и др.) для разработки программных систем различного назначения.

От рецензента

Доктор физ.-мат наук, профессор, Лауреат премии Совета Министров СССР (1985г.), дважды лауреат Государственной премии Украины, 1991 и 2003 гг., заслуженный деятель науки и техники Украины 2008 г. Екатерина Михайловна Лаврищева известна как крупный специалист в области программной инженерии и технологий программирования. С 1959 по 2014 год она работала в Институте кибернетики им. В.М. Глушкова в Киеве и с 2014 года она работает Институте системного программирования им. В.П. Иванникова РАН.

Е.М. Лаврищева внесла существенный вклад в развитие различных задач системного программирования:

- методами компиляции,
- технологиями разработки больших программных комплексов ответственного назначения,
- методами моделирования программ,
- методами описания вариабельности и управления конфигурациями,
- парадигмами сборочного программирования (объектного, модульного, компонентного, аспектного и др.),
- развитием стандартов в области программной инженерии,
- методами обеспечения качества и надежности и безопасности программных и операционных систем.

Особое место в творческой и практической деятельности Е.М. Лаврищевой занимают работы по сборочному программированию. Эти работы стали классическими, на них ссылаются как отечественные, так и зарубежные исследователи.

Д.ф.-м.наук, профессор А.К. Петренко

Предисловие

Одним из современных и перспективных направлений развития системного программирования является повышение уровня технологии сборки сложных программных агрегатных структур (программ, комплексов, систем, пакетов программ) и технических приборов и устройств для медицины, физики, биологии, химии, авиации и др.

В последнем десятилетии значительное повышение уровня технологии создания отдельных программ происходило за счет применения высокоразвитых языков программирования (ЯП) и постоянного развития их возможностей в направлении сборки отдельных модульных элементов в сложные структуры [1]. В результате в практике прикладного программирования прочное место заняли ЯП: ПЛ/1, Фортран, Кобол, Алгол-60, Смолток в системном программировании, с помощью которых собираются сложные программные и технические средства и приборы.

С возникновением ЯП высокого уровня использовались такие методы программирования: структурный [2], композиционный [3], модульный [4], R-технология [5], метод сверху-вниз [6], метод формализованных технических заданий [7] н др. Цель всех этих методов — совершенствовать технологию создания сложных программ из готовых компонентов повторного использования (КПИ) и готовых информационных ресурсов, Reuses, находящихся в Общих Хранилищах и Библиотеках современных сред Интернет.

Разработан метод модульного программирования (1975), методологическую основу которого составляет анализ алгоритмов задач с целью выделения модулей и их объединения (сборки) в требуемые конфигурации программных и технических систем.

Понятие модуля в последние годы расширено объектами, компонентами, сервисами, агентами и др. Каждый такой элемент обладает общими свойствами, к которым относятся такие: неделимость, логическая и программная завершенность, независимость (свойство повторной используемости), согласованность по управлению, передаче данных, использованию общей памяти и раздельная компиляция.

Рассмотренные свойства являются общими для модульных элементов в классе ЯП, широко используемых при описании алгоритмов функций, и эти свойства однотипно задавать в виде специальной информационной интерфейсной части программного элемента — паспорта, задаваемого стандартно, независимо от ЯП представления модульных ресурсов.

Благодаря паспортным данным модульный элемент становится, аналогичен детали с конструкторской документацией в технике конвейерной сборки автомобилей, которую можно использовать как при ручном, так и при автоматизированном создании программных и технических систем.

Таким образом, модульное программирование позволяет представить, создаваемый сложный программный агрегат в виде совокупности независимых модулей, заданных в графе предметной области. Процесс формирования модульной структуры агрегата проводится как в направлении сверху-вниз, так и в направлении снизу-вверх. В начале разработки из общих задач выделяются частные, а затем для них создаются простые конструктивные объекты — модули, интегрируемые в программный агрегат, система. Готовые типовые модули накапливаются в Хранилищах для повторного использования в Интернет среде.

Технология создания модульных структур программ ставит перед разработчиками сложных структур не только задачу расчленения на модули, но и задачу сборки модулей с обработкой передаваемых данных в разных средах и платформ. Для определения операций сборки существенное значение имеет форма представления передаваемых данных и принципы их реализации в разных вычислительных средах Интернет.

Объектом исследования и реализации данной книги является метод сборки исходных и реализованных модульных элементов в виде объектов, компонентов, сервисов, представление задач преобразования данных в ЯП (Фортран, Алгол, ПЛ/1, Кобол, Снобол)

и в новых современных ЯП (Ada, Basic, C++, Python, Ruby, Java и др.) к типу связанного объекта с учетом платформы ЭВМ. В работе проведено сопоставление средства первых ЯП по представлению и обработке их данных исходя из фундаментальной теории типов данных (ФДТ). В первом варианте данной книги рассматривались данные исходя из ФДТ ЯП и библиотеки из 64 примитивных функций преобразования типов данных, разработанных В, Н. Грищенко (глава 3, 4). Варианты сборки модулей в разные структуры проводились по графу подсистем, систем, как агрегатных структур.

Данная книга расширена разделами 2.1.7 и 2.2 теорией ФДТ и преобразования данных в разных ЯП. Дополнена глава 5 по теории преобразования ТД обмениваемых модульных элементов согласно стандартов ISO/IEC GDT-2009, GPD- 2012; ISO 11404 2002, ISO/IEC Cycle Life-2007, 2012 для новых ЯП, реализованных в операционных средах после 1996 - MS.Net, Java.Net, Grid, Ethics, Linus, Intel и др. Глава 6 посвящена реализации метода сборки информационных и интеллектуальных ресурсов. Представлены операции конфигурационной сборки программных, технических и операционных систем стандарта IEEE/ISO 828 Configuration и набор операции сборки - assembly модулей

Приведена реализация новых операций сборки (make, config, building, cmake, weaver) систем с помощью систем BSD (make), Grid (config, assembly), Sparol (assembly), Ethics (build<config), JavaE (cmake). Apache Maven и др.

Представлен общий сборщик таких ресурсов, который позволит проводить автоматизированную сборку в разных средах Интернет с операциями преобразования обмениваемых данных, включая большие данные типа Big Data в вычислительной среде Cloud Computing.

Описаны средства Семантик Веб по онтологизации предметных областей знаний. Реализованы Веб приложения и домен жизненного цикла (ЖЦ) программных и технических систем с использованием информационных, интеллектуальных и сервисных ресурсов Интернет.

Приводятся средства автоматизации программ на основе сборки Веб-систем и сайтов из ресурсов в современных ЯП (C, C++, Ruby, Python,, Ada, и др.) с помощью операций: make, config, build Grid, VS.Net, javaE и приведен экспериментальный вариант сборки ядра OS Linux для применения в автомобильной, авиационной и в предметных областях знаний (физика, биология, медицина, математика и др.).

Доктор физ.-мат наук, профессор Е.М. Лаврищева

Глава 1

Модульный принцип в языках программирования и задачи сборки разноязыковых модульных элементов

В период 1975 года в стране был создан вариант IBM-360, названный ЕС ЭВМ. На этой ЭВМ была сделана система автоматизации производства программ (АПРОП), предназначенная для автоматизированного изготовления сложных программ их модулей, записанных в ЯП. Исходные модули преобразовывались в объектный вид с помощью соответствующих трансляторов на ЯП.

Модуль в объектном виде – это описание исходного текста для преобразования в машинный код с дополнительной информацией, используемую при сборке оттранслированных модулей. Такие модули состоят из словаря внешних символов (ESD), текста (TXT), словаря перемещаемых символов (RLD) и записи конца модуля (END). ESD содержит информацию для Редактора связей, необходимую для объединения модулей. TXT содержит текст модуля: команды, константы, поля. Словарь RLD содержит ссылки на адресные константы, значения которых зависят от размещения загрузочного модуля в оперативной памяти.

Поскольку разные трансляторы создают объектный модуль однотипно, они освобождены от завершающего этапа изготовления рабочей программы. Эту функцию выполняют Редактор связей и Загрузчик. Редактор связей создает из объектных модулей (либо объектных и загрузочных) загрузочный модуль, в котором разрешены межмодульные связи. Загрузочный модуль становится готовым к выполнению после настройки его адресных констант по месту загрузки.

В состав одного загрузочного модуля, как правило, входят несколько модулей, каждый из которых может вызывать друг друга. Модуль, который передает управление другому, называется *вызывающим,* а модуль, которому передается управление, называется *вызываемым.* В одной программе один и тот же модуль может использоваться в качестве в качестве вызывающего и вызываемого. Элементы загрузочного модуля объединяются на основе реализации связей по управлению и по данным. Связь по управлению заключается, с одной стороны, в сохранении регистров вызывающего модуля и в передаче управления вызываемому и, с другой стороны, в восстановлении регистров и возврате управления вызывающему модулю. Связь по данным заключается в обеспечении передачи параметров из вызывающего модуля вызываемому, и наоборот, а также в совместном использовании общих наборов и областей данных.

Каждый компилятор формирует соответствующие команды для обеспечения соглашений о связях и поэтому в рамках одного языка не возникает никаких проблем объединения, т. е. связь происходит автоматически. Однако при объединении модулей, созданных разными компиляторами, возникают проблемы, которые состоят:

1) в особенностях организации передач управления между модулями на языках программирования высокого уровня;
2) в организации вызовов подпрограмм-функций;
3) в обмене данными через внешние носители и общие области;
4) в представлении компиляторами разных организаций данных.

1.1. Стандартные соглашения о связях модулей по управлению и по данным

Самым распространенным способом обмена данными между модулями является передача их через параметры с помощью оператора CALL. Стандартные соглашения о связях, обеспечивающие однозначную связь между модулями на уровне их объектного

представления, основываются на определенных правилах использования регистров общего назначения и регламентируют структуру области сохранения [1].

В соответствии с соглашениями о связях при передаче управления и данных вызываемому модулю используются пять общих регистров (14, 15, 1, 0, 13) и один регистр с плавающей точкой (0). Далее на примерах в языке Ассемблера объясняется использование каждого из регистров.

Регистры 14, 15. При обращении вызывающего модуля к вызываемому в регистре 15 должен быть задан адрес точки входа в вызываемый модуль, а в регистре 14 — адрес возврата в вызывающий модуль. Для наглядности действия этого механизма рассмотрим пример.

Пример 1.

```
A1          CSECT          — начало программной секции с именем A1;
            L  15, VCONST  — загрузка на регистр 15 адреса точки входа
                           вызываемого модули A2;
            BALR  14, 15   —передача управления по адресу,
                  …        содержащемуся в регистре 15 и занесение в
                           регистр 14 адреса команды в модуле A1,
                           следующей за BALR;
VCONST      DC    V(A2)    — адрес точки входа в модуль A2;
            …
A2          CSECT          — начало программной секции с именем A2;
            ….
            BR    14       — возврат в вызывающий модуль (в AI).
            …
            END
```

В данном примере модуль A1 вызывает A2, после окончания работы модуля A2 управление будет передано в модуль A1 на команду, следующую за командой BALR.

Регистр 1. Для передачи параметров в вызываемый модуль используется регистр 1. В нем размещается адрес начала списка адресов передаваемых параметров. Этот список представляет собой последовательность адресных констант, содержащих адреса соответствующих параметров. Старший бит последнего слова списка должен быть установлен в единицу.

На приведенном в примере 2 фрагменте программы демонстрируются загрузка регистра 1 и подготовка списка параметров.

Пример 2.

```
            LA LADSPIS     — загрузка адреса списка констант на регистр 1;
            …
ADSPIS      DC   A(X)      —  адрес поля X;
VCONST      DC   A(Y)      — адрес поля Y;

            DC   X ′ 80′   — установка старшего бита в 1;

            DC   AL3(Z)    — адрес поля Z;
            DS   …
X                          }   — поля значений
Y           DS   …         }   параметров.
Z           DS   …
```

Регистр 0. Если вызываемый модуль является подпрограммой-функцией, то результат передается через регистр 0 (общий для целых величин и с плавающей точкой для вещественного значения функции).

Регистр 13. При входе в вызываемый модуль необходимо сохранить содержимое регистров вызывающего модуля с помощью макрокоманды SAVE. По окончании выполнения вызываемого модуля содержимое общих регистров вызывающего модуля восстанавливается по макрокоманде RETURN. Для обеспечения сохранения регистров вызывающего модуля делается следующее:

а) в вызывающем модуле выделяется участок (новая область сохранения) памяти длиной 72 байта, выравненный по границе полного слова;

б) адрес старой области сохранения вызывающего модуля заносится во второе слово новой области в вызываемом модуле;

в) адрес новой области сохранения заносится в третье слово старой области сохранения и в регистр 13.

Пример 3.

```
A1          CSECT           — начало программной секции;
            …
SAVE        (14,12)         — сохранение регистров 14-12;

BALR        12,0            — занесение в базовый регистр 12 адреса
                              следующей команды;
USING       *,12            — установка базового регистра;

ST          13, AREA+4      — адрес старой области сохранения во 2-е
                              слово новой области сохранения;
LA          10, AREA        — адрес новой области сохранения
                              посылается в регистр 10;
ST          10,8 (13)       — адрес новой области сохранения в 3-м
LR          13,10           слове старой области сохранения и в
            …               регистре 13;
AREA        …               — новая область сохранения.
            DC  18F ´0´
            …
            END
```

Первоначальный вход в программу пользователя осуществляет управляющая программа ОС ЕС, которая обеспечивает область для сохранения содержимого своих регистров. В дальнейшем внутри программы предусматриваются все действия по заполнению и сохранению указанных регистров.

Рассмотрим программу, в которой содержатся все действия, связанные с использованием регистров и областей сохранения, причем модуль А2 не вызывает других модулей.

Пример 4.

```
A1          CSECT
            SAVE        (14,12)
            BALR        12,0
            USING       *,12
            ST          13, AREA+4
            LA          10, AREA
            ST          10,8 (13)

            LR          13,10
            …
            LA          1,ADSPIS
            L           15,VCONST
            BALR        14,15
            …
            L           13,4(13)
            RETURN      (14,12)
AREA        DC          18F ´0´
VCONST      DC          V(A2)
```

```
ADSPIS   DC        A(X)
         DC        A(Y)
         DC        X ′80′
         DC        AL3(Z)
X        DS        …
Y        DS        …
Z        DS        …
         …
         END
A2       CSECT
         SAVE      (14,12)
         …
         RETURN    (14,12)
         …
         END
```

Приведенную структуру программы должны иметь модули, написанные на языке Ассемблера. При работе с одним конкретным языком программирования высокого уровня проблемы установления связей не существует, поскольку соответствующий компилятор берет на себя решение этой проблемы. Однако при объединении разноязыковых модулей сказанное имеет силу лишь для модулей на языке Ассемблера. Для обеспечения передач управления из одного модуля на языке Ассемблера другому вызываемому модулю в этом же языке используется специальная макрокоманда CALL.

Объединение исходных модулей. Каждый из рассматриваемых языков программирования располагает средствами для обращения к часто повторяющейся части — модулю. К этим средствам относятся операторы обращения к процедурам, функциям и подпрограммам для случая их описания в исходных программах, а также оператор вызова, имеющий следующий общий вид:

CALL <имя вызываемого модуля> <список параметров>

Данный оператор дает возможность обращаться к модулям, находящимся вне вызывающей программы (они могут находиться либо на перфокартах отдельным пакетом, либо в библиотеке модулей). При этом исходный вызывающий модуль с оператором CALL и вызываемый им модуль могут независимо транслироваться и объединяться с помощью Редактора связей. В списке параметров этого оператора указываются передаваемые данные, которые в вызываемом модуле располагаются либо на месте соответствующих им фактических параметров, либо в общей области памяти.

Таким образом, оператор вызова позволяет лишь формально объединять модули, записанные на разных языках программирования. На практике в вызываемом или вызывающем модуле должны быть разработаны операторы, обеспечивающие устранение имеющихся различий в типах передаваемых данных и в способах их хранения.

1.2. Сопоставление языковых средств и особенностями и их представления компиляторами

Каждый язык программирования обладает некоторым множеством типов данных. При создании программ из модулей, записанных на разных языках программирования, приходится решать вопрос об эквивалентности представления информации. Объединив все типы данных, встречающихся в языках Фортран, ПЛ/1, Ассемблер, можно произвести условное разделение их на следующие подгруппы.

Эквивалентные данные. К ним относятся типы данных, содержащиеся в языках программирования, для которых внутреннее представление, сгенерированное компиляторами, идентично.

Косвенно эквивалентные данные. К этой подгруппе относятся типы данных, для которых нет эквивалентного описания в некоторых языках программирования, но с помощью языковых средств им можно поставить в соответствие некоторую группу данных, имеющую идентичное представление.

Неэквивалентные данные. К ним относятся те типы данных, для которых невозможно описать с помощью языковых средств группу данных, имеющих идентичное представление.

Ниже описываются типы данных, встречающиеся в языках Фортран, ПЛ/1 и Ассемблер, их организация в соответствующих языках программирования и устанавливается между ними отношение эквивалентности.

Язык Фортран. В этом языке имеются четыре типа данных: вещественные (REAL), целые (INTEGER), комплексные (COMPLEX) и логические (LOGICAL). По организации в памяти эти данные делятся на простые переменные и массивы, располагаемые в памяти статически, т. е. им выделяется память во время компиляции, которая постоянно присутствует при выполнении загрузочного модуля. В Фортране массив отображается в вектор, элементы которого располагаются в порядке изменения индексов слева направо (по столбцам), а в ПЛ/1 —наоборот. Поэтому согласование элементов массивов в модулях на ПЛ/1 и Фортране обеспечивается либо описанием массивов с противоположным следованием индексов, либо использованием противоположной индексации для элементов массива. Наряду с перечисленными типами данных в языке Фортран имеется возможность работать с символьными и шестнадцатеричными данными. Над ними могут производиться операции чтения, сравнения, пересылки с одного места памяти в другое и вывод на внешние устройства. Память под эти типы данных отводится так же, как и при распределении арифметических массивов. Операции пересылки и сравнения выполняются так же, как над арифметическими переменными и массивами. Для осуществления обмена с внешними устройствами символьные и шестнадцатеричные данные требуют задания спецификаций утверждения FORMAT.

Язык ПЛ/1. У этого языка больше разнообразных данных, чем у Фортран. В ПЛ/1 имеются следующие типы данных: вещественные (REAL), целые (BINARY), десятичные (DECIMAL), комплексные (COMPLEX), битовые строки (BIT), символьные строки (CHARACTER). При этом комплексные переменные в ПЛ/1 по сравнению с языком Фортран могут быть с плавающей точкой, десятичными и целыми (табл. 1.1).

По организации все эти данные делятся на переменные, и массивы. Кроме этого, в языке ПЛ/1 имеются более сложные организации данных — структуры, а также переменные и массивы типа метки. Память под данные отводится статически при компиляции модуля или динамически при каждом входе в него.

Многие языковые отличия практически не вызывают проблем при объединении модулей. Например, тип данных REAL FLOAT BINARY может быть вполне передан в модуль на Фортране, если ему сопоставлен тип REAL; соответствующая ему величина в дальнейшем должна использоваться без перевода.

Язык Ассемблера. Этот язык предоставляет пользователю возможность самому управлять размещением данных. Все данные условно можно разделить на две группы исходя из того, имеется ли эквивалент в языках высокого уровня или нет.

К первой группе относятся целые константы типа H и F, константы с плавающей точкой типа E и D, символьные поля типа C, битовые строки типа B и десятичные константы типа P *(см. табл. 1.1).*

Ко второй группе относятся шестнадцатеричные константы типа X и все адресные константы (A, V, Y, Q). Заметим, что аналогом адресных констант типа A служит переменная типа метки в языке ПЛ/1. Пользователь языка Ассемблера может управлять распределением памяти статически — с помощью команд языка DC, DS и динамически — с

применением макрокоманд GETMAIN, FREEMAIN [1]. Для рассматриваемых трех языков, помимо различий в типах, имеются отличия в задании длин отдельных элементов. Длина элементов может быть указана явно либо по умолчанию. Так, язык Ассемблер требует явного описания переменных и полей данных.

Таблица 1.1. Сопоставление типов данных в языках ПЛ/1, Фортран, Ассемблер

ПЛ/1		**Фортран**		**Ассемблер**		Внутреннее представление
тип данных	длина, байт	тип данных	длина, байт	тип данных	длина, байт	
REAL FIXED BINARY	2	INTEGER	2	H	2	Число с фиксированной точкой длиной 2 байта.
	4		4	F	4	Число с фиксированной точкой длиной 4 байта
REAL FLOAT BINARY	4	REAL	4	E	4	Число с плавающей точкой длиной 4 байта.
	8		8	D	8	Число с плавающей точкой длиной 8 байт.
REAL DECIMAL FIXED	~	_	_	P	~	_
REAL DECIMAL FLOAT	4	REAL	4	E	4	
	8		8	D	8	_
COMPLEX FLOAT DECIMAL	8	COMPLEX	8	_	_	Два числа с плавающей точкой длиной 4 байта.
	16		16	_	_	Два числа с плавающей точкой длиной 8 байт.
COMPLEX FIXED DECIMAL	~	_	_	_	_	Два десятичных числа
COMPLEX FLOAT BINARY	8	COMPLEX	8	_	_	_
	16		16	_	_	
CHARACTER	~	–	–	C	~	Символьная строка длиной от 1 байта до 32767

BIT	~	LOGICAL	–	B	–	Битовая строка длиной от 1 до 32767
	1		1	–	–	
	4	–	4	–	–	
	–		–	X	~	Последовательность шестнадцатеричных цифр

1.2.1. Способы описания простых переменных в языках программирования ОС ЕС

Для простых переменных трех рассматриваемых языков программирования проведено сопоставление по способу их описания и занимаемой памяти (в байтах). Результаты сопоставления представлены в табл. 1.2—1.9 по всем подгруппам типов данных.

Эквивалентные данные

1. *Число с фиксированной точкой и целые числа (*табл. 1.2).

Таблица 1.2

Фортран	ПЛ/1	Ассемблер	
а) INTEGER *4	REAL FIXED BIN (p.m.) 16 ⩽p ⩽31, q <p	DC DS DC	F ′число′ или 0F <4 байта любой информации>
б) INTEGER *2	REAL FIXED BIN (p.m.) 1 ⩽p ⩽15, q <p	DC DS DC	H ′число′ или 0H <2 байта любой информации>

В случае, *а)* переменная занимает 4 байта и выравнена по границе полного слова, в случае *б)* переменная занимает 2 байта н выравнена по границе полуслова. Выравнивание переменных в языке производят компиляторы специальными средствами. Так, в языке Ассемблера выравнивание переменных осуществляется с помощью констант соответствующего типа (F, H) или с помощью явного выравнивания (DS␣0F, DS␣0H).

2. *Числа с плавающей точкой* (табл. 1.3).

В табл. 1.3 переменная типа REAL (случай, *а)* занимает 8 байт и выравнена по границе двойного слова, а в случае б) — 4 байта и выравнена по границе полного слова.

Таблица 1.3

Фортран	ПЛ/1	Ассемблер	
а) REAL *8	REAL FLOAT BIN (p) 22 ⩽p ⩽53	DC DS	D ′число′ Или 0D

	REAL DEC FLOAT (p) 7⩽p ⩽16	DC	<8 байт информации, первый байт — порядок числа>
б) REAL *4	REAL FLOAT BIN (p) 1 ⩽p ⩽15 REAL DEC FLOAT (p) 1 ⩽p ⩽6	DC или DS DC	E ′число′ 0H <4 байта информации, первый байт — порядок числа>

3. *Логические переменные* (табл. 1.4).

Таблица 1.4

Фортран	ПЛ/1	Ассемблер	
а) LOGICAL *4	Используется целая переменная длиной 4 бита или битовая строка BIT (32)	DC F DS 0F DC XL4	′значение′ либо ′значение′
б) LOGICAL *1	Можно использовать битовую строку: BIT (8)	DC XL1	′значение′

Операциям над логическими переменными в языке Фортран соответствуют операции над битовыми строками в языке ПЛ/1. Поскольку в языке ПЛ/1 отсутствуют логические переменные, то в случае, *а)* может использоваться переменная целого типа длиной 4 байта, выравненная по границе полного слова, либо битовая строка, выравненная по границе полного слова. Логические переменные в случае *б)* занимают поле длиной 1 байт и для них выравнивание не производится.

Косвенно эквивалентные и неэквивалентные типы данных

1. *Комплексные переменные с плавающей точкой*.

Таблица 1.5

Фортран	ПЛ/1	Ассемблер	
а) COMPLEX *16	COMPLEX FLOAT BIN (p) 22 ⩽p ⩽53 COMPLEX FLOAT DEC (p) 7 ⩽p ⩽16	DC DC DS DC DC	D′Re числа′ D′I′m числа′ или 0D <8 байт информации под действительную часть, первый байт — порядок числа> <8 байт информации под мнимую часть, первый байт — порядок числа>
	COMPLEX FLOAT	DC	E ′Re числа′

б) COMPLEX *8	BIN (p) 1 ⩽p ⩽21 COMPLEX FLOAT DEC (p) 1 ⩽p ⩽6	 DS DC DC	E′I′m числа′ или 0F <4 байта информации под действительную часть, первый байт — порядок числа> <4 байта информации под мнимую часть, первый байт — порядок числа>

Комплексные величины представляют собой два последовательных числа: действительную часть (Web) величины и мнимую часть (Im) величины. При работе с комплексными величинами на языках высокого уровня операции над переменными производятся как над единым числом. В языке Ассемблера над числами, эквивалентными комплексным, операции производятся над каждой частью числа в отдельности по соответствующим правилам.

Комплексные величины в приведенной таблице состоят из двух чисел с плавающей точкой. В случае, а) переменная занимает два восьмибитовых поля, выравненных по границе двойного слова, в случае б) переменная занимает два четырехбайтовых поля и выравнена по границе полного слова.

2. *Комплексные переменные с фиксирующей точкой* (табл. 1.6).

Таблица 1.6

Фортран	ПЛ/1	Ассемблер
а) INTEGER *4 L (2) (L – имя комплексной переменной)	COMPLEX FIXED BIN (p) 16 ⩽p ⩽31	DC F′Re число′ DC F′I′m число′ Или DS 0F DC <4 байта под действительную часть> DC <4 байта под мнимую часть>
б) INTEGER *2 L (2) (L – имя комплексной переменной)	COMPLEX FIXED BIN (p) 1 ⩽p ⩽15	DC H′Re числа′ H′I′m числа′ или DS 0H DC <2 байта под действительную часть> DC <2 байта под мнимую часть>

Комплексным простым переменным с фиксированной точкой в языке Фортран не существует эквивалента. Но поскольку комплексное число состоит из двух частей одинакового представления н одинаковой длины, то эквивалентную структуру можно

создать с помощью описания массива, состоящего из двух элементов, как показано в табл. 1.5 в колонке для языка Фортран.

Комплексная переменная имеет две формы представления. Так, в случае, а) она занимает два последовательных слова, а в случае б) — два последовательных полуслова и выравнена на границе полуслова.

Обе части комплексной величины представляются в виде целых чисел; на языке Фортран и Ассемблер требуется выполнять действия над каждой частью в отдельности.

3. *Десятичные переменные* (табл. 1.7).

Таблица 1.7

Фортран	ПЛ/1	Ассемблер
а) Аналога нет	REAL FIXED DEC (p) $1 \leqslant p \leqslant 31$	DC Plan ′число′ $1 \leqslant n \leqslant 16$

Десятичные переменные представляют собой последовательность десятичных цифр (от 1 до 31) со знаком. Каждая цифра занимает полубайт. Длина десятичной переменной находится в пределах от 1 до 16 байт. Аналога в Фортране данные переменные не имеют. Другими способами описания величин, аналогичных десятичным переменным, язык Фортран не располагает.

4. *Символьные строки* (табл. 1.8).

Аналога символьным строкам в языке Фортран не существует. Однако для выполнения простейших операций (пересылки и сравнения) можно использовать любые переменные или массивы.

При обмене символьными строками в модулях, написанных на языках ПЛ/1 и Ассемблер, необходимо исходить из следующего соотношения: e = n × e′

В нем e и e′ изменяются в пределах, указанных в табл.1.8.

Таблица 1.8

Фортран	ПЛ/1	Ассемблер
Описание переменной или массива	CHAR (e) $1 \leqslant e \leqslant 32767$	DC uncle′ ′строка′ $1 \leqslant n$ $1 \leqslant e' \leqslant 256$

5. *Битовые строки* (табл. 1.9)

Таблица 1.9

Фортран	ПЛ/1	Ассемблер
Описание логических или целых данных	BIT (e) $1 \leqslant e \leqslant 32767$	DC noble′ ′строка′ $1 \leqslant n$ $1 \leqslant e' \leqslant 256$

Переменные, аналогичные битовым строкам языка ПЛ/1, в языке Фортран не содержатся. Однако можно моделировать операции с этими строками в языке Фортран с помощью логических пли арифметических операции, описывая соответствующим образом используемые переменные.

При обмене битовыми строками в модулях, написанных на языках ПЛ/1 и Ассемблер, необходимо учитывать соотношения:

[e/8]+ 1 = n× e′, если e не делится нацело на 8 и

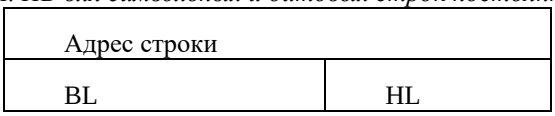
E/8 = n× e′, если e делится нацело на 8

Остальные типы данных, существующие в языках Фортран, ПЛ/1 и Ассемблер, либо не имеют эквивалента в другом языке, либо применяются очень редко. К ним относятся данные типа LABEL и TASK языка ПЛ/1, адресные константы языка Ассемблера и некоторые другие.

1.2.2. Внутреннее представление организации данных компиляторами

Как уже отмечалось выше, в языках ПЛ/1 и Фортран имеются различия в организации данных. Память под переменные и массивы в модулях на Фортране выделяется статически, поэтому особых сложностей при работе с данными не возникает. В языке ПЛ/1 под сложные типы данных (массивы, строки и т.д.) память может выделяться динамически. В связи с этим компилятор генерирует информационные векторы (ИВ), которые содержат сведения о строках или массивах. Ниже приведена структура ИВ для данных языка ПЛ/1.

А. ИВ для символьных и битовых строк постоянной и переменной длины.



BL – максимальная длина строки;

HL – текущая длина строки (строка постоянной длины BL= HL).

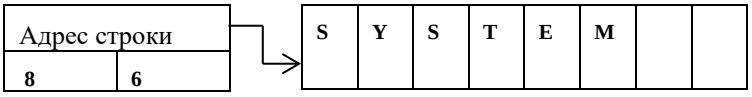
При использовании строк переменной длины в случае несовпадения текущей длины и максимальной программе доступны самые левые байты. Например, имеется описание строки

DCL X CHAR (8) VAR;

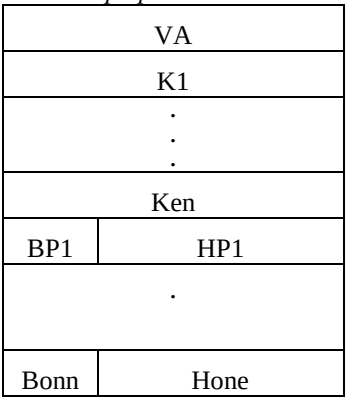
Данная строка описана, как строка переменной длины. Максимальная длина 8 байт.

После использования оператора X= 'SYSTEM';

Информационный вектор и поле имеют вид:



Б. ИВ для арифметических массивов.



где VA – виртуальный адрес массива, соответствующий элементу массива со всеми нулевыми значениями нижних границ размерностей.

Фактический адрес массива определяется по формуле:

$$A = VA + \sum_{i=1}^{n} K_i * Ho_{pi}$$

HP_i – нижняя граница i-й размерности массива; BP_i -верхняя граница i-й размерности массива. Коэффициент K_i вычисляется по следующей формуле:

$K_i\ (i = 1, 2, \ldots, n - 1) = K_{i+1} * (BP_{i+1} - HP_{i+1} + 1) \quad K_n = L,$

где L – длина одного элемента массива.

Пусть задан массив X(4,3,5,8), длина элементов которого составляет 4 байта, т.е. L=4, а коэффициенты K_i (i=1,2,3,4) вычисляются так:

$K_4=4,$

$K_3=K_4*(8–1+1)=4*8=32,$

$K_2=K_3*(5–1+1)=32*5=160,$

$K_1=K_2*(3–1+1)=160*3=480$

Для данного массива ИВ имеет следующую структуру (адрес первого элемента массива 16000):

15324	
480	
160	
32	
4	
4	1
3	1
5	1
8	1

Виртуальный адрес равен:

$$VA = A - \sum_{i=1}^{4} K_i * HP_i = 16000 - 480*1 - 160*1 - 32*1 - 32*1 = 15324$$

Память, занимаемая этим информационным вектором:

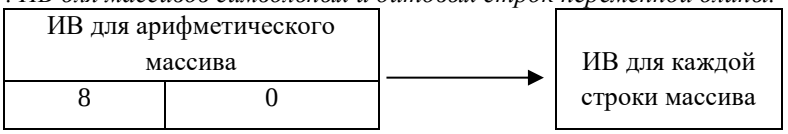
$S=4*(2*N+1) = 4*(2*4+1)=36.$

В. ИВ для массива символьных и битовых строк постоянной длины

ИВ для арифметического массива	
BL	HL

BL=HL и оба равны длине одной строки.

Г. ИВ для массивов символьных и битовых строк переменной длины.



ИВ для массива строк переменной длины адресует не элемент массива, а ИВ символьной или битовой строки этого элемента.

Д. ИВ для структур.

Данные типа «структура» представляются конкатенацией ИВ массивов и структур, а также адресных для простых переменных.

Пусть дано описание структуры в языке ПЛ/1:

```
DCL     1A,
        2B  CHAR (80),
        2C,
        3D BIN FIXED,
        3E (10,10);
```

ИВ для данного описания имеет вид:

ИВ символьной строки	B
Адрес	D
ИВ арифметического. массива	E

Кроме отличий в организации данных по ИВ в Фортране и ПЛ/1 имеются отличия во внутреннем представлении массивов. В модулях, написанных на языке Фортран, массивы располагаются в памяти таким образом, что при переходе к очередному элементу меняется самый левый индекс. Для массивов в модулях, написанных на языке ПЛ/1, первым меняется самый правый индекс.

Избежать отличий в индексации массивов и ИВ можно путем создания промежуточного модуля-связки (см. ниже) либо, используя средства языка, применять iSUB — определения н описатель DEFINED.

Фортран **ПЛ/1**

```
DIMENSION A (10,20,30)      DCL B (30,20,10) DEC FLOAT,
                                A (10,20,30) DEC FLOAT (6)
                                DEFINED B (3SUB, 2SUB, 1SUB)
```

Массив А имеет одинаковую индексацию в обоих модулях. Конструкция DEFINED iSUB позволяет избежать затрат памяти для буфера, необходимого при транспонировании массива (массивы А и В занимают одну и ту же память). При этом несколько увеличивается время доступа к элементам А.

Ниже указан способ, как избежать различий, связанных с ИВ в ПЛ/1:

- массив ПЛ/1, который нужно поставить в соответствие массиву Фортрана, объявляется базированным;
- используя встроенную функцию ADDR делается совмещение начала массива ПЛ/1 с началом массива в Фортране.

Пример 5.

Фортран	**ПЛ/1**
DIMENSION A (10,20,30)	PL: PROCEDURE (S) OPTIONS (MAIN);
...	DCL B(30,20,10) DEC FLOAT(6) BASED (P),
CALL PL (A)	S FIXED DEC(l,0),
...	A(10,20,30) DEC FLOAT(6) DEF
END	B (3SUB,2SUB,1SUB);
	P = ADR(S);
	/*МАССИВ А ДОСТУПЕН*/
	. . .
	END PL;

1.2.3. Структура общих областей

Кроме рассмотренного способа передачи данных через параметры (оператор CALL) языки программирования обеспечивают обмен данными через общие области.

Общая область — это статически распределенный (постоянно присутствующий во время выполнения программы) участок памяти, к которому может обращаться любой модуль независимо от того, на каком языке он написан. Память под общую область отводит Редактор связей во время создания загрузочной программы из совокупности общих областей отдельных модулей.

Каждый из рассматриваемых здесь языков программирования имеет средства для описания общих областей: в Фортране — оператор COMMON; в ПЛ/1—STATIC EXTERNAL; в Ассемблере — COM. Для общих областей сохраняются все особенности распределения данных, связанных с конкретным языком.

Они состоят в следующем:

обратное расположение элементов массивов в языке Фортран;

создание информационного вектора для каждой общей области я языке ПЛ/1;

выравнивание полей.

Обратное расположение элементов массивов рассматривалось выше, там же предложены способы избежать имеющиеся в этой связи отличий в модулях, записанных на языках Фортран и ПЛ/1.

Создание информационного вектора (ИВ). Для общей области, описанной средствами языка ПЛ/1, создается ИВ, аналогичный информационному вектору структуры. Этот ИВ размещается в начале общей области. Сами элементы обшей области расположены со смещением, равным длине ИВ. В общих областях, описанных средствами языков Фортран и Ассемблер, данные располагаются непосредственно с самого начала. Поэтому использование общих областей в модулях, написанных на языках ПЛ/1 и Фортран, оказывается затруднительным из-за отличий ИВ. Для их устранения возможны два способа.

1.Введение в общую область фиктивных переменных.

В модуле, описанном па языке Фортран, фиктивные величины занимают в общей области участок памяти, равный длине ИВ. Эти величины должны описываться в начале общей области. ИВ общей области (структуры) состоит из конкатенации адресных констант простых переменных и ИВ сложных конструкций (строк, массивов н др.).

Поэтому общая длина ИВ общей области в байтах может быть вычислена по формуле:

$$L = 4 \sum_{i=1}^{n} L_i$$

Здесь n — число переменных и массивов, входящих в общую область;

L_i может принимать следующие значения:

1 — для простых переменных;

2m+1— для арифметических массивов (m — число размерностей);

2 — для символьных п битовых строк;

2m+2 — для массивов и символьных и битовых строк постоянной длины;

$2m + 2 + 2 * \prod_{j=1}^{m}$ (Lbj—LHj+1) — для массивов символьных строк переменной длины (Lbj и LHj — соответственно верхняя и нижняя граница j-й размерности).

Рассмотрим этот способ для следующего описания:

```
DCL     1A STATIC EXTERNAL
        2B,
        2C (10,10),
        2D (20) CHAR (4);
```

Здесь B — простая переменная, C — арифметический массив, P — массив символьных строк постоянной длины. Поэтому $L_1 = l$, $L_2 = 5$, $L_3 = 4$. Длина ИВ общей области:

L= (1 +5 + 4)*4= 40.

При использовании общей области в модуле, написанном на языке Фортран, необходимо указать следующие описания:

```
COMMON /A/ FIKT, B, C, D
REAL*4 FIKT (10), B, C (10, 10)
INTEGER * 4D (20)
```

Структура общей области для данного описания будет:

Адрес B	ИВ массива C	ИВ массива D	Переменная B	Массив C	Массив строк D
0	4	24	40	44	444

2. Применение промежуточного модуля.

Один из способов устранения отличий в ИВ состоит в применении промежуточного модуля, написанного на языке Ассемблера. Его параметрами являются: адрес общей области, который можно получить из словаря ESD загрузочного модуля; длина общей области; длина ИВ общей области.

Для совмещения общей области в модуле, написанном на языке Фортран, перед вызовом последнего производится циклический сдвиг влево на длину ИВ общей области. Перед возвратом общей области в модуль на языке ПЛ/1 производится циклический сдвиг вправо на то же число байт.

Выравнивание полей. Для избегания различий, связанных с выравниванием, рассмотрим различные способы.

1. Описание общей области через атрибут UNALIGNED языка ПЛ/1. Этот способ является наиболее простым, хотя не выровненные данные будут требовать больше времени при обращении к ним. Аналогичного атрибута в языке Фортран не содержится.

2. Автоматическое выравнивание полей производится посредством описания общей области атрибутом ALIGNED в языке ПЛ/1 Аналогичная возможность имеется в языке Фортран. Предполагается описание и размещение данных в списке оператора COMMON в порядке уменьшения их длин. При этом описание данных в языке Фортран должно производиться, например, в следующем порядке:

COMPLEX * 16, COMPLEX * 8, REAL * 8,
REAL * 4, INTEGER * 4, LOGICAL * 4,
INTEGER * two, LOGICAL * one.

3. Первый и второй способы могут быть представлены в языке Ассемблера.

1.2.4. Особенности передачи управления в модулях на ЯП высокого уровня

При передаче управления из модуля в модуль, записанных па языках программирования высокого уровня, требуется устанавливать «среду», в функции которой входит инициация программ обработки прерываний и аварийных завершений, установка регистров, содержащих адреса областей, использующихся на протяжении выполнения задачи и др. Поэтому при организации вызова модулей, записанных на разных языках, необходимо установить среду вызываемого модуля. Рассмотрим механизм установления среды для модулей, написанных на языках Фортран и ПЛ/1.

Установка среды в языке Фортран. Среда устанавливается модулем 1BCOM#, который находится в библиотеке компилятора с Фортрана. Помимо установки среды, этот модуль выполняет операции ввода-вывода, а также некоторые другие функции. Для идентификации выполнения конкретной функции модуль IBCOM# имеет несколько точек входа, которые характеризуются смешением от начала программной секции. Точка входа для установления среды имеет смещение, равное 64.

Пример 6.

Вызов модуля на языке Фортран (F1), из модуля на Ассемблера (A1).

```
A1          CSECT
            SAVE        (14,12)
            BALR        12,0
            USING       *,12
            ST          13, AREA+4
            LA          10, AREA
            ST          10,8 (13)
            LR          13,10
            ...
            L           15,VCONST1
            BAL         14,64 (15)
            L           15,VCONST2
            BALR        14,15
            ...
            L           13,4 (13)
            RETURN      (14,12)
            ...
AREA        DC          18A(0)
VCONST1     DC          V(IBCOM#)
VCONST2     DC          V(FI)
            END
            SUBROUTINE F1
            ...
            RETURN
            END
```

В этом примере модулю F1 не передаются параметры. В противном случае строится список адресов передаваемых параметров, а адрес начала списка загружается в регистр 1 (см. п. 1.1).

Необходимо отметить специфику обращения к модулю 1BCOM#. Управление передается не в начало программной секции, а в точку входа, не имеющую внешнего имени и характеризующуюся только смещением. Поэтому установка среды языка Фортран может производиться только из модулей, написанных в языке Ассемблера.

Создание среды в языке ПЛ/1. Установка среды обеспечивается с помощью управляющих секций IHENTRY и IHEMAIN (которые генерируются компилятором ПЛ/1) и библиотечного модуля IHESAP (IHETSA в мультизадачной среде).

Секция IHENTRY имеет следующую структуру:

IHENTRY	
	15, = V($IHESAP_X$) 15

Здесь $IHESAP_X$ – точка входа в модуль IHESAP (для мультизадачной среды точка входа IHETSAx). Выбор конкретной точки, проставляемой в модуле IHENTRY, выбирается компилятором языка ПЛ/1 в зависимости от мультизадачности (опция TASK в PROC); значения опции OPT компилятора и наличия поля PARM на этапе выполнения (информация из этого поля может обрабатываться модулем, записанным на языке ПЛ/1).

Точки входа могут иметь значения, приведенные в табл. 1.10.

Таблица 1.10

Условие		Точка входа	
OPT = 29	PARM	однопрограммная среда	мультизадачная среда
0	Присутствует	IHESAPA	IHETSAP
1 или 2	Присутствует	IHESAPB	IHETSAP
0	Отсутствует	IHESAPC	IHETSAA
1 или 2	Отсутствует	IHESAPD	IHETSAA

Управляющая секция IHEMAIN имеет следующую структуру:

IHEMAIN
DC V(XXXXX)

Здесь XXXXX – имя первого выполняемого модуля в среде ПЛ/1. Вызов этого модуля производится из секции IHESAP следующим образом:

```
L 15, = V(IHEMAIN)   — загрузка в регистр 15 адреса секции
                       IHEMAIN;
L15, 0 (15)          — загрузка в регистр 15 адреса
                       выполняемого модуля;
BR 15                — передача управления.
```

Структурно каждый модуль, записанный на языке ПЛ/1, состоит из пролога, тела модуля и эпилога. В функции пролога и эпилога входит распределение и освобождение динамической памяти, осуществляемое обращением к модулю IHESAP (точки входа IHESADA и IHESAFA соответственно).

Схематически установление среды ПЛ/1 и вызов модуля (P1) на рис. 1.

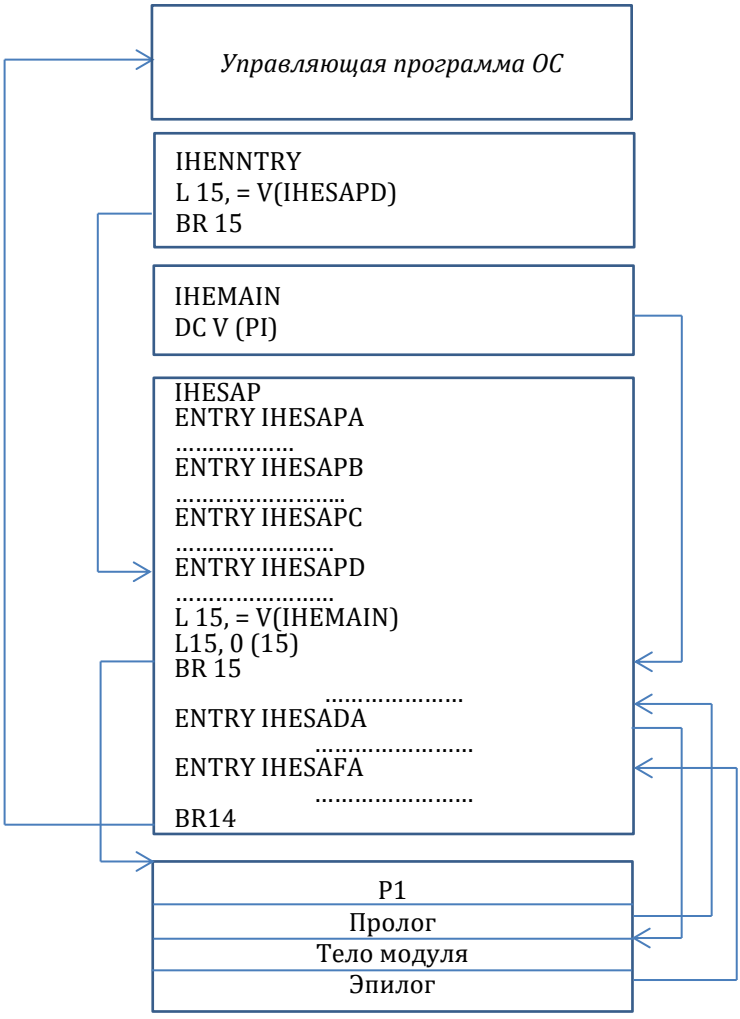


Рис.1. Установление среды ПЛ1 и вызов Модуля P1

При обращении к точке входа IHESAFA из эпилога корневого модуля управление не передается обратно, а возвращается к управляющей программе ОС или к программе, вызвавшей корневой модуль.

При установлении среды компилятора ПЛ/1 и вызова модуля программист может смоделировать секции IHENTRY и IHEMAIN самостоятельно. Проиллюстрируем это на примере 7.

Пример 7.

```
A1          CSECT
            SAVE        (14,12)
            BALR        10,0
            USING       *,10
            ST          13, AREA+4
            LA          9, AREA
            ST          9,8 (13)
```

```
                    LR          13,9
                    …
                    LA          15,VCONST
                    BALR        14,15
                    …
                    L           13,4 (13)
                    RETURN      (14,12)
AREA                DC          18A(0)
                    DC          V(IHESAPD

                    DS          0D
IHEMAIN             CSECT
                    DC          V(P1)
                    END
P1:                             PROC OPTIONS (MAIN);
                    RETURN;
                    END P1;
```

В примере 7, модуль A1, написанный на языке Ассемблера, вызывает модуль P1, написанный на языке ПЛ/1. Для передачи параметров может быть применен тот же аппарат, который описан выше.

В приведенных выше примерах рассмотрена связь типа Ассемблер — Фортран и Ассемблер — ПЛ/1. Учитывая, что для связей типа Фортран — Ассемблер и ПЛ/1 — Ассемблер трудностей при передаче управления не существует (для модулей, написанных на языке Ассемблера, понятие среды отсутствует), аналогично строятся связи Фортран — Ассемблер — ПЛ/1 я ПЛ/1— Ассемблер — Фортран.

Рассмотрим связь между модулями F1 (на языке Фортран) н PI (на языке ПЛ/1) с помощью промежуточного модуля A1, написанного на языке Ассемблера.

Пример 8.

```
                    SUBROUTINE F1
                    REAL * 4 A (10, 10), B (10)
                    …
                    CALL A1 (A,B)
                    …
                    RETURN
                    END
A1                  CSECT
                    SAVE        (14,12)
                    BALR        10,0
                    USING       *,10
                    ST          13, AREA+4
                    LA          9, AREA
```

```
                ST          9,8 (13)
                LR          13,9
                L           2,0 (1)
                L           3,4 (1)
                LA          3,0 (3)
                S           2, K1 получение виртуального
                            адреса массива A
                S           3, K2 получение виртуального
                            адреса массива B
                ST          2, IVA
                ST          3, IVB
                CALL        TRAN A обращение для
                            транспортирования массива A
                CALL        IHESAPD, (IVA, IVB), VL = 1
                CALL        TRAN обратное
                            транспортирование массива A
                L           13,4 (13)
                RETURN      (14,12)
AREA            DC          18A(0)
IVA             DC          A(0)
                DC          A(40)
                DC          A(4)
                DC          AL2 (10,1)
                DC          AL2 (10,1)
IVB             DC          A(0)
                DC          A(4)
                DC          AL2 (10,1)
K1              DC          A(44)
K2              DC          A(4)
                END
P1:             PROC (A,B)  OPTIONS (MAIN);
DCL             (A10,10),   B(10) ;
                ...
                END P1 ;
```

В этом примере используется секция IHEMA1N, генерируемая компилятором при описании опции MAIN. Эта секция содержит точку входа в модуль P1. Управление передается на точку входа IHESAPD в модуле IHESAP. Для передаваемых массивов A и B модуле A1 строятся информационные векторы IVA, IVB. Константы K1 и K2 необходимо отнять от фактических адресов массивов для получения виртуальных адресов. Виртуальные адреса заносятся в ИВ.

Установка среды в языке Ассемблера. Для модулей, написанных на языке Ассемблера, компилятор не создает среды. Чтобы обеспечить нормальное функционирование программы для обработки программных прерываний и аварийных ситуаций, пользователь сам должен включать в свою программу соответствующие средства. К ним относятся макрокоманды Ассемблера SPIE, STAE, ABEND и др. Используя различные варианты этих макрокоманд, программист сам управляет средой в процессе

выполнения программы, устанавливая или отменяя ее в зависимости от создаваемых условий.

Открытая, полузамкнутая и замкнутая среды. Выше был рассмотрен механизм установки среды для модулей, написанных на языках Фортран, ПЛ/1 и Ассемблер. Для этих трех языков среда создается по-разному в зависимости от механизма ее установки.
Рассмотрим разные типы среды, которые будут использоваться в дальнейшем при определении понятия агрегат.

Открытой средой называется среда, которая создается различными вариантами макрокоманд Ассемблера, применяемых в модулях на языке Ассемблера. Механизм задания среды определяется по усмотрению пользователя. Требуемый вариант макрокоманд задания среды пользователь располагает в любом месте исходного модуля, в чем и состоит открытость среды.

Полузамкнутой средой называется среда, которая создается с помощью модуля IBCOM# в программах на языке Фортран. При этом место программы, где задается среда, не определяется однозначно. Для программ на Фортране, не содержащих ввода-вывода прерываний, среда может не создаваться. Вызов модуля, написанного на Фортране, и установка среды представляют разные, не связанные между собой функции. Пользователь сам задает среду, но устанавливается она стандартно, что и определяет ее полузамкнутость.

Замкнутой средой называется среда, которая устанавливается в модулях на ПЛ/1 с помощью стандартных программ компилятора, причем место задания среды определяется вызовом модуля на ПЛ/1, т. е. установка среды в языке ПЛ/1 и вызов самого модуля представляет собой неразрывное целое и считается одной операцией. Все три типа механизма установки среды можно представить в виде следующей таблицы:

Таблица 1.11

Способ задания	Необходимость задания	
	всегда	не всегда
Стандартный	Замкнутая среда для ПЛ/1	Полузамкнутая среда для Фортран
Произвольный	—	Открытая среда Ассемблера

1.2.5. Организация вызовов подпрограмм-функций

При вызове подпрограммы-функции, написанной на языке Фортран, результат функции передается через регистр 0 (общий для целых и логических величин и с плавающей точкой для переменных типа REAL). При передаче комплексных значений используется 2 регистра с плавающей точкой — 0 и 2.

Механизм вызова подпрограмм-функций, написанных на языке ПЛ/1, несколько иной. При компиляции процедуры-функции компилятор ПЛ/1 строит «дополнительный» аргумент, т. е. добавляет в список аргументов адрес еще одного аргумента, длина и тип которого соответствуют типу процедуры-функции, и через него возвращает результат.

Сказанное выше показывает, что при вызове подпрограммы функции, написанной на языке, отличном от языка вызывающей модуля, промежуточный модуль на языке Ассемблера необходим.

В случае, если вызывающий модуль написан на языке ПЛ/1, а подпрограмма-функция — на языке Фортран, в функции промежуточного модуля входят:

выбор значения подпрограммы-функции из регистра 0 (общего или с плавающей точкой);

занесение этого значения по адресу «дополнительного» аргумента (последнего по списку).

Если вызывающий модуль написан на языке Фортран, а под программа-функция — на языке ПЛ/1, то модуль на Ассемблере должен выполнить следующие функции:

построение нового списка параметров, включая «дополнительный»;

выделение памяти под поле «дополнительного аргумента»;

после вызова подпрограммы-функции выборку из поля «дополнительного» аргумента и занесение его на регистр 0 (общий или с плавающей точкой).

Ограниченность типов подпрограмм-функций в языке Фортран накладывает ограничение на их совместное использование с модулями, написанными на языке ПЛ/1. Например, подпрограммы-функции, написанные на языке ПЛ/1, своим результатом могут иметь значение символьной или битовой строки любой длины. Подпрограммы-функции, написанные па языке Фортран, не обладают таким свойством.

Библиотеки программ компиляторов с ЯП ОС. Каждый компилятор с языка программирования в ОС ЕС имеет свою библиотеку программ, в состав которой входят два вида подпрограммы. К первому из них относятся подпрограммы, реализующие такие средства языков, как элементарные функции, операции обмена с внешними устройствами, управление распределением памяти и т. п. Ко второму виду относятся подпрограммы (установления среды, обработки ошибочных ситуаций и др.), вызванные спецификой взаимодействия с ОС ЕС и обеспечивающие функционирование изготовленных компиляторами программ. Исходя из того, что операционная среда, в которой такие программы будут работать, одна, а обеспечение взаимодействия с нею каждым компилятором осуществляется разными подпрограммами, общий объем комплекса программ существенно возрастет, так как в каждой компоненте будут присутствовать своп подпрограммы.

Ссылки на такие подпрограммы делаются каждым компилятором по-разному (явно или неявно). Так, для элементарных функций компилятор с Фортрана формирует к ним явное обращение, а компилятор с ПЛ/1 — неявное, пользуясь для функции SIN, EXP и SQRT соответственно следующими внешними ссылками: IHESNSS, IHEEXS0 и IHESAS0.

1.2.6. Совместное использование наборов данных

В языках ПЛ/1 и Фортран различают потока-ориентированный и записи-ориентированный ввод-вывод. При записи-ориентированном вводе-выводе запись из буфера переносится на внешнее устройство как единое целое без преобразования. При потоко-ориентированном вводе-выводе запись, находящаяся в буфере, рассматривается как непрерывная последовательность (поток) символов, и единицей обработки является символ. Символы обрабатываются последовательно, и при этом чаще всего происходит преобразование. Ниже приводится список операторов ввода-вывода для языков Фортран 11 ПЛ/1 и устанавливается между ними эквивалентность в 12).

Таблица 1.12

Поток ориентированный ввод-вывод		Запись-ориентированный ввод-вывод	
ПЛ/1	Фортран	ПЛ/1	Фортран
GET EDIT	READ + FORMAT	READ	READ бесформатный
PUT EDIT	WRITE + FORMAT	WRITE	WRITE бесформатный
GET DATA	READ +	REWRITE	Нет эквивалента

	NAMELIST		
PUT DATA	WRITE + NAMELIST	LOCATE	Нет эквивалента
GET LIST	Нет эквивалента	—	—
PUT LIST	Нет эквивалента	—	—

Следующий пример иллюстрирует одни из способов совместимого использования наборов данных:

Пример 9.

```
      SUBROUTINE F1
      INTEGER *2 I (50)
      …
      WRITE (3,1) (I(J),J=1,50)
1     FORMAT (50A2)
      …
      END
```

Массив I записывается в набор данных со ссылочным номером 3. Он может быть считан следующей программой, написании на языке ПЛ/1:

```
P1:       PROC OPTIONS (MAIN);
          DCL 1(50) BIN FIXED(I5),
          SET FILE INPUT STREAM;
          …
          GET FILE (SET) ED1T(I) (50A(2));
          …
          END PI;
```

Принцип формирования имени DD-предложения в модулях на языках Фортран и ПЛ/1 различен. Поэтому для описания совместно используемого файла требуется указание двух DD-предложений.

Для примера 9 необходимо задать описание следующих DD-предложений:

```
//    FT03F001    DD  UNIT=SYDRA, SPACE = (TRK,5),
//                DCB = (DSORG=PS,BLKSIZE=100,RECFM = F
//    SET         DD  DSN = *. FT03F001
```

В этом примере предполагается, что набор данных использует только в одном шаге задания, на котором выполняется программ, состоящая из модулей F1 и PI.

1.3. Структура программ, создаваемых из модулей

Операционная система ОС ЕС позволяет строить программы трех типов структур: простой, оверлейной и динамической. Возможна также комбинация нескольких структур. Каждая из этих структур имеет свои достоинства и недостатки. При этом большинство

недостатков появляется при создании программ с использованием языка программирования высокого уровня.

Простая структура. Это наиболее распространенная структура программ. Перед выполнением, программа такой структуры полностью размещается в оперативной памяти и в процессе своей работы не требует дозагрузки каких-либо модулей, как это требуется в программах с оверлейной и динамической структурами. Создание программ простой структуры обеспечивается с помощью оператора CALL в языках высокого уровня, а в языке Ассемблера — макрокомандой CALL и командами передачи управления. После перевода исходного текста в объектный вид имена вызываемых модулей н имена точек входа заносятся в словарь внешних символов (ESD) объектного модуля. На основе ESD, Редактор связей выбирает нужные модули, н формирует единый загрузочный модуль, готовый к выполнению. В нем все адресные константы устанавливаются относительно его начала. Окончательная настройка на фактические адреса осуществляется в момент загрузки модуля в основную память.

Таким образом, создание программы простой структуры обеспечивается информацией, содержащейся в ESD объектных модулей.

Отметим недостатки в работе с программами простой структуры:

1) требование наличия максимального объема основной памяти для загрузки программ;

2) дублирование в памяти модулей, выполняющих аналогичные функции, и модулей, которые созданы разными компиляторами для установки среды.

Если первый недостаток в основном исчезнет па машинах с виртуальной памятью, то второй будет иметь место при объединении модулей, написанных на разных языках. Поскольку транслятор с каждого языка программирования высокого уровня располагает своей библиотекой стандартных подпрограмм, реализующей обработку прерываний, создание «среды», выполнение операции ввода-вывода и др., то эти подпрограммы постоянно присутствуют в памяти как составная часть загрузочного модуля, создаваемого Редактором связей. Все это и создает предпосылки для дублирования в памяти загрузочного модуля эквивалентных программ. В качестве примера можно привести модуль идентификации «среды» из библиотеки компилятора ПЛ/1 с именем IHESAP (занимает 2,5К основной памяти) и аналогичный ему модуль IHCECOMH (занимает около 4К основной памяти) из библиотеки компилятора Фортран.

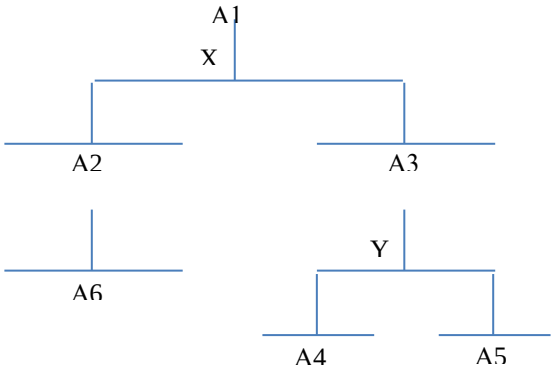
Таким образом, для программ средней сложности в каждом из языков программирования в основном требуется 20К памяти под служебные программы. При объединении двух модулей на разных языках эта память почти удваивается. Таким образом, при объединении модулей на языках ПЛ/1 и Фортран под служебные программы будет отведено около 40 К.

Оверлейная структура. С помощью оверлейной структуры программист может управлять расположением сегментов в основной памяти. Данную структуру удобно создавать при предварительном представлении программ в виде иерархических деревьев.
Понятие уровня при создании оверлейной программы отличается от общепринятого, т. е. считается, что два или более модулей находятся на одном уровне только в том случае, если они вызываются одним модулем.

Смысл оверлейной структуры заключается в том, что в каждый момент времени в основной памяти присутствует не вся программа, только некоторая ее ветвь, управление которой осуществляет сегмент, постоянно находящийся в памяти; он называется корневым сегментом.

Пример 10.

Пусть дана структура сегмента А1, представленная в виде графа. В ней два уровня (А2, А3), (А4, А5) и две ветви А2, А3.



При создании оверлейной структуры исходим из того, что модули А2 и А3 согласно определению, находятся на уровне I, а модули А4 и А5 — на уровне II. Модуль А6 не входит в список модулей уровня II, т.к. вместе с модулем А2 должен представлять собой единый загрузочный сегмент, вызов которого осуществляется передачей управления из модуля А2.

Для оверлейной структуры понятие уровня тесно связано с понятием точки загрузки. Последовательность точек загрузки как раз и определяет различные ветви программы. Для рассматриваемого примера выделены две точки загрузки с величинами X и Y. При работе оверлейной программы модули одного уровня (например, А2, А3) будут загружаться на один участок памяти; его длина определяется длиной наибольшей ветви.

Таким образом, модули одного уровня загружаются с одного и того же адреса оперативной памяти до вызова модуля того же уровня.

Так, для примера 10 можно образовать три ветви с точками загрузки X и Y:

A1 — A2 — A6; A1 — A3 — A4; A1 — A3 — A5,

Для образования оверлейной структуры в ОС ЕС должны быть составлены управляющие предложения Редактору связей, которые для примера 10, имеют, например, следующий вид:

```
 NSERT        A1
 OVERLAY      X
 INSERT       A2, A6
 OVERLAY      X
 INSERT       A3
 OVERLAY      Y
 INSERT       A4
 OVERLAY      Y
 INSERT       A5
```

Заметим, что в случае необходимости модули А3, А4 и А5 могут быть объединены в единый загрузочный модуль, тогда программа А1 будет содержать один уровень, а значит, и значит и одну точку загрузки (X). В процессе создания загрузочной программы Редактор связей присоединяет к ней модуль с именем SEGTAB, который осуществляет связь с управляющей программой ОС для загрузки очередного сегмента.

Используя оверлейную структуру, можно создавать программы, объем которых значительно имеющийся объем оперативной памяти. Вместе с тем отметим следующие недостатки:

1) все внешние имена и общие области должны находиться либо в корневом модуле, либо дублироваться в каждом загрузочном модуле;

2) снижается быстродействие при частом обращении и к сегментам уровня;

3) присутствуют в памяти модули, выполняющие аналогичные функции.

Первый недостаток возникает в случае, когда внешние имена и общие области определяются для модулей, находящихся на одном уровне. Поэтому их требуется выносить в такой участок памяти, содержимое которого не будет изменяться при смене сегмента программы; таким свойством обладает, например, корневой модуль.

Второй недостаток появляется в том случае, когда время выполнения некоторого сегмента значительно меньше, чем время, требуемое для его занесения в память. В этом случае такие сегменты целесообразнее делать простой структуры. Если время выполнения сегмента неизвестно или зависит от исходных данных, то желательно проверить работу данного сегмента отдельно перед включением его в основную программу. Причины третьего недостатка аналогичны ранее высказанным для простой структуры.

Для рассматриваемого примера 10 положим, что модуль А2 написан на языке ПЛ/ 1, а модуль А3- на языке Фортран. При создании загрузочного модуля Редактор связей помещает в него служебные программы компилятора ПЛ/ 1 для модулей А2 и служебные программы компилятора Фортрана для модуля А3. В результате появляется возможность одновременного присутствия в памяти модулей с различными именами, но выполняющими аналогичные функции, т.е. в этом случае затрачивается дополнительная память под идентичные служебные программы.

Динамическая структура. Эта структура характеризуется тем, что связь между элементами определяется в процессе выполнения программы. Различаются два вида динамической структуры: структура без образования подзадачи и структура с подзадачами.

В ОС ЕС программы с динамической структурой осуществляют динамический вызов модулей в память по мере необходимости. При этом каждый модуль предварительно редактируется и помещается в библиотеку загрузочных модулей. Динамическая структура образуется с помощью следующих макрокоманд языка Ассемблера:

LINK — для структуры без образования подзадач и ATTACH для структуры с подзадачами. В качестве одного из параметров каждой макрокоманде присутствует имя вызываемого модуля, а макрокоманды инициируют соответствующую программу супервизора, которая осуществляет поиск объекта в библиотеке загрузочных модулей и его занесение на свободный участок памяти. Затем передается управление данному загрузочному модулю; после выполнения память, занятая модулем, считается свободной; каждое повторное использование аналогично.

При создании программ с динамической структурой имеют место следующие недостатки:

1) снижается эффективность программы при частом обращении к динамическому объекту;

2) дублируются служебные модули в пределах одного языка программирования.

Первый недостаток отмечается в случае, когда время выполнения данного динамического объекта значительно меньше времени, требуемого на его загрузку, и, если объект вызывается довольно часто.

Второй недостаток свойственен всем языкам программирования высокого уровня в ОС ЕС. При создании программ простой и, оверлейной структуры модули, написанные на одном и том же языке высокого уровня, используют одни и том же язык высокого уровня, используют одни и те же служебные программы, которые подсоединяются в загрузочную программу. При создании динамической структуры вызываемый модуль должен быть предварительно отредактирован, так как макрокоманды LINK и ATTACH вызывают модули в загрузочном виде. В него будут включены все необходимые служебные модули, которые могут быть уже включенными в главную программу. Поэтому при занесении динамического модуля в память в выполняемой программе будут присутствовать две или более одинаковые части. Учитывая, что служебные модули суммарно могут занимать до 20

К памяти следует осторожно использовать динамическую структуру в программах, создаваемых из модулей, написанных на разных языках высокого уровня.

1.4. Модуль-связка как способ сборки разноязыковых модулей

Исходя из проведенного выше анализа возможностей языка программирования и ОС ЕС следует, что вопрос объединения разноязыковых модулей не решается однозначно. Он предполагает дополнительный объем работ, связанный с устранением программным путем указанных выше проблем. Одним из способов обеспечения связи разноязыковых модулей является разработка промежуточного модуля - *модуля-связки*, в котором разрешаются все проблемы, связанные неэквивалентностью типов данных и отличий в выходном коде компиляторов. Создание модульного посредника имеет место, когда необходимо связать пару исходных модулей, записанные разных языках программирования.

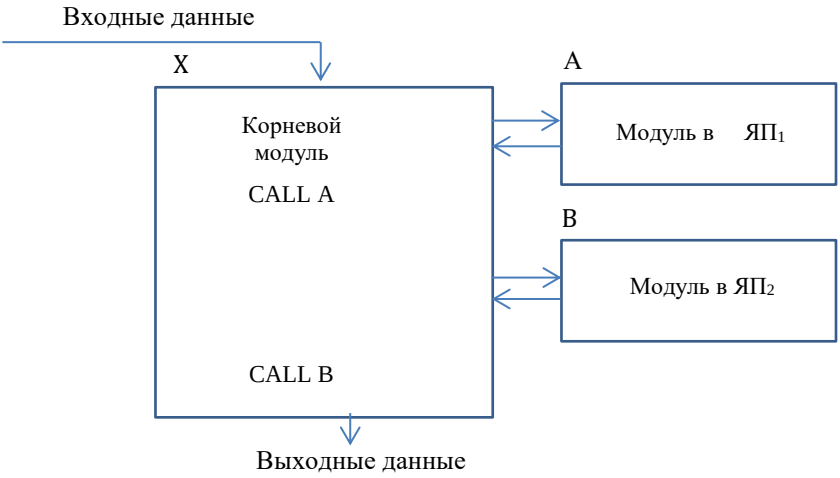


Рис. 2. Схема программы X с вызовом A, B

Для представленной схемы программы (рис. 2), практически должна быть использована схема, приведенная на рис. 3. В ней для двух вызываемых модулей (A, B) формируется два модуля-связи.

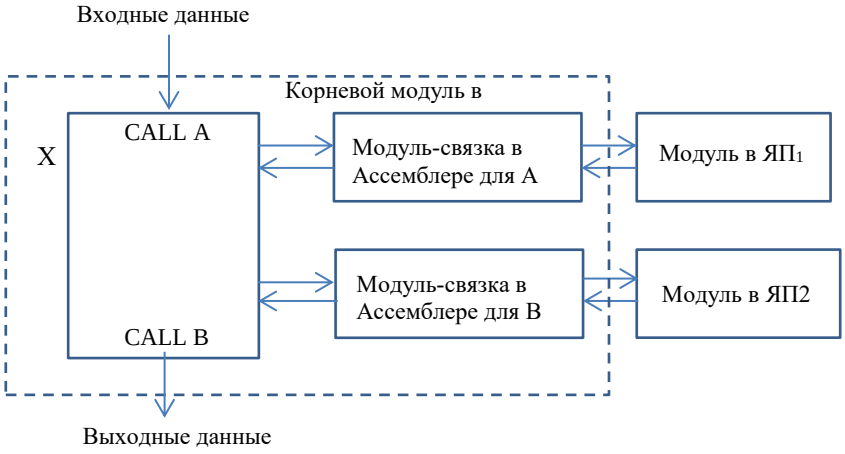


Рис. 3. Схема программы X с модулями-связками для A, B

Структура модуля-связки. В примере 4 п. 1.1 рассмотрены основные действия, связанные с установлением связей по управлению и по данным между вызывающим и вызываемым модулями, описанными на языке Ассемблера. В том случае, когда объединяются разноязыковые модули, то наряду с командами, приведенными в примере 4, в модуле-связке будут присутствовать команды преобразования типов и структур данных, т. е. общая структура модуля-связки будет иметь вид, приведенный на рис. 4.

	<Имя модуля-связки> CSECT	Имена макроопределений БМИ
Требования соглашений о связях	Сохранение регистров вызывающего модуля, выделение новой области сохранения, установка базового регистра, выделение памяти для функционирования модуля-связки	SAVEP
Несогласованность обращения	Изменение порядка следования передаваемых параметров	SCMOVE
Отличия языковых средств и выходного кода компиляторов	Создание ИВ. Формирование адреса массива по ИВ	ASPL 0 PLAS 1 2 3 4 5
Отличия языковых средств	Транспонирование массивов, передаваемых через параметры	TRAN
Отличия выходного кода компилятора	Преобразование общих областей	
Отличие выходного кола компиляторов и языковых средств	Преобразование совместно используемых наборов данных	
Отличия выходного кода компиляторов	Установка среды вызываемого модуля	ASFT ASPL
Требования соглашений о связях	Передача управления вызываемому модулю	CALLP LINKP
Отличия выходного кода компиляторов и языковых средств	Обратное преобразование совместно используемых наборов данных	
Отличие выходного кода компиляторов	Обратное преобразование общих областей	
Отличие языковых средств	Обратное транспонирование массивов, передаваемых через параметры	TRAN
Особенности модуля-связки	Освобождение динамической памяти	FREEMNP
Требования соглашений о связях	Восстановление регистров вызывающего модуля и возврат ему управления	RETURNP

Рис. 4. Общая структура модуля-связки

Количество модулей-связок в программе из n-разноязыковых модулей зависит от структуры программ и находится в пределах от 0 до n(n-1).

Экспериментальные данные показали, что в среднем для программы, состоящей из n/2 разноязыковых модулей, требуется n/2 модулей-связок.

Учитывая, что средний объем модуля-связки составляет 50—60 команд Ассемблера, общее число V команд, которые требуется разработать дополнительно к исходным модулям, составляет: V= (25/30)n.

В некоторых случаях (транспонирование массивов, преобразование ИВ больших размерностей и т.д.) требуется включить в состав программы соответствующие загрузочные модули, что увеличивает объем создаваемого программного продукта в среднем на 300 команд Ассемблера.

В главе 4 приводятся примеры конкретно создаваемых модулей-связок.

Глава 2

Способы автоматизированного связывания (объединения) разноязыковых модулей

К настоящему времени накопился опыт по реализации способ связи одноязыковых модулей [1, 18], систем программирования ориентированных на класс языков [16, 19], принципов создания модульных структур программ [4, 18] и т.д.

Вместе с тем проводились исследования и разработки способов, обеспечивающих разрешение проблем связи модулей, записанных на разных языках программирования, в частности для ЕС ЭВМ, IBM, Intel и др.

К ним относятся:

1) сопряжение модулей с помощью специальной программы или стандартного модуля-связки, настраиваемых на конкретные применения [12, 15];

2) прямого и обратного преобразования передаваемых данных-вызываемом и вызывающем модулях;

3) описания межмодульных связей с помощью специальных таблиц [13];

4) разработки дополнительного модуля (модуля-связки [9]) на языке Ассемблера и т.д.

Отличительной особенностью первых трех способов является то, что в них проблема связи решается частично. Рассматриваем в дальнейшем четвертый способ, реализованный в системе АПРОП, практически обеспечивает полную связь модулей, задаваемых в классе входных языков ОС ЕС. С помощью модуля-связки реализуются не только описанные в первой главе проблемы связи, и функции управления отладкой и выполнением цепочек модулей в модульной структуре.

Составление модулей-связок может быть осуществлено с помощью системы АПРОП либо самим разработчиком. В последнем случае пользователь языков программирования высокого уровня кроме языковых средств, должен знать еще и структуру выходного кода компиляторов. Также он должен знать назначение служебных программ, выполняющих функции установки среды, обращения к общим областям и наборам данных и т. д. Вместе с тем на пользователя возлагается выполнение всех шагов заданий в ОС ЕС для получения совокупности исходных модулей и модулей-связок готового программного продукта.

В рамках системы АПРОП реализована методика модульного изготовления программных агрегатов, выполняемая под управлением пользователя. Методика ориентирована на полный цикл работ, начиная с этапа функционального проектирования сложных программ с представлением их структур ориентированными взвешенными графами и кончая изготовлением программного продукта готового к исполнению.

Метод является аппаратом, основанным:

- на формализованных типах отношений между модулями [21];
- на наборе функций и операций, обеспечивающих способы манипулирования модулями и управление построением различных структур модульных программ;
- библиотеке межъязыкового интерфейса, реализующей проблему взаимосвязи модулей, записанных па разных языках программирования;
- банке модулей общего пользования, хранящем готовые модули, которые могут применяться различными категориями пользователей.

Пользователь системы освобожден от изучения упомянутых выше задач. Он должен разработать библиотеку функциональных модулей или использовать готовые из банка модулей и на их основе средствами системы объединять модули в требуемые структуры. Система берет на себя вес функции по формированию программного продукта, готового к исполнению.

Данная глава посвящена описанию методики построения сложных интегрированных агрегатов из исходных модулей, а также принципам программной реализации этой методики в системе АПРОП.

2.1. Элементарные объекты конструирования сложных программ

Конструирование программ из модулей заключается в выделении в проектируемой программе элементарных функционально независимых объектов и задании способов построения из них сложных интегрированных программных объектов. Элементарные объекты входят в класс основных конструктивных понятий метода конструирования. Определены формальные правила представления элементарных объектов и правила конструирования из них более сложных объектов. Понятие элементарного объекта включает описание его функции средствами любого языка программирования и спецификаций, определяющих правила формальной взаимосвязи данного объекта с другими. Элементарными объектами конструирования могут быть модули и макромодули.

Модуль — это функционально независимая именованная часть программы, реализующая некоторую функцию. Модуль является обобщением понятий процедур и функций, содержащихся в языке программирования. С математической точки зрения модуль представляет собой функцию, отображающую множество исходных данных во множество результатов.

Элементами множества исходных данных могут быть:

атомарные данные (простые переменные и числа различных типов, содержащихся в языках программирования;

массивы (атомарные данные, связанные отношением линейного порядка);

структурные данные — записи, файлы (производные массивов произвольной длины).

Модуль как функция обладает следующими свойствами:

- независимостью;
- повторной используемостью (это означает, что изменения в одном модуле не вызывают изменений в другом модуле);
- раздельной компиляцией;
- параметризацией.

Последнее свойство связано с определением основных характеристик модуля, задающих способ обращения на модуль для получения результатов.

Модуль имеет три вида представления: исходный, объекту и загрузочный.

Исходный вид — это описание модуля средствами языка высокого уровня, предназначенное для обработки компилятора.

Объектный вид — это представление после компилятора, содержащее выходной код и таблицы внешних символов.

Загрузочный вид — это представление одного или цепочки объектных модулей, готовых для исполнения на заданном наборе исходных данных.

Другим объектом конструирования является ***макромодуль***. Это текстовая заготовка, из которой генерируется описание модуля в исходном виде, ориентированное на конкретное применение. В тексте макромодуля содержатся операторы-шаблоны, инструкции, одного из языков программирования, параметры генерации и другие макросредства. Макромодуль имеет один вид представления — исходный. К нему применяется техника макрогенерации [14] для получения исходного вида модуля.

2.2. Стандартизация элементарных объектов

Технология ручной и автоматизированной сборки программы изделий из элементарных объектов требует, чтобы каждый конструктивный объект имел конструкторскую документацию в виде системы правил и ограничений на сам объект и на его сопряжение с

другими. Конструкторская документация содержит характеристики объекта, определяющие его с учетом программного и информационного объема, входов в него и выходов из него, а также тип перерабатываемых им данных. Описание такого рода характеристик объекта представляет собой паспорт объекта. Опыт разработки показывает, что класс паспортных данных объектов типа модуль в языке программирования является весьма общим и может быть приведен к единой стандартизованной форме для всех широко и пользуемых языков. Для обеспечения независимости описания модуля от его характеристик последние задаются в специальном информационном разделе. Таким образом, в описании модуля выявлены три части (рис. 5).

MOD

	Имя модуля
	Список формальных параметров
PASSPORT	
	Имя модуля LANG ARG … RES … MDL … .. DES описание параметров средствами ЯП
TEXT	
	Текст на ЯП
MEND	

Рис. 5. Описание модуля в языках программирования

В *первой части рис.5*, начиная со слова MOD, указываются имя модуля и прототип обращения к нему. Здесь же в виде комментария может указываться информация о возможностях модуля.

Вторая часть является паспортом. В нем перечисляется набор входных (ARG) и выходных (RES) параметров и их спецификации (раздел DES), списки внешних меток и модулей (EXT, MOD) и т.п.

Третья часть TEXT - это описание алгоритма модуля в ЯП. Заканчивается описание модуля служебным словом MEND.

Обязательной информацией при описании модуля будет:
оператор прототипа;
слово PASSPORT;
имя модуля;
название языка программирования.

Этой информации достаточно при объединении одноязыковых объектов. В разделе паспорта порядок следования его параметров произволен. Параметры ARG и RES содержат списки переменных, являющиеся соответственно входными и выходными параметрами модуля. Промежуточные величины MDL - идентификаторы, передаваемые как параметры из данного модуля в другие через оператор CALL. Вызываются модули оператором CALL.

Описание наборов данных IOL содержит сведения об именах и характеристиках наборов данных, передаваемых в другие модули. Спецификация DES содержит сведения о типе и структуре параметров н других величин; составляются спецификации в терминах языка LANG описания модуля.

Аналогично задается описание макромодуля. При этом вместо MOD употребляется MACRO, и среди параметров паспорта могут присутствовать параметры KPV, KPS и 1NC, которые служат для обозначения переменных периода генерации и их значений, а также объектов, вызываемых в процессе генерации текста модуля на основе текста макромодуля.

Подробное описание модуля и макромодуля содержится в [8] и в документации по системе АПРОП.

На основе краткого изложения структуры описания элементарных объектов и содержательного описания их характеристик можно сделать вывод о том, что это описание по своему внешнему виду стандартизовано. В нем по сравнению с принятым описанием ОС ЕС (третья часть) добавляются первая и вторая части, которые (до раздела DES второй части включительно) описываются для всех языков программирования идентично. Вследствие того, языки оперируют с различными типами и организациями данных, параметры прототипа специфицируются в разделе DES средствами тех языков, в которых представлен текст модуля.

Пример 11. Описание фрагмента модуля

```
MOD                ALFA
                   (A,B,C)
PASSPORT                    ALFA
                   LANG     PL1
                   ARG      A,B
                   RES      C
                   MOD      PROG (X,Y), FI(Z,N)
                   MDL      X,Y,Z,N <промежуточные величины>
                   DES      A REAL DECIMAL FLOAT (6,2),
                            B,C REAL DECIMAL PLOAT (6,2),
                            X CHARACTER (80), Y BIT (10),
                            Z COMPLEX BINARY FLOAT
                            N COMPLEX DECIMAL FLOAT;
TEXT
                   ALFA:    PROC (A,B,C) OPTIONS (MAIN) ;
```

```
                    DCL A,B DEC FLOAT, …
                    …
                    CALL PROG (X,Y) ;
                    …
          END       ALFA;
```

MEND

Если в тексте модуля используется массив, и он является параметром, то при его описании в паспорте только верхние пределы изменения индексов, считая пределом 1.

Например,

Описание в ПЛ/1	**Описание в паспорте**
DCL A(1:10,-5:4);	DES A(10,10)

Для модулей, написанных на языке Фортран, общие области специфицируются аналогично их описанию в исходном языке.

Например,

В языке Фортран	**В паспорте**
COMMON /A/B,	COM /A/B,

Поскольку один модуль может вызывать другой, то в списке параметров вызывающего модуля (см. пример 11) в описании MD должны указываться имена вызываемых модулей, которые могут быть с параметрами или без параметров. Аналогично описывают имена внешних меток, к которым обращается вызывающий модуль. Эта информация требуется для автоматической связи модулей по управлению, объединяемых в один программный агрегат. Связь по данным обеспечивается на основе раздела DES паспорта. При установлении связей предварительно осуществляется проверка правильности типов и организации передаваемых данных, а затем — автоматическая генерация модуля-связки.

2.3. Программный агрегат

2.3.1. Определение агрегата

Программный агрегат — это составной интегрированный объект, создаваемый из элементарных конструктивных объектов.

Программный агрегат имеет три вида представления:

- исходный,
- промежуточный;
- загрузочный.

Исходный вид — это представление при проектировании; оно включает набор элементарных конструктивных объектов и правил формирования из них агрегата.

Промежуточный вид — это внутрисистемное представление (называемое нами полуфабрикатом). Оно получается после проверки правильности задания на формирование агрегата и включает исходные элементарные объекты, сгенерированные системой, модули-связки и (или) корневой модуль, матрицу отношений (внутреннее представление графовой модели агрегата), а также управляющие предложения для компиляторов и Редактора связей.

Загрузочный вид агрегата получается из промежуточного и представляет собой готовый для исполнения программный продукт. Исходя из данных определений, условимся в дальнейшем термин «программный агрегат» использовать для исходного вида представления агрегата, задаваемого при проектировании, а термин — программный продукт — для загрузочного вида представления агрегата, изготовляемого системой для использования.

К программным агрегатам относятся: сегменты, программы, комплексы и пакеты программ.

Определим сначала среду агрегата, создаваемую системой АПРОП для обеспечения функционирования программного продукта. В среду агрегата входят: корневой модуль; модули-связки; таблицы внешних символов агрегата, по которым происходит связь, синхронизация и выполнение объектов, объявленных как подзадачи или динамические объекты.

В задачу корневого модуля входят: ввод-вывод информации, вызов и запуск элементов агрегата с помощью упомянутой выше таблицы внешних символов агрегата.

Создаваемая системой среда агрегата условно делится на:

- открытую;
- полузамкнутую;
- замкнутую.

Открытая среда агрегата имеет аналогию со средой, создаваемой компилятором с языка Ассемблера. В системе АПРОП статус открытой среды имеют отдельные модули, создаваемые для агрегата. Для получения загрузочного вида агрегата пользователь системы может по своему желанию применить два различных способа. При первом способе последовательно используются операторы транслирования и редактирования. В этом случае, в загрузочный модуль, создаваемый системой, элементы среды не входят. Во втором случае, модуль может быть представлен агрегатом из одного элемента. При обработке этого агрегата среда для него создается в зависимости от типа. Поэтому, как и для модулей в языке Ассемблера, открытая среда может быть установлена или нет по усмотрению пользователя. Это определяется дальнейшим использованием модуля.

Полузамкнутая среда агрегата имеет аналог среды и способ ее установки в языке Фортран. Если агрегат состоит из двух н более модулей, то полузамкнутая среда формально должна присутствовать, и только в том случае, если агрегат состоит из одноязыковых модулей, среда может не содержать элементов. Поэтому, как и для модулей в языке Фортран, полузамкнутая среда формально должна задаваться всегда, однако, в некоторых случаях, она не содержит элементов. При этом обращение к агрегату и задание среды являются отдельными функциями.

Замкнутая среда агрегата определяется аналогично тому как она определялась ранее для программ на ПЛ/1. В замкнутой среде всегда присутствует корневой модуль, определяющий функционирование среды, независимо от того, одноязыковые или разноязыковые модули входят в агрегат. Вход в модуль пользователя осуществляется только через корневой модуль среды. Обращение к агрегату и установка среды являются неразрывными функциями.

Сегмент — это совокупность модулей, записанных в одном или разных языках программирования и объединяемых в один загрузочный агрегат. Для сегмента создается полузамкнутая среда, но в случае, когда модули, из которых сегмент формируется, записаны на одном языке, среда для сегмента не создается.

Программа — это совокупность модулей и сегментов, объеденных по памяти и по очередности их выполнения. Для программы среда устанавливается системой всегда, и при этом она будет замкнутой. В случае, когда программа будет простой структуры и состоять из элементов, записанных на одном языке программирования, среда все равно появится. Ее составляет сгенерированный системой корневой модуль, имя которого будет отличаться от имени главного модуля в заданной совокупности исходных элементов.

В функции корневого модуля входят: вызов программных элементов, составляющих агрегат; передача данных через общую область и (или) общие наборы.

Комплекс — совокупность программ, которые оформляются самостоятельно. Для всего комплекса понятие среды отсутствует, оно фактически существует для входящих в него программ. Во время выполнения комплекса, среда меняется каждый раз, когда вызывается новая программа, входящая в комплекс. Вызов каждой программы комплекса осуществляется динамически макрокомандой LINK.

Пакет — это взаимосвязанный набор программных элементов, содержащихся в личной библиотеке модулей разрабатываемой предметной области. Элементы набора определяют системную и функциональную часть пакета. Для обеспечения функционирования элементов пакета система обеспечивает генерацию среды. В ее функции входят: прием и анализ входных сообщений, планирование цепочек модулей, корректировка данных и т. п.

Каждый из перечисленных типов программных объектов может быть ориентирован на динамическую, оверлейную структуры, а также на режим подзадач. Формируемые при этом элементы среды могут входить в область действия разных загрузочных модулей агрегата. В этом случае агрегат имеет несколько типов среды. Структура агрегатов рассматривается в п. 2.5.

2.3.2. Представление программных агрегатов

Методика модульного конструирования программ предполагает функциональное проектирование проблемы сверху вниз. Формой представления структур создаваемых программных агрегатов является ориентированный взвешенный граф (дерево). Вершинами графа являются имена модулей, а дуги задают типы отношений, которые могут существовать между модулями. Граф представляется так:

вершины обозначаются кружками; дуги изображаются линиями со стрелками, которые могут снабжаться дополнительными отметками (*, +, ¤, &, /), каждая из которых обозначает тип отношений между вызывающим и вызываемым модулями при выполнении.

Первая вершина соответствует главному модулю, в нее не входит стрелка; во все остальные входят дуги со стрелками и не изо всех вершин стрелки выходят.

Значение, которое каждый модуль принимает на заданном множестве исходных данных, определяется процедурой выполнения. На первом шаге исходные данные выбираются из исходного набора данных и задают начальное состояние памяти для главного модуля. На следующих шагах каждым модулем вырабатывается промежуточная информация, которая размещается в общей области главного модуля или передается через параметры и необходима для выполнения последующих модулей.

Взаимодействие модулей, предписываемое процедурой выполнения, регламентируется операциями:

отношения вызова, задаваемого оператором CALL в модулях на ЯП, определяющего имена модулей-преемников и способа передачи данных;

управления, которые задают только имена модулей-преемников, явно не предписывая способ передачи данных для их выполнения;

рекурсии.

Рассмотрим эти операции подробнее.

Отношение вызова – модуль А вызывает модуль В. Этот тип отношений характеризует связь модулей по управлению и по данным, осуществляемую через фактические параметры, и служит способом задания программ различной структуры, принятой в ОС ЕС [1]. Связь модулей может производиться следующими типами вызова:

Линейный (простой) вызов link А→В.

Этот вызов определяет передачу параметров из вызывающего модуля А в вызываемый В и служит основой построения программ простой структуры. Линейный вызов в ЯП задается

с помощью оператора CALL, а в системе АПРОП, он реализуется специальным макроопределением CALL (см. п. 3.2.4), разработанным для этих целей.

Динамический вызов A*→B

Этот вызов применяется для модулей с редким к ним обращением и инициирует динамическую загрузку модуля с внешней памятью в оперативную с последующим удалением его из памяти после завершения работы. Современные ЯП высокого уровня, в общем, не содержат средств для управления динамическим выполнением модулей, поэтому такой вызов моделируется либо обеспечивается средствами операционных систем. Так, в ОС ЕС динамическое выполнение модуля обеспечивается, например, макрокомандами LINK, LOAD, DELETE, а в системе АПРОП для этих целей используется макроопределение LINKP (см. п. 3.2.5). Данный тип отношений инициирует построение агрегата с динамически управляемыми объектами.

Оверлейный вызов A¤→B, C

Данный вызов инициирует совмещение программной памяти модуля C модулем C и служит средством управления созданием программных структур, критичных по используемой ими оперативной памяти. В ОС ЕС этот вызов реализуется с помощью оператора CALL и управляющих операторов типа INSERT, OVERLAY, задаваемых дополнительно в процедуре выполнения. С помощью оверлейного вызова, система обеспечивает управление построением агрегатов с оверлейными объектами.

Вызов подзадачи A+→B

Такой вызов предназначен для организации процессов, выполняемых одновременно (параллельно) с задачей (модуль A), вызывающей подзадачу (модуль B). Организация и синхронизация задач в ОС ЕС поддерживаются не языком программирования высокого уровня, а макрокомандами Ассемблера: WAIT, POST, ATTACH. Такой тип вызова в системе АПРОП реализуется макрокомандой ATTACHP и используется для построения агрегатов с объектами, выполняемыми как подзадачи. Рассмотренный тип отношений позволяет проектировать модульные структуры с минимальными связями. При этом каждый модуль сохраняет относительную независимость, а построенная система легка для понимания, изменения и исправления.

Отношение сцепления. Этот тип отношений имеет место при связи модулей по управлению. Передача данных от одного модуля к другому производится, в основном, не через параметры, как при отношении вызова, а через элементы общего пользования. Ими могут быть элементы данных, определенные операторами COMMON; элементы общих наборов данных; общие переменные, определяемые в главном модуле для всей программы, и др. Рассматриваемый тип отношения определяет сложный интерфейс в создаваемой модульной структуре, поскольку общие данные приводят к большому количеству ссылок между модулями. В программно-создаваемой структуре пара сцепляемых модулей A, образует монолитную программную часть простой структуры с общим полем данных. Отношение сцепления задается условно двумя типами:

Сцепление A&→B - это отношение задает объединение модулей, каждый из которых обменивается данными через общую область данных и (или) общие переменные.

Отношение присоединения A/→B. Данный тип отношения позволяет объединять модули A и B в структуру, обмен данными в которых производится через общие наборы. Модульные структуры, создаваемые на основе отношения сцепления, являются сильно связанными и имеют слабую возможность к расширению и изменению. Системные трудности возрастают, если модулю, взаимодействующему с элементами общего пользования, данные передаются через входные и выходные параметры. Так, если в модульную систему объединяется m модулей и имеется n элементов общего пользования, используемых в этих m-модулях, то количество взаимосвязей будет равно n*m(m-1).

Отношение рекурсии. Данный тип отношений имеет большое теоретическое значение. Символически это отношение обозначается A↔A. Этот тип отношений не

рассматривается в системе АПРОП, поскольку реализованы структуры программ, которые представляются в виде ориентированного взвешенного графа без рекурсий.

Пример 12.

Пусть дан граф программы с главным модулем А1 в его вершине (рис. 6).

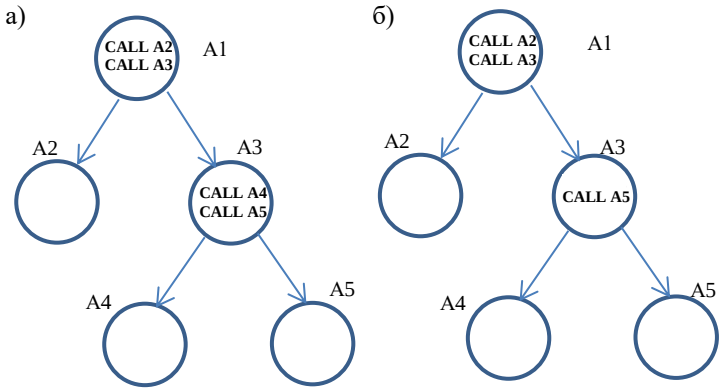


Рис. 6. Исходный граф абстрактной программы А1:

а) граф с линейным вызовом; б) граф с линейным вызовом и отношением сцепления

В графе (рис 6, а) вершины, обозначенные А1/А5, соответствует модулям, записанным на разных ЯП, и связаны линейным вызовом. Модули вызывают друг друга сверху вниз с помощью оператора CALL( ), передавая друг другу данные через параметры.

Для каждой пары разноязыковых модулей графа система генерирует (или разработчик сам выполняет это) модуль-связку (М — С), который выполняет функции, описанные в п. 1.5. В результате создается расширенный граф (рис 7).

В нем модули-связки отмечаются пунктиром и располагаются между вызывающим и вызываемым модулями. Модули-связки получают имена, модифицированные от имен вызываемых модулей. На графе рядом с этими именами в круглых скобках указываются М—С и имя конкретного вызываемого модуля, чтобы подчеркнуть, для какого модуля разработан дополнительный модуль. При этом в вызывающем модуле во всех операторах вызова CALL разно языковых модулей их имена заменяются модифицированными, что соответствует вызову модуле связок (на графе они отмечаются теми же именами со штрихом).

В модуле-связке для рассматриваемого примера указывается оператор CALL и имя вызываемого модуля. Данный оператор является макровызовом макроопределения CALL P, которое содержится в библиотеке межъязыкового интерфейса. Функции этого макроопределения описаны в п. 3.2.4 и состоят в том, чтобы обеспечить вызов и передачу параметров из модуля на одном языке программирования в модуль на другом языке.

2.4. Средства конструирования программных агрегатов

К средствам конструирования программных агрегатов в системе АПРОП относится язык конструирования, обеспечивающий непосредственное участие разработчика в процессе конструирования. Операторы языка описания построены по единому принципу, являются директивными и имеют простой синтаксис. Каждый из них начинается служебным словом, отражающим функцию, которую необходимо выполнить. Затем идет список параметров, в который, как правило, входят элементарные конструктивные объекты, подлежащие соответствующем обработке.

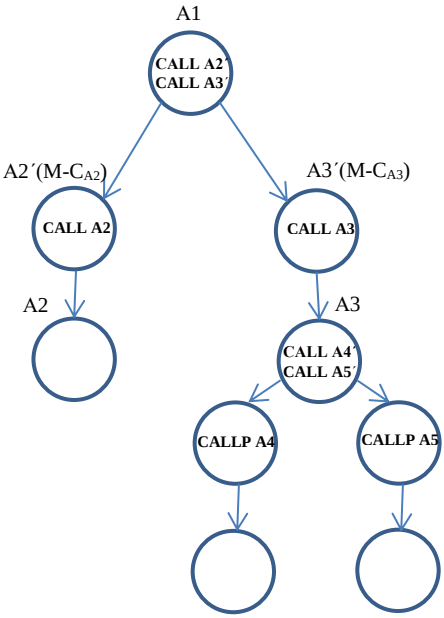


Рис. 7. Расширенный граф программы А1

В языке описания содержатся операторы, обеспечивающие описание графа для различных способов объединения, генерации описаний модулей из макромодулей, отладки и моделирования работы созданного для агрегата программного продукта. Имеются средства для ведения личных библиотек и банка модулей, а также для получения сведений о размещенных там объектах. Каждый объект может быть оттранслирован, отредактирован и выполнен. Исходные модули описываются согласно принятому в системе стандарту.

В системе обеспечивается диалоговый режим работы пользователя на всех этапах создания программных агрегатов. За пультом дисплея параллельно с выполнением трансляции, редактирования или счета отдельных модулей может проводиться исправление, ввод-вывод текстовой информации, подготовка исходных данных, ввод очередного оператора языка конструирования, обеспечивающего создание программного продукта. Результаты всех видов работ реализуются на экране дисплея.

В данном параграфе рассматриваются только языковые средства, которые необходимы при построении сложных программных агрегатов из более простых.

2.4.1. Описание графа агрегата

Процесс проектирования программных агрегатов начинается с выделения функционально независимых программных частей, установления принципов их взаимосвязи и представления их структур в виде ориентированных графов. В графе стрелки, устанавливающие связи между объектами, могут быть взвешенными и служат для отметки способа функционирования объектов. Для представления на машине граф приводится к виду оператора объединения скобочной структуры. В нем элемент, обозначающий главный модуль, является обозначением агрегата. Остальные элементы сохраняют свои отметки в графе и представляются в скобках, согласно синтаксису, дано ниже.

<оператор объединения>: = <оператор задания граф-схемы объединения> | <оператор связи>

Оператор объединения позволяет концентрировать все сведения о внешних и внутренних связях создаваемого агрегата и следить за связями, которые могут изменяться в процессе конструирования. Предлагаемые в системе средства не исключают использования средств ОС ЕС: оператора вызова CALL в языках высокого уровня, соответствующих макрокоманд операционной системы и аппарата внешних ссылок Редактора связей.

Операторы объединения описывают по исходному представлению графа структуры агрегата и правила функционирования составляющих его объектов. На основании этого задания и паспортов компонуемых элементов система формирует готовую для исполнения программу.

Конструируемый агрегат имеет статус объектов системы и может использоваться в качестве исходного объекта в другом формируемом агрегате. Если появится необходимость изменения схемы агрегата (создать оверлейную структуру или обеспечить динамический вызов других объектов, заменить одни модули другими и т.д.), то используется оператор связи.

Оператор задания граф-схемы для объединения. Синтаксическое описание операторов конструирования программных агрегатов дается в форме Бэкуса – Наура.

Ниже приводится описание оператора задания схемы, с помощью которого проверяется правильность объединения модулей.

<оператор задания граф-схемы > :: = <тип агрегата> <имя агрегата> (<список имен элементов схемы>)

<тип агрегата> :: = G/PROG/COMP/PACT

<имя агрегата> :: = <идентификатор>

<список имен элементов схемы> :: = <имя элемента схемы>|<имя элемента схемы>, <список имен элементов схемы>

<имя элемента схемы>:: =<составное имя

<составное имя> :: = <имя модуля> | <имя модуля> .. <имя библиотеки>

<имя модуля> :: = <идентификатор>

<имя библиотеки> :: = <идентификатор>

Оператор задания граф-схемы объединения является информирующим и определяет элементы, из которых конструируется данный агрегат. По этому оператору система контролирует наличие в библиотеках не только указанных элементов, но и их составляющих; о результатах контроля пользователю выдаются диагностические сообщения.

Рассматриваемый оператор является компактной формой представления агрегата. В нем может указываться в качестве элемента схемы только имя корневого модуля. Все остальные элементы на нижних уровнях иерархии графа можно не указывать. Графовая модель будет построена системой па основе паспортных данных элемента, указанного в главной вершине графа, и элементов, перечисленных в списке внешних имен паспорта.

Данный оператор позволяет задавать все элементы, входящие в граф. В зависимости от типа создаваемого агрегата участвующие при конструировании элементы могут быть разными. Так, для сегмента исходными элементами являются модули и сегменты более низкого уровня; для программ — модули, сегменты и программы; для комплексов — сегменты программ, программы и комплексы. Исходя из того, что до полного окончания конструирования полуфабрикат агрегата хранится в системе АПРОП в виде разрозненных элементов (исходных модулей и дополнительно сгенерированных системой) и некоторой системной информации, то агрегат идентифицируется в операторах объединения простым именем. Это имя фактически является именем корневого модуля, и под таким именем агрегат может участвовать в дальнейшем объединении.

Пример 13.

Пусть пользователю надо сконструировать программу А структуры (для простоты типы отношений в ней не задаются), которая приведена на рис. 8, с главным модулем А1.

Программа А1 строится из сегментов А2, А5, А8; они выполняются под управлением корневого модуля А1, и их имена должны быть описаны в паспорте этого модуля. Если элементы А2, А5, А8 находятся н задании на конструирование или в библиотеке, из которой выбирается корневой модуль, то их в схеме объединения можно не указывать, и задание схемы выглядят так: PROG A (A1).

Явное задание элементов в операторе схемы объединения увеличит время его анализа. Если составляющие элементы выбираются из других библиотек, то их надо представлять в операторе явно, т.е. в виде составных имен.

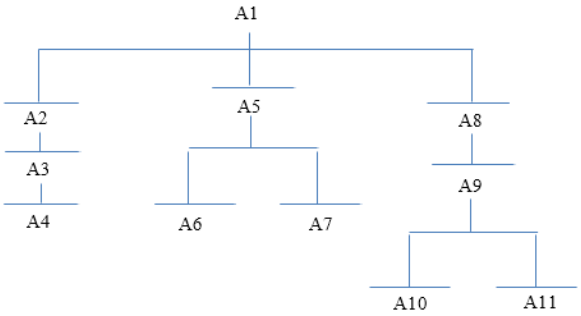


Рис. 8. Схема программы А с корневым модулем А1

Явное задание элементов в операторе схемы объединения увеличит время его анализа. Если составляющие элементы выбираются из других библиотек, то их надо представлять в операторе явно, т.е. в виде составных имен.

Все исходные элементы, из которых строится агрегат, должны быть предварительно созданы и размещены в личной библиотеке пользователя.

Оператор задания граф-схемы объединения позволяет представлять разнообразные структуры агрегатов путем незначительных перекомпоновок составляющих элементов, т.к. система сохраняет информацию, выбираемую по схеме, и сами компонуемые элементы в исходном виде.

Оператор связи. Рассматриваемый ниже оператор позволяет управлять созданием агрегатов с элементарными объектами, выполняемыми последовательно, динамически или с совмещением памяти (в графе им соответствуют разные типы отношений).

<оператор связи> :: = LINK <тип агрегата> <имя> (<список объектов>)
<список объектов> :: = <объект> | <список имен элементов схемы>, <объект>
<объект> :: = <простой объект> |<сцепленный объект> | <пусто>|<подзадача>
<простой объект> :: = <имя модуля>|<сложное имя> | <заменяемое имя>
<имя модуля> (<список ключевых параметров>) |<сложное имя> (<список ключевых параметров>) |<заменяемое имя> (<список ключевых параметров>)
<сцепленный объект> :: = <элемент сцепления>|<сцепленный объект> & <элемент сцепления> |<элемент сцепления>|<сцепленный объект>
<элемент сцепления> :: = |<простой объект> | <пусто>
<сложное имя> :: = <имя модуля>. <имя точки входа>. <имя библиотеки>
<имя модуля> :: = <идентификатор>
<имя точки входа> :: = <идентификатор> | <пусто>
<имя библиотеки> :: = <идентификатор>
<заменяемое имя> :: = <имя модуля> = <имя модуля>|<имя модуля>. <имя точки входа> = <имя модуля> . <имя точки входа>

<список ключевых параметров> :: = <ключевой параметр> | <список ключевых параметров>, <ключевой параметр>
<ключевой параметр>:: = <формальный параметр модуля> = <фактический параметр>
<динамический объект> :: = * <простой объект>
<оверлейный объект> :: = ¤ <простой объект>
<подзадача> :: = + <простой объект>

Оператор связи отображает исходный граф в линейном виде. Он предназначен для объединения объектов на уровне исходных языков программирования. Все сведения об объектах агрегата выбираются из их паспортов, присутствующих в системе в системном виде. Для управления структурой создаваемого агрегата соответствующие объекты отмечаются специальными значками в операторе объединения, имеющими тот же смысл, что при задании типов отношении в графе.

Объединяемые объекты могут передавать информацию друг другу через оператор CALL либо с помощью ключевых параметров, в операторе объединения.

Если объекты оператора LINK соответствуют линейному вызову (образуют простую структуру), то все нужные сведения для установления связей берутся из паспортов заданных исходных объектов.

Оператор сборки позволяет с помощью сложных имен задавать имена модулей, их точки входа и имена библиотек, в которых модули содержатся. Это позволяет реализовать программные агрегаты по методике, принятой в ОС ЕС, IBM, Corba.

Если объединяется несколько модулей с одинаковыми именами, находящимися в разных библиотеках, или в нескольких модулях имеются точки с одинаковыми именами, то система обеспечивает необходимую корректировку имен (см. п. 2.6). Пользователь может сам менять имена точек объектов с помощью введенного понятия — заменяемое имя. Это понятие не вызывает изменения текста модуля, а служит для корректировки имен.

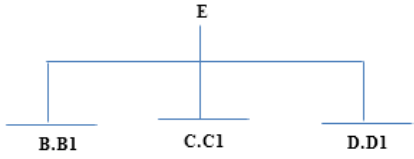


Рис. 9. Схема программы E, модули которой имеют точки входа

Пример 14.

Пусть имеется структура программы E, приведенная на рис. 9. На этом рисунке имена B1, C1, D1 — это точки входа в модули B, C, D. Пусть пользователь в процессе создания агрегата заменяет модуль C на X с тем же именем точки входа и модуль D на модуль Y с именем точки входа Y1. Тогда оператор сборки будет иметь следующий вид:

LINK PROG E (B.B1, C = X.C1, D = Y, D1 = Y1

Рассмотрев понятия заменяемого имени и точек входа, остановимся на использовании оператора связи. Для этого опишем последовательностью операторов связи структуру программы, представленную ранее на рис. 8.

Здесь имена агрегатов типа сегмент совпадают с именами корневых модулей. Объясним смысл каждого из приведенных операторов.

1) LINK SEG A2 (A2, A3,*A4)
2) LINK SEG A5 (A5, ¤ A6, ¤ A7)
3) LINK SEG A8 (A8, A9,*A10, A11)
4) LINK SEG A1 (A1,*A2)

Первый оператор описывает ветвь, начиная с модуля с именем A2. Модуль с именем A4 вызывается динамически. Модуль с именем A3 можно не указывать, если он вызывается с

помощью оператора CALL. Данному агрегату присваивается имя A2. Это дает возможность в дальнейшем использовать его в сегментах более высокого уровня.

Во втором операторе модули с именами A6 и A7 будут загружаться на один и тот же участок памяти, т.е. данная ветвь оверлейной структуры.

В третьем операторе модуль с именем A10 вызывается динамически. Остальные модули вызываются с помощью операторов CALL, их можно не указывать.

Четвертый оператор объединяет все три ветви в единый агрегат. Поскольку агрегаты с именами A5 и A8 вызываются с помощью операторов CALL, то они не указаны. Ветвь с именем A2 вызывает динамически, поэтому она указывается.

В зависимости от типа схемы PROG, SEG, COMP, то данному агрегату необходимо присвоить уникальное имя.

Если образование агрегата выполнялось без предварительного задания схемы, то все, не указанные в операторе LINK, но участвующие в конструировании модули агрегата, выбираются из библиотеки, в которой находился корневой модуль.

Сцепленный объект. Ключевые параметры, задаваемые для получения сцепленного объекта, позволяют формировать агрегаты, настроенные на фактические параметры. В паспортах спиливаемых объектов содержатся сведения о связях по данным, которые устанавливаются полностью или частично в зависимости от типов параметров, заданных в операторе объединения.

Поскольку при описании каждого модуля все переменные, массивы и т.п., которые необходимы при переходе из модуля в модуль, выносятся в список параметров либо описываются как внешние имена, либо присутствуют в COMMON-областях, то наиболее удобным случаем конструирования объектов является использование ключевых параметров. Это позволяет проще устанавливать соответствия между внешними именами и COMMON-областями, присутствующими в различных модулях, в случае, если эти модули написаны разными программистами независимо друг от друга.

COMMON-области и внешние имена удобно использовать для передачи данных между модулями, написанными на одном языке программирования. Если данные передаются между модулями, написанными на различных языках, то все переменные, массивы и т.п., участвующие в передаче данных, должны быть вынесены в список параметров.

Для переменных, массивов и т.п., вынесенных в параметры, транслятор не резервирует необходимую память. Подразумевается, что эта память выделяется в модуле, который будет вызывать данный. Таким образом, память, необходимая для передачи данных между модулями в агрегате, будет резервироваться системой автоматически. Это освобождает программиста от распределения памяти в своей программе. При этом в паспорте требуется указывать какую максимальную память необходимо выделить переменным и массивам для выполнения данного агрегата.

При использовании такого принципа построения агрегатов передача управления может осуществляться без помощи оператора CALL, т.е. подразумевается, что он является последним выполняемым оператором в модуле. Тогда в операторе объединения необходимо указать последовательность выполняемых модулей.

Пример 15.

Пусть имеются 4 модуля A1, A2, A3, A4, написанные на языке Фортран (модуль A2 находится в библиотеке):

```
                A1 (A,B,N,X)
SUBROUTINE
DIMENSION       A(N,N), B(N), X(N)
…
CALL A2 (A,N)
…
RETURN
END
```

```
SUBROUTINE      A3(D,E,F,K)
DIMENSION       E(K,K),D(K),F(K)
…
RETURN
END
SUBROUTINE      A4(Z,L)
DIMENSION       Z(L,L)
…
RETURN
END
```

После выполнения модуля A1 необходимо передать управление на модуль с именем A3 с таким соответствием параметров A=E, B=D, N=K, X=F. После выполнения модуля A3 необходимо передать управление на модуль с именем A4 со следующим соответствием параметров: E=Z, K=L. Так как A1 является корневым модулем для данного агрегата, то в его паспорте необходимо указать максимальный размер памяти (например, N=100). Тогда параметры в паспорте модуля A1 должны иметь вид:

```
  ARG    REAL      *4   A(100,100), B (100)
         INTEGER   *4   N
  RES    REAL      *4   X(100)
```

Для создания агрегата данного примера оператор сборки, объединения будет иметь следующий вид:

```
LINK SEG A1 (A1(A=E,B=D, X=F, N=K) & A3(E=Z,K=L) &A4,A2)
```

В дальнейшем данный агрегат можно использовать как программный модуль, паспорт которого будет иметь вид:

```
  ARG    A(N,N), B (N),N ;
  RES    X(N);
```

Память под параметры A, B, X резервироваться не будет.

В рассмотренном примере A1 (A=E…) &A3(EZ, K=L) &A4 является сцепляемым объектом. Он рассматривается как единое целое, т.е. использовать динамическую и оверлейную структуру можно только по отношению ко всему объекту, т.е. к A1.

2.5. Управление созданием агрегатов

Рассмотрим процесс создания программных продуктов для агрегатов, задаваемых различными графами. Основой будет исходный граф с различными типами отношений. Для каждого графа приводится функциональная схема сформированного продукта. В схемах сплошной линией отмечаются входы на начало модулей, а пунктиром – выходы из них.

В левой части схем располагаются объекты, сгенерированные системой. В модулях-связках, как и в графах, указываются основные макровызовы элементов библиотеки межъязыкового интерфейса, которые реализуют указанные типы отношений между модулями. К ним относятся CALLP, LINKP, RETURNP, ATTACHP и др., и функции которых идентичны аналогичным макрокомандам ОС ЕС и описаны в гл. 3. Остальные макровызовы, которые используются в модулях-связках, сознательно опускаются, чтобы не загромождать схему. В ряде случаев в левой части будут, кроме модулей-связок, еще и корневые модули, которые генерируются системой в случае смешанных структур программ. В правой части схемы указываются исходные объекты, которые располагаются пользователем в личной библиотеке и отмечаются в вершинах исходного графа.

Каждому исходному графу сопоставляется оператор объединения, на основе которого система обеспечивает построение агрегата простой структуры, агрегатов с динамическими объектами и с подзадачами.

Процесс проектирования и агрегата производится в несколько этапов:

1) Декомпозиция программы сверху вниз и разработка исходного графа проектируемого программного агрегата.
2) Построение отдельных элементов дерева (графа) программного агрегата и описание их функций средствами подходящих ЯП.
3) Манипулирование построенными модулями — создание библиотеки модулей и подбор готовых из байка модулей.
4) На основе графа проектируется процесс управления объединением модулей в агрегаты и управление их выполнением.
5) Моделирование работы сформированного агрегата на заданном наборе входных данных.
6) Уточнение графа и отдельных его объектов.

2.5.1. Агрегат простой структуры

Агрегат простой структуры строится по графу, содержащему линейный тип вызова объектов (см. рис. 6, а) и тип отношения сцепления (см. рис. 6, б). Это означает, что все связи между объектами устанавливаются по передаче управления и по данным оператором CALL с входными и выходными параметрами. В случае связи пары разно языковых модулей в параметры обмена выносятся данные, которые являются элементами общего пользования. Другим способом управления общей памятью является оператор сборки, в котором указывается модули с использованием оператора CALL.

На основе анализа паспортных данных модулей обеспечивается резервирование необходимой памяти в общей области.

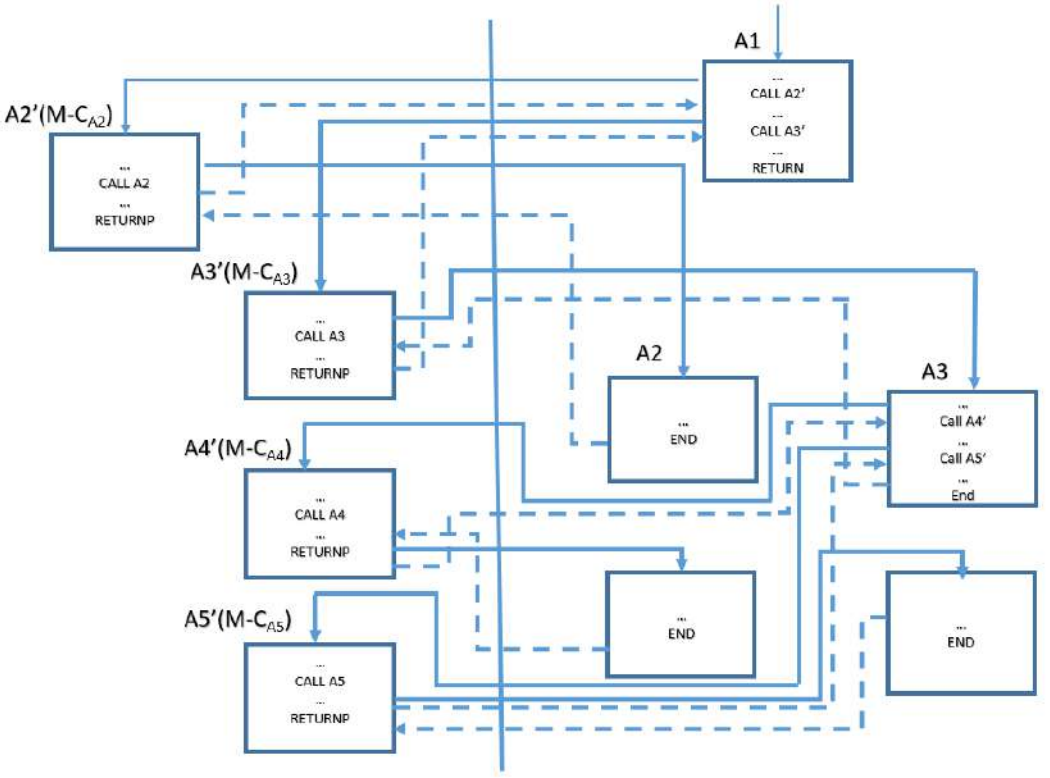


Рис. 10. Схема программы А для графа на рис. 6а.

Создание агрегата с помощью линейного вызова.

Рассмотрим два случая, когда пользователь по заданному графу (рис. 6) создает агрегат простой структуры самостоятельно, и второй случай — с помощью АПРОП.

Первый случай. Для создания агрегата по схеме рис. 10

Неавтоматизированным способом пользователь должен предварительно разместить в одном наборе данных исходные объекты агрегата правой части и дополнительно создаваемые, указанные в левой части. Для каждого из них требуется задать управляющие операторы на трансляцию и размещение полученного выходного кода другом (или том же) наборе данных. С помощью управляющих операторов Редактора связей задать сборку всех объектов в один загрузочный продукт.

Таблица 2.1

Пункт задания	Операторы, задаваемые в языке управления заданиями ОС ЕС	Пояснения
1	// PROG JOB MSGLEVEL = (2,0)	Оператор начала задания

	// EXEC ASMFC, PARM.ASM = ´LOAD´
	// ASM SYSGO DD DSN = &LOADSET,
	// DISP = (MOD, PASS),UNIT = SYSSQ,
	// SPACE = (TRK,15),DCB=BLKSIZE= 80
	// ASM. SYSIN DD *
	Перфокарты с модулем A1
		Трансляция модуля A1

2	// EXEC FORTGC
	// FORT.SYSIN DD DSN = B(A2),
	// UNIT = SYSDA, DISP = SHR,
	// VOL = SER = APROP1
	// DD DSN = B(A5),
	// UNIT = SYSDA, DISP = SHR,
	// VOL = SER = APROP1
		Трансляция модулей A2 и A5

3	// EXEC PL1LFC
	// PL1L.SYSIN DD DSN = B(A3),
	// UNIT = SYSDA, DISP = SHR,
	// VOL = SER = APROP1
		Трансляция модуля A3

4	// EXEC ASMFC, PARM.ASM = ´LOAD´
	// ASM SYSGO DD DSN = &LOADSET,
	// DISP = (MOD, PASS),UNIT = SYSSQ,
	// SPACE = (TRK,15),DCB=BLKSIZE= 80
	// ASM. SYSIN DD DSN = B(A4),
	// UNIT = SYSDA, DISP = SHR,
	// VOL = SER = APROP1
		Трансляция модуля A4

5	// EXEC ASMFC, PARM.ASM = ´LOAD´
	// ASM SYSGO DD DSN = &LOADSET,
	// DISP = (MOD, PASS),UNIT = SYSSQ,
	// SPACE = (TRK,15),DCB=BLKSIZE= 80
	// ASM. SYSIN DD *
	Перфокарты с модулем-связкой для связи A1-A2
	/*
		Трансляция модуля-связки для связи A1-A2

6	// EXEC ASMFC, PARM.ASM = ´LOAD´ // ASM SYSGO DD DSN = &LOADSET,	Трансляция модуля-связки для связи A1-

	// DISP = (MOD, PASS),UNIT = SYSSQ,
	// SPACE = (TRK,15),DCB=BLKSIZE= 80
	// ASM. SYSIN DD *
	Перфокарты с модулем-связкой для связи A1-A3
	/*
		A3

7	// EXEC ASMFC, PARM.ASM = ´LOAD´
	// ASM SYSGO DD DSN = &LOADSET,
	// DISP = (MOD, PASS),UNIT = SYSSQ,
	// SPACE = (TRK,15),DCB=BLKSIZE= 80
	// ASM. SYSIN DD *
	Перфокарты с модулем-связкой для связи A3-A4
	/*
		Трансляция модуля-связки для связи A3-A4

8	// EXEC ASMFC, PARM.ASM = ´LOAD´
	// ASM SYSGO DD DSN = &LOADSET,
	// DISP = (MOD, PASS),UNIT = SYSSQ,
	// SPACE = (TRK,15),DCB=BLKSIZE= 80
	// ASM. SYSIN DD *
	Перфокарты с модулем-связкой для связи A3-A5
	/*
		Трансляция модуля-связки для связи A3-A5

9	// EXEC LKED, PARM.LKED = ´LET´
	// LKED. SYSIN DD DSN = &LOADSET,
	// DISP = (OLD, DELETE)
	// DD *
	ENTRY A1
	NAME A
	/*
	// LKED, SYSLMOD DD DSN = LIBPOL,
	// DISP = OLD, UNIT = SYSDA,
	// VOL = SER = APROP1
	// LKED. SYSLIB DD DSN = SYS1.FORTLIB,
	// DISP = SHR
	// DD DSN = SYS1.PL1LIB,
	// DISP = SHR
		Редактирование программы A и помещение ее в библиотеку с именем LIBPOL

	//	Оператор конца задания

Таблица 2.2

Номер оператора	Оператор языка системы	Пояснения
1 2 3	SUPER READ PORT ACTIVE PORT	Начало работы Ввод порции с модулем A1 Запись модуля во временную библиотеку (ВБМ)
4	DSCR PLIB MOD B.A2 SOUR	

5	DSCR PLIB MOD B.A3 SOUR DSCR PLIB MOD B.A4 SOUR DSCR PLIB MOD B.A5 SOUR LINK PROG A(A1)	
	SEND	Конец

В таблице 2.1. приводятся все требуемые управляющие операторы заданий (причем пункты отмечаются порядковыми номерами) для получения программного агрегата. При этом считается, что пользователь знаком с языком управления заданиями [1], используемым для записи управляющих операторов. Первым оператором является оператор JOB.

Второй случай. В случае создания агрегата средствами системы АПРОП пользователь должен (табл. 2.2) выбрать исходные объекты A2/A5 из личной библиотеки, ввести модуль A1 с перфокарт и задать оператор объединения, имеющий вид, указанный в пятой строке этой таблицы, LINK PROG A(A1).

На основе этого оператора и паспортных данных исходных объектов система АПРОП обеспечивает:

- обработку объектов A/A5 и занесение во временную библиотеку монитора системы;
- создание модулей, указанных в левой части рис. 10;
- размещение созданных модулей в специальном наборе данных;
- выбор исходных модулей из личной библиотеки;
- трансляцию модулей из набора данных и из личной библиотеки;
- по окончании обработки отдельных модулей формирование управляющей информации, необходимой для создания единого программного продукта;
- формирование загрузочного продукта и размещение его во временной библиотеке монитора системы.

Выполнение готового программного продукта может производиться и в среде системы АПРОП, тогда пользователь дополнительно к ранее приведенному оператору LINK вводит оператор запуска EXEC A.

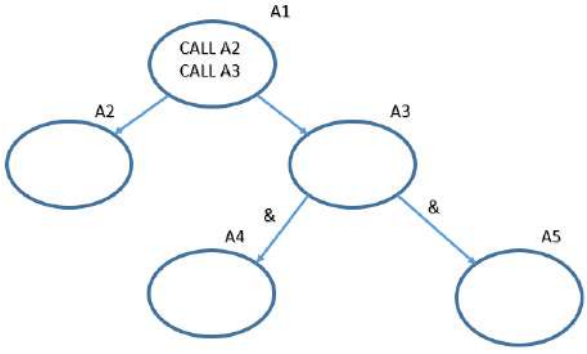


Рис. 11 Исходный граф программы A1 с отношением сцепления

Создание агрегата с помощью отношения сцепления. Изменим в исходном графе примера 9 типы связей и зададим их отношением, сцепления (рис. 11). При этом в модуле A3 исключим операторы вызова CALL, считая, что обмен данными между модулями A3, A4 и A5 производится не через параметры, а через общие области.

Для данного случая модули A4 и A5 присоединяются к A3 к его непосредственное продолжение, и на основе паспортных данных система обеспечивает доступ к данным.

Исходя из таких требований на рис. 12 приводится функциональная схема.

В ней по сравнению со схемой рис. 10 изменены не только типы связей, но и участвует сгенерированный системой корневой модуль (КМ) (на рисунке условно назван A3").

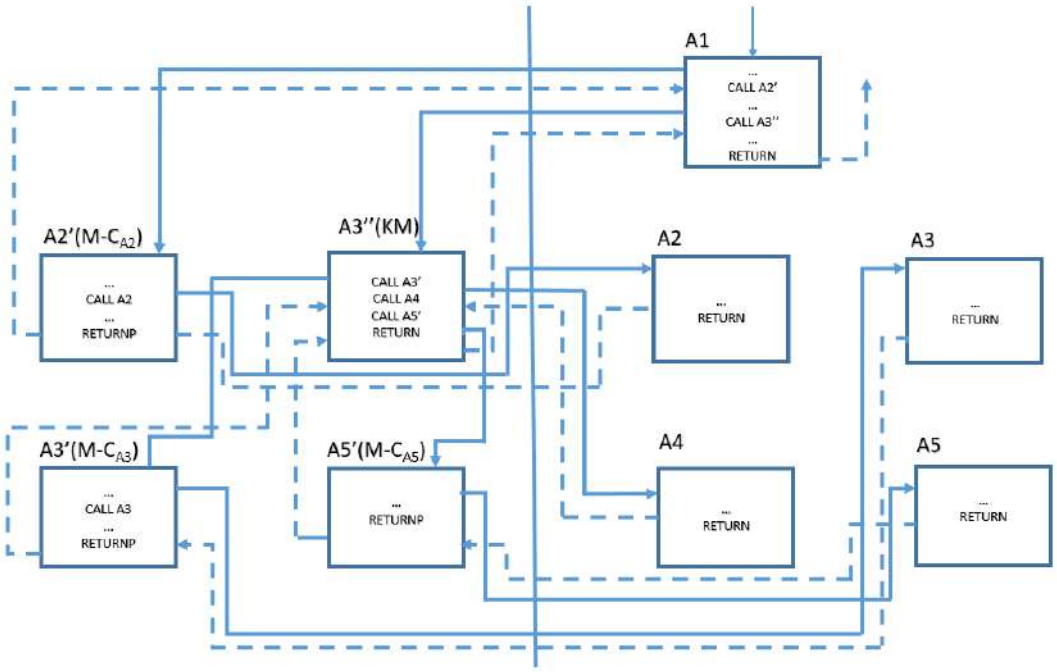


Рис. 12 Функциональная схема для графа, приведенного на рис. 11

Элементы, расположенные в левой части этой схемы, образуют среду агрегата. В нее входят три модуля-связки и один корневой модуль. Он создается всегда, когда цепочка модулей (в данном случае A3, A4, A5) сцеплена по данным. В функции корневого модуля входит:

- формирование обращений на сцепляемые объекты A3, A4, A5 (через модули-связки A3' (M —C_{A3}) и A5' (M — C_{A5}));
- параметры для общих областей которых генерируются на основе паспортных данных этих объектов. Для A4 модуль-связка не нужна, так как A1 и A4 написаны на языке Ассемблера;
- управление выполнением модулями-связки и самими модулями A4, A5.

Корневой модуль генерируется на том языке, на каком описан модуль, из которого вызывается первый модуль, входящий в сцепленный объект A1, т. е. в данном случае — на языке Ассемблера.

Поскольку по функциональной схеме трудно проследить за данными, расположенными в общей области, на рис. 13 рассматривается более подробно фрагмент структуры агрегата для двух связываемых модулей A и B, обмен данными в которых происходи через общую область.

Заштрихованные нижние части модулей A и B на рис.13 содержат адресные константы и различные переменные. При этом корневой модуль выполняет функции управления первым исходным модулем A, а также модулей обслуживания общих областей. На основе таблицы внешних символов (ТВС), создаваемой Редактором связей, и специальных макровызовов макроопределений БМИ, размещаемых в модуле-связке, система обеспечивает механизм пересылки данных через общую область одного модуля в другой.

В случае обмена данными между модулями через общие наборы в паспорте в разделе идентификации операторов ввода-вывода специфицируются используемые наборы данных. С их помощью аналогично общим областям система генерирует корневой модуль, который управляет общими наборами.

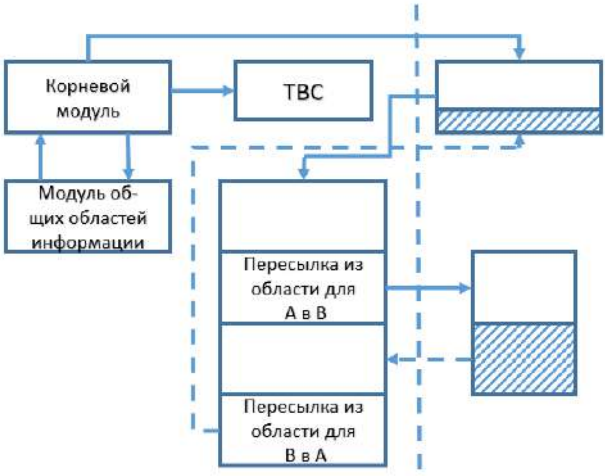


Рис. 13. Фрагмент связи модулей А и В с данными, передаваемыми через общую область

В рамках системы АПРОП указанием на изготовление программного продукта для агрегата по рассмотренной схеме (рис. 12) является:

LINK SEG A1 (A1, A2, A3 & A4 &A5)

При формировании корневого модуля для А3 система АПРОП на основе паспортных данных исходных модулей А1, А3/А5 устанавливает соответствие по используемым наборам или общим областям, а затем генерирует корневой модуль, в котором содержатся операторы CALL для связи КМ с другими модулями по данным и управлению (на рис. 12 эти модули обозначены соответственно А3´, А4´, А5´, а сгенерированный КМ получает имя А3´´.

Таким образом, в рамках системы АПРОП пользователь задает оператор LINK и набор входящих в него модулей. Система сама строит функциональную схему для каждого агрегата и запоминает в виде специальной таблицы отношений объектов графа. Затем на основе граф-схемы объединения обеспечиваются действия, аналогичные тем, которые написаны в табл. 2.1 и 2.2 для создания программного агрегата.

2.5.2. Агрегат с оверлейными объектами

Создание агрегата с оверлейными объектами также первоначально сводится к разработке исходного графа (рис. 14) и функциональных схем, аналогичных тем, которые приведены ранее на рис. 10 и 12.

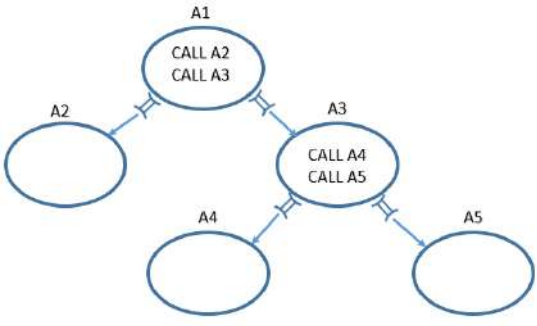


Рис. 14. Исходный граф программы А1 с оверлейным вызовом

Затем пользователь должен обеспечить трансляцию исходных модулей (и модулей-связок), после чего задать для них управляющие операторы Редактора связей с заданным точками загрузки.

При применении систем АПРОП от пользователя требуется задать оператора связи: LINK SEG A1 (A1, ¤ A2, ¤ A3, ¤ A4, ¤ A5).

2.5.3. Агрегат с динамическими объектами

Если предположить, что в примере 9 исходный объект A3 вызывается динамически, то для него создается отдельный загрузочный продукт, который должен выполняться самостоятельно. При этом создается среда, которая включает требуемые модули-связки и корневой модуль.

Программный продукт для A1 должен состоять из основной управляющей части – корневого модуля, загрузочного модуля, включающего A3 и вызываемого для работы в основную память, а также описываемых ниже.

Агрегат из одно языковых объектов. Для графа на рис. 15 приведена функциональная схема на рис. 16. Она построена первоначально для случая, когда в вершинах исходного графа расположены исходные объекты A1 — A5, написанные на одном языке программирования, объект A3 связан с A1 динамическим типом отношения.

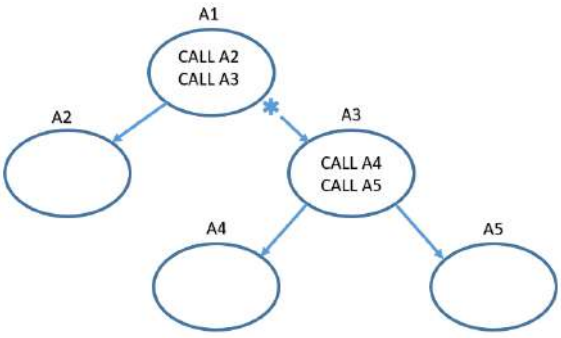


Рис. 15. Исходный граф программы A1 с динамическим вызовом

Из схемы следует, что для обеспечения динамического вызова объекта A3 требуется разработать модуль-связку A3´ (M - C_{A3}), в которой используется макровызов LINKP (см. п. 3.2.5) для обеспечения динамической загрузи объекта A3.

Шаги, приводящие к созданию агрегата рядовым пользователем и системой АПРОП, во многом совпадают с ранее описанными, поэтому остановимся на особенностях выполнения созданного агрегата.

Готовый продукт для агрегата должна состоять из двух загрузочных модулей. Первому их них на рис. 16 соответствует совокупность объектов, расположенных выше жирной горизонтальной черты, т.е. те, помечены динамическим типом вызова. Во время выполнения первого загрузочного модуля, содержащийся в нем модуль-связка A3´(M-C_{A3}) обеспечит динамический вызов второго агрегата (LINKP A3).

Данный агрегат будет проигрывать в памяти и во времени выполнения, поскольку оба загрузочных модуля целиком будут находиться в оперативной памяти. Однако в том случае, когда в агрегате будет несколько динамически вызываемых загрузочных модулей, то тогда будет общая экономия, т. е. именно в этом случае целесообразно применять динамический тип отношений между модулями.

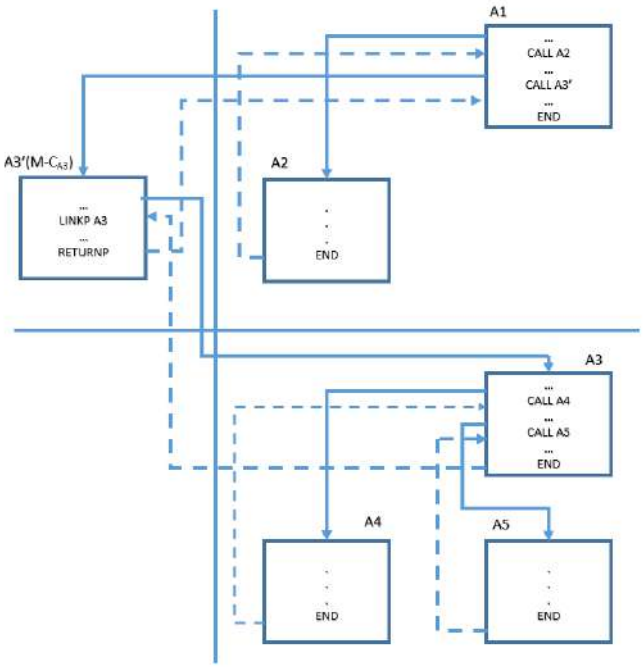


Рис. 16 Функциональная схема связи одноязыковых модулей для графа, приведенного на рис. 15

Связь разно языковых объектов. Исходя из предположения, что объекты A1/A5 (рис. 15) записаны на разных ЯП, рассмотрена функциональная схема (рис. 17), кото является более общим случаем схемы по сравнению с ранее приведенной на рис. 16. Поскольку исходные объекты A2, A3, A4, A5 вступают друг с другом в разные типы отношений и записаны на разных ЯП, то в левой части данного рисунка содержатся четыре модуля-связки.

При этом, поскольку модуль A3 объявлен как динамический объект, то в соответствующем для него модуле-связке, кроме обычных функций, выполняемых для обеспечения сохранения, восстановления регистров и областей, а также преобразования данных, выполняет функция динамического вызова модуля A3 (LINKP A3).

я данной схемы система создает также два загрузочных модуля. В первый из них попадают объектные модули A1, A2, A2´ (M – C_{A2}) и
A3´ (M – C_{A3}), во второй – остальное. Динамически вызываемый модуль располагается во внешней памяти и загружается в оперативную память в момент выполнения макрокоманды LINKP A3. При этом модуль-связка с этой макрокомандой располагается в первом загрузочном модуле. После первоначального запуска модуля A1 и до конца работы первый загрузочный модуль будет находиться в оперативной памяти.

Таким образом, для создаваемого агрегата формируется столько загрузочных модулей, сколько исходных модулей объявлено динамически выполняемыми плюс первый загрузочный корневой модуль.

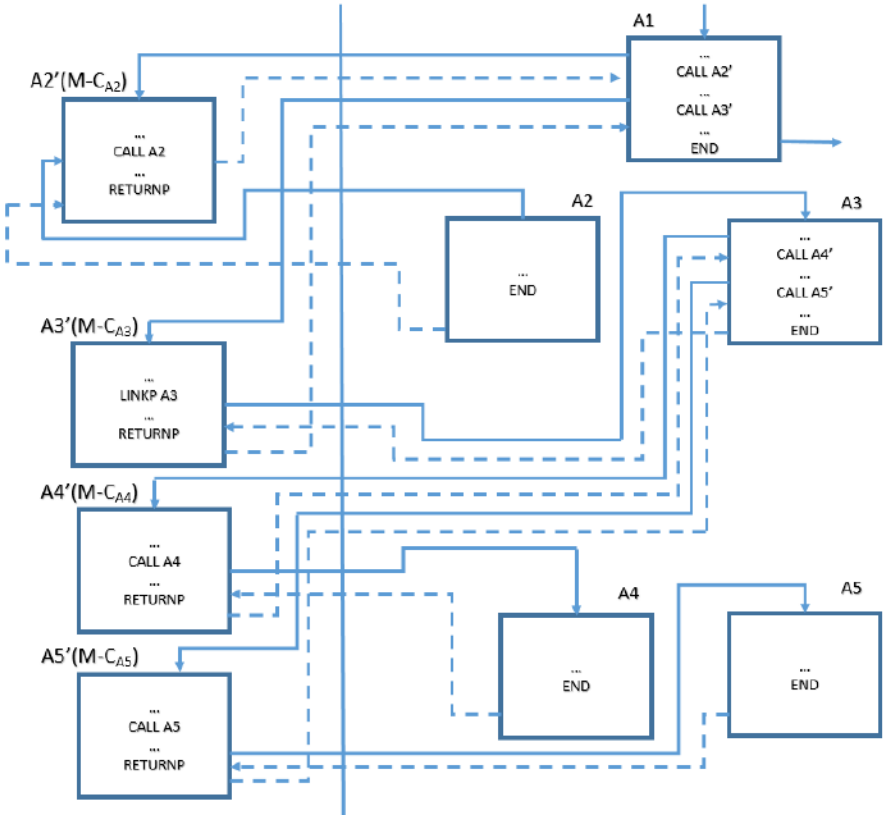


Рис. 17. Функциональная схема связи разноязыковых модулей для графа, приведенного на рис. 15

2.5.4. Агрегат с подзадачами

Пусть в рассматриваемом примере 9 объекты A2 и A3 должны быть оформлены как подзадачи, т. е. имеет место граф (рис. 18).

Для этого случая оператор объединения имеет следующий вид:

LINK PROG A1 (A1, +A2, +A3, A4, A5).

Система по этому оператору и паспортным данным исходных объектов, участвующих в нем, создаст три отдельные программные частя — A, A2, A3. При этом создаваемый системой корневой модуль обеспечивает режим подзадач с помощью специально создаваемой среды включающей модули-связки.

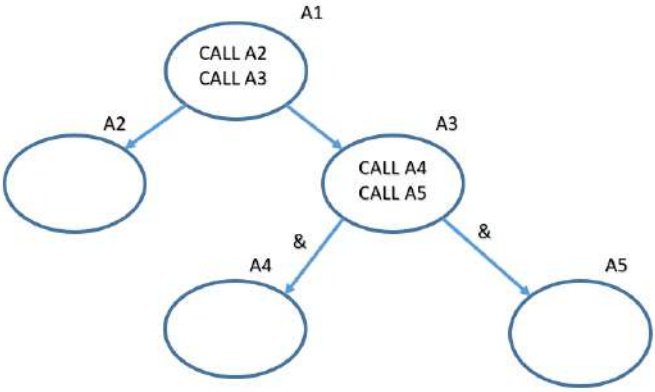


Рис. 18. Исходный граф программы A1 с вызовом подзадач

Модуль IGWA — основной модуль управления подзадачами. Он содержит всю информацию, необходимую для создания и уничтожения подзадач. Эта информация находится в таблице подзадач (ТПЗ). Всякий раз, когда образуется новая подзадача, имя ее и блоки управления заносятся в ТПЗ. Это осуществляется специальной макрокомандой ATTACHP. Соответствующее ей макроопределение является модифицированным макроопределением ATTACH ОС ЕС, оно дополнено вызовом модуля IGWAF, который обращается к модулю IGWA для занесения в ТПЗ информации об образовавшейся подзадаче.

Если подзадачи, образовываемые в одном агрегате, связаны по информации (например, через внешние наборы данных), то возникает необходимость синхронизации выполняемых подзадач. Для синхронизации пользователь в текст своих модулей включает операторы CALL IGWAF, если модуль, выдавший этот оператор, написан на языке Фортран или на Ассемблере и CALL IGWAP, если модуль, выдавший этот оператор, написан на языке ПЛ/1. Эти операторы включаются в тех местах текста, где необходимо использовать информацию, вырабатываемую другой подзадачей. Параметром для модулей IGWAF и IGWAP служит имя подзадачи, завершение которой нужно ожидать. Модули IGWAF или IGWAP обращаются к модулю IGWA, который по таблице ТВС и ТПЗ проверяет завершение нужной подзадачи. Если данная подзадача не окончена, то модуль, то запрос на синхронизацию, переводится в состояние ожидания.

После завершения нужной подзадачи, управление возвращается модулю, выдавшему оператор CALL IGWAF или CALL IGWAP.

В задачу корневого модуля, генерируемого системой операции:

- занесение в основную память модуля IGWA и таблицу ТВС;
- вызов корневого модуля пользователя;
- проверка завершения подзадач и уничтожение информации, носящейся к ним;
- уничтожение модуля IGWA из основной памяти.

Перед завершением работы корневой модуль обращается к модулю IGWA для проверки окончания всех подзадач и удаления информации, относящейся к ним. Если некоторые подзадачи не окончены, то происходит ожидание их завершения. После этого модуль IGWA удаляются из основной памяти.

Для примера 9 и рис. 18 функциональная схема изображена на рис. 19.

Программный продукт состоит из следующих частей (на рисунке они выделены пунктирными линиями):

- Корневой модуль КМ, модуль A1, модули-связки A2' и A3', модуль IGWAF;
- модуль IGWA;
- модуль, обрабатывающий ТВС;

- модуль A2 и IGWAF;
- модули A3, A4, A5.

Каждая часть представляет собой отредактированный загрузочный модуль. Модули IGWA, IGWAF и IGWAP находятся в библиотеке системы АПРОП. На рисунке модуль A2 ожидает завершения подзадачи A3. Макроопределение ATTACHP состоит из макроопределений ATTACH и CALL (на рис. 19 вхождение обозначается «+»).

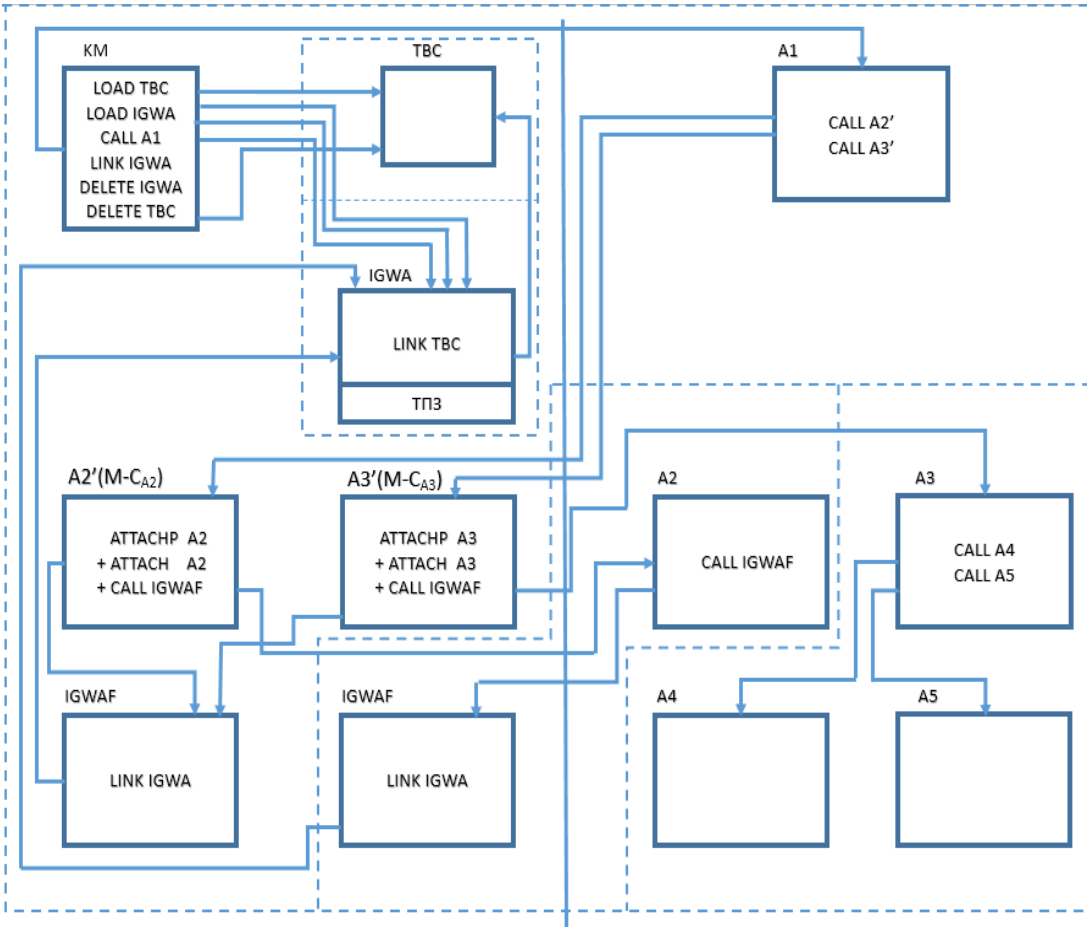


Рис. 19. Схема связей программного агрегата с подзадачами

Грабовая структура некоторой системы (рис. 20), по которой строятся интерфейсные объекты, которые проводят преобразование передаваемых данных между объектами в ЯП для некоторой предметной области (рис. 20) [43, 44].

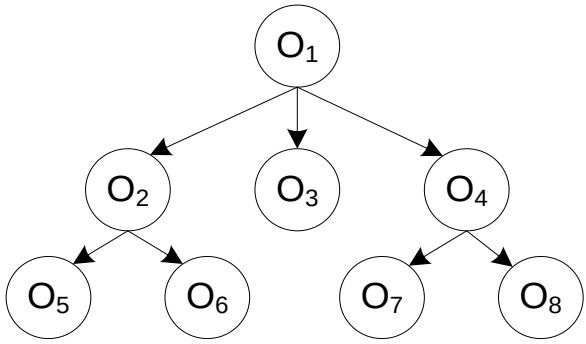


Рис. 20 Структура графа G ={O}

На основе данного графа *G* строятся программы:

1) $P_1 = O_2 \cup O_5$
2) $P_2 = O_2 \cup O_6$,:
3) P_3 :
4) $P_4 = O_4 \cup O_7$,
5) $P_5 = O_4 \cup O_8$, $P_0 = (P_1 \cup P_2 \cup P_3 \cup P_4 \cup P_5)$.

Так как объекты записаны в разных ЯП, то между ними должен быть созданы интерфейсные посредники, обеспечивающие преобразование обмениваемых данных, они задаются в виде, например, *In* $O'_5 (O_2 O_5)$:

В результате граф *G* пополняется интерфейсными объектами посредниками (рис. 21) O_5, O_6, O_7, O_7.

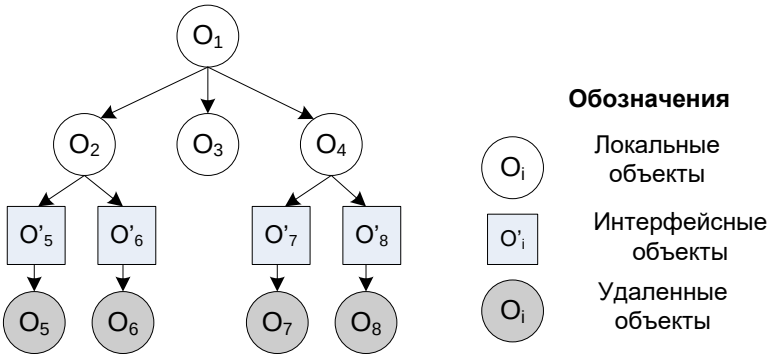


Рис.21. Граф G (O. I, R) с интерфейсными объектами.

Вершины данного графа *G* задают 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 репозитарии, а дуги соответствуют отношениям между всеми видами объектов.

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

Множество функциональных объектов графа *G* соответствует методу реализации этих объектов ПрО. При конкретизации объекты графа *G* устанавливают связь с другими объектами через интерфейсные параметры *in*, *out*, *input* множества интерфейсов I.

Интерфейсным объектам графа соответствуют описание данных, методы их передачи

по запросам к удаленным процедурам или методам и возможным операциям преобразования этих данных к соответствующим форматам среды выполнения.

Для графа G на рис.1 можно построить программы P_1 – P_5 с использованием операторов объединения (сборки) *link*:

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

Система АПРОП берет на себя всю работу, связанную с формированием функциональной схемы (по оператору связи LINK и на основе паспортных данных объединяемых модулей), а также созданием по ней программного продукта. В этом случае, объем работ, выполняемых системой, можно оценить не менее чем на 50%, что является серьезным аргументом в пользу применения средств автоматизации системы АПРОП.

2.6. Реализация межъязыкового интерфейса

Функцию обеспечения связи разно языковых модулей в системе АПРОП выполняет программная компонента «межъязыковый интерфейс», которая состоит из компоновщика агрегатов, библиотеки интерфейса и программы обработки паспортных данных исходных объектов.

Компоновщик осуществляет объединение одноязыковых и разноязыковых модулей по формальному заданию оператора связи агрегата (оператор LINK) и с помощью паспортов модулей, входящих в граф-схему агрегата.

Исходные модули предварительно отрабатываются до состояния готовности и накапливаются в личной библиотеке или перед началом конструирования вводятся с перфокарт и хранятся во временной библиотеке монитора (ВБМ) на время конструирования.

При вводе исходных объектов в эти хранилища производится их обработка, которая состоит в следующем:

- проверяется правильность составления паспортов с выдачей соответствующих диагностических сообщения;
- формируется системная таблица исходных объектов;
- преобразуются паспортные данные к внутрисистемному представлению, и формируется библиотека паспортов;
- тексты объектов заносятся в указанные хранилища.

Таким образом, работа программного блока межъязыкового интерфейса основывается на следующих системных функциях:

- Манипулирование исходными объектами с целью доведения их до готового вида и занесения в системные хранилища;
- проверка правильности паспортных данных и установление соответствия данных между каждой парой связываемых модулей;
- формирование среды агрегата и управляющих предложений для получения программного продукта.

Описываемая компонента при своей работе основывается системных таблицах, которые ей передаются при запуске на выполнение.

2.6.1. Общая схема интерфейса

В данной работе рассмотрены отличия средств в современных широко используемых языках программирования и способы обеспечения межъязыкового интерфейса для языков программирования ОС ЕС. Однако затрагиваемые вопросы актуальны при построении межъязыкового интерфейса в любой операционной системе. Общность проблем возникает из одинаковых целей при создании программ из модулей и из использования одних и тех же языков программирования.

В связи с этим рассмотрим наиболее общую функциональную схему интерфейса, отражающую указанные вопросы. В общности решаемых вопросов схему условно будем называть между модульным интерфейсом, в которой рассмотренный межъязыковый интерфейс является составной его частью. Общая схема межмодульного интерфейса представлена на рис. 22(а), а на рис. 22(б) — межязыкового интерфейса.

Межмодульный интерфейс.

Основные функции межмодульного интерфейса (рис.20):

- обработка паспортных данных исходных модулей;
- средства композиции программных агрегатов;
- межъязыковый интерфейс и средства отладки и моделирования программных агрегатов.

Для формализации представления программных объектов вводятся паспортные данные, содержащие основные характеристики модулей. Для них вводятся два вида представления: исходный и системный. Исходный вид представления паспортных данных определяется языком описания исходного модуля и приведен в п.2.2. Системный вид — это представление паспортных данных, заданных в исходном виде, в форме таблиц и списков с целью уменьшения времени работы межмодульного интерфейса.

Приведение паспортных данных к системному виду определяется классом языков, для которых интерфейс реализуется. Эта обработка большей частью одинакова для любых языков, поскольку паспортные данные имеют стандартный вид.

Они задаются общими для любого языка программирования характеристиками: аргументами (ARG), результатами (RES), спецификации величин (DES), промежуточные величины (MDL), вызываемые модули (MOD), вызываемые подпрограммы функции (FUN), общие области (COM), список внешних имен (EXT), список точек входа в модуль (ENT), список операторов ввода-вывода и их характеристик (IOL).

Среди этих характеристик, задаваемых идентично для всех языков программирования, характеристика «спецификация величин» определяется способом их описания, принятым в конкретном языке программирования (подробнее см. п. 2.6.1.)

К средствам композиции программных агрегатов относятся способы построения программных агрегатов и способы создания различных типов программных агрегатов. Эти средства необходимы при композиции программных агрегатов из одинаковых и разноязыковых исходных модулей.

Как было сказано ранее, структура программ, создаваемая в рамках этой книги ОС ЕС, является простой, оверлейной или динамической.

Как было сказано ранее (см. п. 1.3), структура программ, создаваемая в рамках ОС ЕС, может быть: простой, оверлейной, динамической или структурой с подзадачами. В случае другой операционной системы может появиться новый тип структуры программы, который, не нарушая общности, дополняется в схему.

Способы создания различных типов агрегатов (сегментов, программ, комплексов и пакетов) определяются назначением агрегата и дальнейшим его использованием (см. п. 2.5).

Средства отладки и моделирования агрегатов (см. рис. 20) необходимы для проверки для проверки работоспособности создаваемого программного агрегата. Эти на средства

включают пакетную и диалоговую отладку, которые во многом сходны; исключением являются формы выдачи результата. В пакетном режиме результаты выводятся на печать.

В случае другой операционной системы может появиться новый тип структуры программы, который, не нарушая общности, дополняется в схему. Основные функции межмодульного интерфейса (рис.20):

- обработка паспортных данных исходных модулей;
- средства композиции программных агрегатов;

большей частью одинакова для любых языков, поскольку паспортные данные имеют стандартный вид.

Они задаются общими для любого языка программирования характеристиками: аргументами (ARG), результатами (RES), спецификации величин (DES), промежуточные величины (MDL), вызываемые модули (MOD), вызываемые подпрограммы функции (FUN), общие области (COM), список внешних имен (EXT), список точек входа в модуль (ENT), список операторов ввода-вывода и их характеристик (IOL).

Среди этих характеристик, задаваемых идентично для всех языков программирования, характеристика «спецификация величин» определяется способом их описания, принятым в конкретном языке программирования (подробнее см. п. 2.6.1.)

К средствам композиции программных агрегатов относятся способы построения программных агрегатов и способы создания различных типов программных агрегатов. Эти средства необходимы при композиции программных агрегатов из одинаковых и разноязыковых исходных модулей.

В диалоговом режиме результаты выводятся на экран дисплея. К средствам отладки и моделирования агрегатов относятся:

- выдача графа создаваемого агрегата;
- трассировка межмодульных переходов в агрегате.

К средствам отладки и моделирования агрегата относится выдача графа, трассировка межмодульных переходов при выполнении агрегата и сбор данных о выполнении.

Под выдачей графа на печать или на экран дисплея понимается, с одной стороны, формирование исходного графа по заданному оператору объединения, представление его кадрами экрана и последующая их выдача. С другой стороны, прослеживание в динамике выполнения агрегата за вершинами графа, формирование динамических ветвей (цепей) и их выдача.

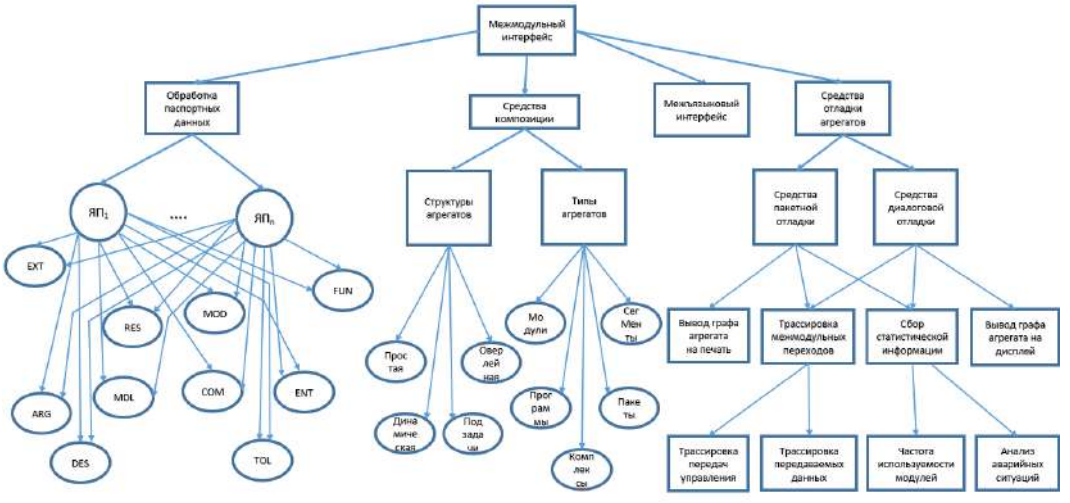


Рис. 22(а). Общая схема межмодульного интерфейса

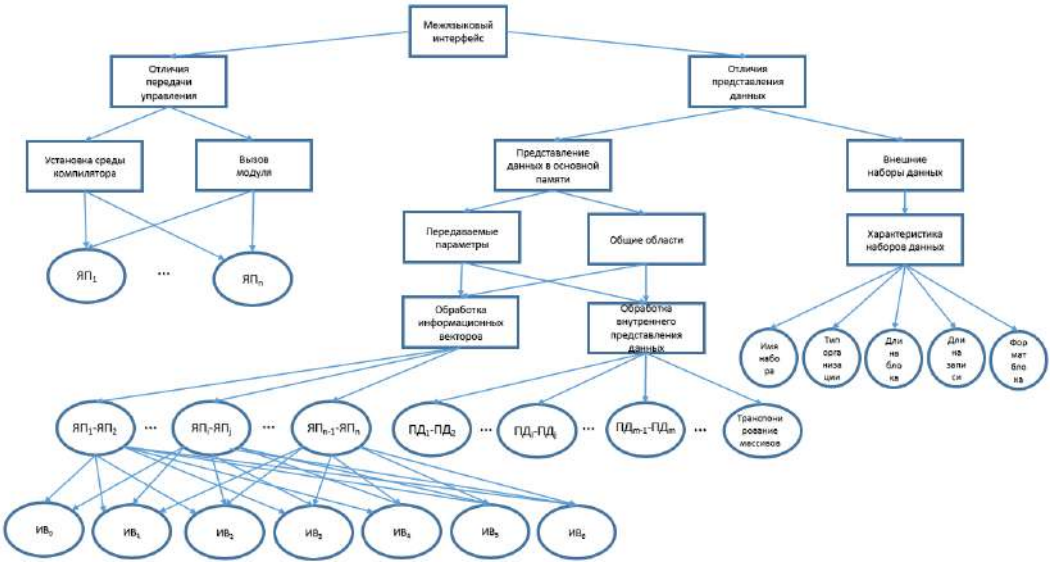


Рис. 22 (б). Общая схема межъязыкового интерфейса

Сборка межмодульных переходов состоит в проверке правильности передаваемых данных и в формировании динамических цепочек выполняемых модулей, связанных передачами управления. Проверка правильности предаваемых данных осуществляется путем сравнения передаваемой информации с шаблоном, задаваемым в паспорте модуля. Результат сравнения во время выполнения агрегата выдается на экран дисплея или на печать, что служит способом контроля правильности передаваемой информации для пользователя. Формирование динамических цепочек выполняемых модулей производится по каждой передаче управления к вызываемому модулю.

Сбор статистической информации состоит в определении частоты используемости модулей и в анализе аварийных ситуаций. Определение частоты используемых модулей позволяет наиболее рационально разместить модули в библиотеках с целью минимизации времени обращения к ним. Анализ аварийных ситуаций позволяет выявить наиболее типичные ошибки, возникающие при выполнении агрегата.

Отличия передачи управления определяются способами установки среды компилятора для вызываемого модуля и вызова модуля (способ обращения). Установка среды и вызова модуля в общем случае могут быть различными для каждого из языков программирования из класса языков $ЯП_1$, $ЯП_n$.

Передаваемые от модуля к модулю данные делятся на данные, передаваемые через основную память и данные, передаваемые с помощью внешних наборов данных. В первом случае данные могут передаваться через основную память либо через параметры модулей, либо через общие области. Для передаваемых параметров и для общих областей устранение отличий происходит в два этапа.

Сбор статистической информации состоит в определении частоты используемости модулей и в анализе аварийных ситуаций. Определение частоты используемых модулей позволяет наиболее рационально разместить модули в библиотеках с целью минимизации времени обращения к ним. Анализ аварийных ситуаций позволяет выявить наиболее типичные ошибки, возникающие при выполнении агрегата.

На первом этапе обрабатываются информационные векторы (ИВ). В общем случае для каждой пары языков $ЯП_1$ и $ЯП_j$ ($i \neq j$) обработка может происходить по-разному.

На схеме это отражение в связях пар языков $ЯП_1$-$ЯП_2$, $ЯП_i$- ЯПj…, $ЯП_{n-1}$-$ЯП_n$.

Межъязыковый интерфейс (см. рис. 22). Эта схема является обобщением рассмотренных ранее отличий в модулях, написанных на разных языках программирования. Все отличия делятся на 2 группы. Первая — содержит отличия, возникающие при передаче управления, а вторая — при передаче данных.

При построении ИВ, можно разделить на следующие этапы группы. К первому относятся простые переменные ($ИВ_0$), арифметические массивы ($ИВ_1$), строки ($ИВ_2$), массивы строк постоянной длины ($ИВ_3$), массивы строк переменной длины ($ИВ_4$), структуры ($ИВ_5$), массивы структур ($ИВ_6$).

На втором этапе осуществляется обработка внутреннего представления данных, которая заключается в переводе величин из одной формы представления в другую.

На схеме изображено в виде связей $ПД_1$-$ПД_2$, $ПД_I$— $ПД_j$, $ПД_{m-1}$ –$ПД_m$, $i \neq j$), если для величины существует другая форма представления; в транспортировании массивов для модулей, написанных на языке Фортран.

Среди отличий, возникающих при передаче данных через общие наборы, основными являются отличия, связанные с характеристиками наборов данных: имя набора, длина и формат блока набора, тип организации, длина записи. Все отличия и способы их устранения описаны в гл. 1.

Таким образом, реализация межмодульного интерфейса для некоторой операционной системы состоит в следующем:

1) Определить используемые языки программирования и возможности создаваемого интерфейса.
2) Выделить из приведенной схемы подсхему, соответствующую выбранным языкам программирования.
3) Определить характеристики, для которых не существует отличий в языках программирования, и убрать в подсхемах соответствующие стрелки.
4) Для каждой из оставшихся стрелок написать соответствующее программное обеспечение.

2.6.2. Компоновщик агрегатов

Программная компонента. Компоновщик реализует в рамках ОС ЕС задачу межъязыкового интерфейса согласно схеме, приведенной на рис. 21.

Исходными данными для Компоновщика являются:

- оператор объединения, задаваемый по исходному графу формируемого агрегата;
- набор модулей, на которых определен граф агрегата;
- паспортные данные модулей.

На основе этих исходных данных Компоновщик устанавливает соответствие между вершинами и дугами графа, осуществляет формальную проверку правильности сопряжения модулей на уровне их исходного представления до компиляции в отличие от средств ОС ЕС, где связь производится во время компиляции или после нее. Поэтому в ОС ЕС ошибки, обнаруженные в связях, вызывают повторение всех видов работ по объединению с самого начала. Компоновщик выявляет ошибки в связях до компиляции. Выявленные им ошибки влекут за собой внесение изменений либо в исходный граф, либо в паспорта, либо в тексты модулей, либо в оператор связи. При каждом внесении изменений в оператор LINK с целью удаления или добавления имей объектов производится корректировка исходного графа и матрицы отношений, являющейся внутренним представлением исходного графа.

В случае, когда изменения касаются отдельных модулей (их текстов или паспортов), производится повторный анализ оператора объединения и связанных с ним систем таблиц для корректировки соответствующего загрузочного модуля в созданном агрегате.

Таким образом, реализованный в системе АПРОП подход к объединению объектов в различные программные конфигурации агрегата способствует созданию качественного программного продукта.

Для обеспечения надежности работы агрегата имеются следующие возможности. В операторе сборки указывается режим моделирования, который рассматривается Компоновщиком как как задание на формирование в среде агрегата средств управления работой элементов агрегата в динамике их выполнения.

Компоновщик системы выполняет создание агрегата в два этапа.

На первом этапе Компоновщик выполняет:

- проверку правильности сопряжения модулей;
- генерацию требуемых модулей-связок для каждой пары разнородных объектов;
- генерацию корневого модуля для сцепленных объектов;
- генерацию корневого модуля для обеспечения динамики и режима подзадач;
- формирование управляющих предложений для проведения трансляции.
- генерацию требуемых модулей-связок для каждой пары разно языковых объектов;
- генерацию корневого модуля для случая сцепления объект генерацию корневого модуля для обеспечения динамики и режима подзадач;
- формирование управляющих предложений для проведения трансляции исходных и сгенерированных элементов для агрегата.

Результатом работы Компоновщика на этом этапе являет полуфабрикат программного продукта, состоящий из следуют элементов: исходных текстов модулей-связок и корневых модулей (среда агрегата); управляющих предложений для компилятора; расширенной градовой модели (матрицы отношений) и разного рода таблиц.

При создании модулей-связок устанавливается соответствие между именами модулей исходя из следующих системных соглашений. Имя вызываемого модуля остается без изменений, а моду связке присваивается модифицированное имя, полученное из имев вызываемого модуля. При этом рассматриваются два случая.

1. Если длина имени не более пяти символов, то из него образуется имя модуля-связки путем добавления к нему трех символов (#, ЯП|, ЯП)), где два последних символа являются сокращением для ЯП вызываемого и вызывающего модулей, ими могут быть ПЛ/1, F (Фортран), а (Ассемблер).
2. Если имя вызываемого модуля состоит из более шести символов, то символы после пятого отбрасываются, шестой символ этого имени заменяется на знак #, седьмой и восьмой символы заменяются указанными сокращениями для ЯП.

После того как произведена обработка имен в вызывающем модуле, делается замена на обращение к модулю-связки, для которого генерируется обращение к вызываемому модуле

Замена имен осуществляется с помощью управляющих предложений Редактора связей INCLUDE и CHANGE. Первое из них идентифицирует дополнительный набор данных со сформированными элементами, второе используется для замены имен внешних величин и имеет вид: CHANGE <мня заменяемого элемента> (<новое мня>).

Подвергаются изменениям также имена точек входа в модулях. Они меняются по такому же принципу, а в управляющих предложениях CHANGE указываются их имена.

В связи с заменой имен, производимой Компоновщиком, на ИМ1 модулей и точек входа накладываются следующие ограничения:

Имена не должны содержать символа, а в личной библиотеке пользователя не должны присутствовать одинаково обозначенные объекты.

Таким образом, на первой фазе работы Компоновщик обеспечивает не только ранее указанные действия, но и необходимую корректировку и замену используемых в агрегате одинаковых имен.

На втором этапе компоновщик через созданные им управляющие предложения обеспечивает сначала обращение к компилятору для получения объектного кода для каждого исходного объекта.

Затем Компоновщик через созданные им управляющие предложения формирует операторы для Редактора связей для редактирования агрегата и помещения его в личную библиотеку пользователя. Вместе с программным продуктом агрегата в библиотеку помещаются расширенная графовая модель.

Состав Компоновщика. Для реализации описанных выше – функций разработаны и включены в состав Компоновщика следующие программные элементы:

- анализатор графовой модели агрегата;
- генератор модулей-связок:
- генератор управляющих операторов:
- корректировщик (ТВС, системных таблиц и матриц отношений;
- модуль выдачи диагностических сообщении;
- модуль запуска системных обрабатывающих программ ОС ЕС (трансляторов. Редактора связей, утилит);
- библиотека макроопределений и загрузочных модулей межъязыкового интерфейса (см. гл. 3).

Эти программные элементы являются основными. Последние два элемента определяются набором языков программирования, включенных в межъязыковый интерфейс. Всякое добавление нового языка не вызывает принципиальных изменений в составе программных элементов Компоновщика, а выражается в добавлении строк в системные таблицы и в БМИ. Принцип создания агрегата тот же.

2.6.3. Обработка паспортов модулей

В процессе обработки оператора объединения используются паспорта входящих в создаваемый агрегат объектов в системном представлении.

Среди паспортных данных используются следующие:

ARG — аргументы, RES — результаты, MDL — промежуточные величины, DES — спецификации величин, MOD — вызываемые Модули, FUN—список используемых подпрограмм-функций, ENT — список точек входа в данный модуль, EXT— список внешних имен, COM — список общих областей, IOL — список операторов ввода-вывода.

Аргументы и результаты этого списка являются передаваемыми параметрами. Переменные, задающие результаты, строго следуют за параметрами (аргументами).

В список промежуточных величин входят те переменные, массивы и т. д., которые являются передаваемыми параметрами при обращении по оператору CALL от данного модуля к другим, а также те, которые присутствуют в общих областях. Если промежуточная величина является аргументом или результатом и передается через параметр в вызываемый модуль, то ее также необходимо заносить в список промежуточных величин.

Спецификация величин является информацией о типе и строении данных. Для Компоновщика требуется, чтобы в этом списке содержались: имя переменной или массива; тип; число элементов, относящихся к данной величине, и длина одного элемента.

Списки вызываемых модулей и функций, точек входа в модуль внешних имен и общих областей предназначены для установления связи между модулями по управлению и по данным. Если управление передается в модуль не через его начало, а в некоторую точку входа, то в списке вызываемых модулей должна присутствовать не точка входа, а имя модуля, содержащего эту точку входа. Окончательное установление связи в этом случае производится указанием с помощью сложного либо заменяемого имени в операторах объели, пения.

В список операторов ввода-вывода входит информация, необходимая для обработки наборов данных, находящихся на внешних устройствах. Эти наборы делятся на два вида: рабочие наборы данных и наборы данных, служащие для передачи информации между отдельными частями создаваемого агрегата. К первому виду относятся наборы данных, которые создаются и используются только в данном модуле. Ко второму виду относятся наборы данных, которые создаются или уже были созданы в одном модуле и используются другими.

Указанные паспортные данные приводятся к системному виду при занесении исходных модулей в личную библиотеку. Затем по мере необходимости для конкретного модуля вызываются паспортные данные и на их основе устанавливаются требуемые связи. Как и тексты исходных модулей, паспорта могут изменяться, но при этом из библиотеки паспортов сначала удаляется старый паспорт, а затем формируется новый в системном виде.

Все данные собираемых модулей используются интерфейсным посредником и сохраняются в Библиотеке межъязыкового интерфейса (БМИ).

2.7. Интерфейс фундаментальных и стандартных типов данных для обмена данными между модулями

Типы данных используются в описании программ в ЯП для вычисления отдельных задач и для обмена данными между модульными объектами. Если обмениваемые данные не совпадают с вызванным модулем, то делается преобразование данных к требуемому представлению формата и данным в операционных средах Интернет. Во всех ЯП используются собственные виды типов данных, которые входят в класс ФДТ.

Фундаментальные типы данных (ФДТ) и общие типы данных - General Data Types (GDT), представлены в стандартах:

- ISO (Fundamental Data Types –FDT: 1996;
- ISO/IEC General Data Types (GDT)-11404 (1999, 2007);
- ISO/IEC General Pupoused DataTypes (GPD) - 2007, 2012;
- IEEE 828 Configuration 2007, 2012. (Config, make, build, weaver).

Стандартные типы данных ЯП используются для представления и обеспечения совместимости данных модульных объектов при их обмене и преобразовании к формату и данным в операционных средах систем программирования Интернет.

Далее описываются ФДТ средства описания типов данных разной структуры (простые, структурные, сложные и структурированные), которые представлены в ФДТ и GPD и используются при сборке модульных объектов.

Приводится описание стандарта конфигурирования (Configuration) сложных программно-технических систем из готовых информационных и интеллектуальных ресурсов, работающих со стандартными типами данных (ТД) и средствами обеспечения безопасности этих ресурсов и систем, защиты данных, оценки надежности и качества ПС и программно-технических систем (ПТС).

Приводится формальное описание стандарта ФДТ и GPD, в ISO/IEC 12207 Life Cycle (ЖЦ ПО, Систем) 2006, 2012, а также представлен подход к автоматизации вариантов данных при онтологическом описании в языке DSL (Domain Spesific Language) среды Семантик Веб. Данные стандарты используются для описания функциональной структуры разных предметных областей знаний. Реализовано описание онтологии домена ЖЦ стандарта описание ISO/IEC 12207 Life Cycle и представлено на конференции в Лондоне в 2015:

- E. 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.
- Ekaterina M. Lavrischeva. Ontology of Domains. Ontological Description Software

Engineering Domain—The Standard Life Cycle, Journal of Software Engineering and Applications, 2015, 8, p.1-15. Published Online July 2015 in SciRes.http://www.scirp.org/journal/jsea.

На конференции в Лондоне комитет IEEE предложил сделать патент на онтологию ЖЦ на анг. языке. В ИСП РАН была проведена регистрация программы ЖЦ - Гос.регистрация программы ЖЦ - Свидетельство №2018615442-2018. «Метод онтологического моделирования домена ЖЦ стандарта ISO/IEC 12207.

2.7.1. ТД стандарта Фундаментальных типов данных

Наиболее используемыми средствами представления данных при реализации различных задач являются ФДТ (Fundamental Data Type) и GDT (ISO/IEC 11404 — General Data Types), независимые от ЯП. При реализации современных E-science задач сформировались новые неструктурированные данные, которые поступают с разных датчиков при исследовании земли, атмосферы, океанов и морей (Grid). Они входят в класс современных больших данных BigData и требуют нестандартных приемов для их анализа и обработки.

Тип данных — это фундаментальное понятие в теории программирования, которое определяет множество значений, набор операций, которые можно применять к этим значениям и способам их хранения.

Данные, которыми оперируют программы в современных ЯП, относятся к простым, структурным и сложным типам данных FTD. Аксиоматика FTD разработана в 1970-х гг. специалистами — С. Дейкстра, Н. Вирт, В. Турский, П. Наур, Н. Агафонов и др. В этой аксиоматике сложные ТД (массив, файл, таблица и др.) рассматриваются как агрегатная структура, которая создается из простых типов данных. Операции над значениями типа задаются аксиомами и функциями, устанавливающими взаимно однозначное соответствие переданных данных. При их несоответствии производится преобразование к соответствующим типам данных другого ЯП. Система аксиом определяет структуру множества значений типа, принадлежность ему отдельных элементов, их свойства и связь с другими типами данных.

Теория структурной организации данных воплощена в Паскале, Модула-2, Ада и др. Неструктурированные данные, которые генерируются устройствами и датчиками в различных физико-технических экспериментах, для проведения вычислений, как правило, приводятся к форматам стандартных структур данных.

ТД в языках программировании определяет множество значений, набор операций, которые можно применять к этим значениям, и способ реализации значений и выполнения операций. Данные, которыми оперируют программы, относятся к определенным ТД.

Роберт Хиндли (1960) исследовал типизацию в комбинаторной логике и в языках, основанных на теории лямбда-исчисления. В 1980-х гг. он исследовал полиморфную систему типов для языков функционального программирования.

Тип (сорт) — устойчивая и независимая совокупность элементов в выделенном множестве предметной области.

Полиморфный тип — представление набора типов как единственного типа.

Математический тип представляется множеством всех значений, принадлежащих типу, и предикатной функцией, определяющей принадлежность объекта к данному типу. Тип в математике «целое число» и не имеет ограничений, а в программировании ограничения связаны с диапазоном значений и объемом занимаемой памяти. Выход за границу целого не приводит к исключительной ситуации.

ЯП по способу определения типов данных разделяются:

1) на языки с полиморфным типом данных (например, Basic — тип вариант, Prolog, Lisp — списки). Переменные принимают значение любого типа и возвращают значения того типа. Выполнение, а + трактуется как сложение чисел, если оба члена имеют числовые значения, или как конкатенация строк, если они имеют строковые значения, и

недопустимые, если типы не совместимы. Это называется динамической типизацией, а в ООП и в теории чисел — полиморфным типом;

2) языки с неявным определением типа. Например, в Basic различают строковые типы с добавлением массивов;

3) языки с типом пользователя. Все типы практически совместимы. Их применяют в любых выражениях, а транслятор совершает необходимое преобразование. В Аде типы строго типизированы и каждая операция требует строго задания типов. Никаких преобразований типов не выполняется.

Использование ФДТ- для описания программ в ЯП

Для анализа основных типов используется теория структурной организации данных ФДТ. Она базируется на формализованных методах определению типов, основанных на аксиоматизации каждого типа и правилах выполнения операций над объектами. Система аксиом определяет структуру множества значений типа, принадлежность ему отдельных элементов, их основные свойства, связь с другими ТД. Для каждой операции, выполняемой над переменными рассматриваемого типа, определяются типы операндов и результата. Теория ФДТ* ориентирована на применение к ЯП в Паскале, Модула-2, Ада и др. Для обозначения простых ТД используются синтаксические конструкции четырех предопределенных типов данных:

- Целый((INTEGER), вещественный (REAL),
- булевой (BOOLEAN),
- символьный (CHARACTER).

Эти ТД характеризуются тем, что на уровне архитектуры большинства ЭВМ имеются средства для их обработки, так как они существуют практически во всех ЯП. Остальные сложные ТД являются производными и образуются с помощью средств конструирования новых ТД.

Все типы данных делятся на простые и структурные. К простым относятся перечислимые и числовые, к сложным относятся - *массивы, записи, множества, списки, последовательности* и т.п. Далее рассматриваются типы данных: перечислимые на основе булевого и символьного типов; числовые на основе целого и вещественного типов; массивы и записи как объекты структурных типов.

Вопросы структурной организации данных в ЯП и преобразования ФДТ в ЯП при обмене данных между разно языковыми модулями в ЯП отражены в ряде зарубежных работ:

* Вирт. Н. Систематическое программирование. Введение. — М.: Мир, 1977. — 188 с.

Вирт Н. Алгоритм + структуры данных = программа. — М.: Мир, 1985. — 406 с.

Хоар Ч. О. О структурной организации данных. — Структурное программирование. — М.: Мир, 1975. С. 92–172.

Холстед М. Х. Начало науки о программах. Перевод. — М.: Финансы и статистика, 1981. — 201 с.

Хорн Э., Виклер Ф. Проектирование модульных программных структур. — Выч. техника Соц. стран. — 1987. — Вып. 21. — С. 64–72.

Бей И. Взаимодействие разноязыковых программ. — М.; СПб.; Киев: Из-во Вильямс, 2005. — 868 с. и др.

Теория ФДТ базируется на формализованном подходе к определению данных, основанном на аксиоматизации каждого типа и правилах выполнения операций над ними. Система аксиом определяет структуру множества значений типа данного и принадлежность ему отдельных элементов, их свойств и значений, а также связи с другими типами данных в ЯП (Алгол, ПЛ/1, Фортран, Кобол).

В ФДТ все типы данных делятся на простые и структурные. В каждом ЯП имеется множество типов данных, совпадающее с отдельными типами данных других ЯП, которые относятся к простым и сложным, структурным типам данных.

Простые типы представлены в виде предопределенных данных с небольшими видоизменениями.

Структурные типы – массивы, записи, таблицы, записи, множества и др. Типы данных этих ЯП и формальные средства их описания представляются для каждого типа алгебраическими системами с учетом архитектуры ЕС ЭВМ (структура памяти, система команд и др.). В алгебраической системе под числа целого типа отводится слово в 4 байта или полуслово (2 байта) памяти. Числа вещественного типа могут занимать слово или двойное слово (8 байт). Существуют вещественные числа, занимающие 16 байтов памяти. Символьные данные представляются последовательностью байтов, каждый из которых содержит один символ. Данные булевого типа представляются и обрабатывались как битовые строки. Числа, кроме целого и вещественного типов, могут быть представлены как последовательности десятичных цифр.

ТД в язык Алгол-60 и ПЛ/1. Простые типы: целые (FIXED), вещественные (REAL), десятичные (DECIMAL) числа. Комплексные числа (COMPLEX) представлялись в виде прямого произведения и занимают в памяти последовательность из двух чисел.

Символьные и логические данные описываются как строки (байт и бит соответственно). Этот тип данных в интерпретации для этих ЯП не входили в теорию ФДТ и занимает промежуточное положение между простыми и структурными типами. С точки зрения реализации строк их относим к простым.

К структурным типам относятся массивы и записи. Такие типы данных реализованы в межязыковом интерфейсе ЯП (МЯИ) системы АПРОП.

Язык Фортран. Простые типы данных - целые (INTEGER) и вещественные (REAL) числа, а также булевые (BOOLEAN) переменные. Комплексные числа (COMPLEX) представлялись последовательностью из двух вещественных чисел. Символьный тип в языке Фортран отсутствует, однако под расположение символов могут быть отведены переменные любых типов. В Фортране имеется возможность использовать шестнадцатиричные данные. Как и символьные, они могут быть расположены в переменные любые типы.

Структурные типы языка Фортран представлены только массивами. При этом в памяти они расположены по столбцам в отличие от других ЯП, в которых массивы располагаются по строкам.

Язык Кобол. В этом языке данные простых типов могут быть с фиксированной точкой (в том числе целые), вещественные, десятичные и типа DISPLAY, который каждый символ представляется 1 байтом. Структурные типы представлены массивами и записями.

Преобразование данных для связи и сборки разноязыковых модулей обусловлена следующими причинами:

Язык ПЛ/1 для представления сложных типов данных (строки, массивы, записи) содержит дескрипторы – информационные векторы (ИВ). Другие ЯП дескрипторов не имеют. При передаче сложных типов данных в ПЛ-модуль необходимо построить ИВ, а из ПЛ-модуля по ИВ определить непосредственный адрес данных.

Массивы в Фортран-модулях расположены по столбцам, а для других ЯП – по строкам. Необходимы средства для транспортирования массивов.

Отсутствие в ЯП Фортран типа данных запись (или структура). При передаче данных такого типа необходимо наличие операций селектора и конструирования.

Отсутствие в ЯП Фортран и Кобол типов данных, эквивалентных строкам ЯП ПЛ/1. В этих случаях символьные строки могут быть представлены в виде массивов, а битовые – в виде массивов или простых типов данных.

Проблема, аналогичная преобразованию строк, возникает при передаче массивов символьных и битовых языка ПЛ/1.

Элементы комплексных чисел в ЯП ПЛ/1 могут быть любыми числовыми значениями, в ЯП Фортран – только вещественными; в ЯП Кобол комплексными числами отсутствуют. Возможны передача и обработка комплексных чисел как массива или записи.

В некоторых случаях возникает необходимость преобразования числовых величин в различные представления.

Передача управления между модулями в ЯП определяется следующими условиями:

а) при входе в ПЛ-модуль или Фортран-модуль необходимо установить среду функционирования для соответствующего ЯП;

б) при сопряжении пары модулей в ЯП, одним из которых является ПЛ-модуль, необходимо специальным образом осуществлять связь цепочек областей сохранения регистров.

2.7.2. Базовые операции для обработки типов данных ФДТ

К ФДТ относятся: множество операций для целого типа совпадает, кроме операции mod, которая может быть представлена через остальные операции.

Множество операций для вещественного типа совпадает с принятыми в системе АПРОП. Символьный и булевый типы не содержат pred и succ.

При обработке **массивов** допускается использование только отдельных элементов и возможны операции «+» и «·» в ПЛ/1. Для одномерных и двухмерных массивов, интерпретируются как соответствующие операции над векторами и матрицами.

При обработке **записей** допускается использование отдельных элементов и в особых случаях вся запись.

Множество операций над строками в языке ПЛ/1 включает операции отношения и выбора подстрок.

К особенности ЯП, упрощающих процесс информационного сопряжения модулей, относятся:

а) множества значений для простых типов данных в разных ЯП, которые совпадают и определяются размером памяти, занимаемой под переменные этих типов;

б) множества значений индексов массивов являются только отрезками целых типов;

в) в рассматриваемых ЯП отсутствует полный контроль типов операндов в операциях. Например, в символьной строке ЯП ПЛ/1 используется описание любого массива ЯП в Фортран с соблюдением необходимого соотношения длины строки и размера массива.

Проведенный анализ типов данных ЯП ОС ЕС включает перечень типов данных ЯП, особенности их представления и выполнения операций над объектами рассмотренных типов, условия и особенности сопряжения, сборки разноязыковых модулей.

Приведенные типы данных были использованы при разработки программных средств реализации функций межязыкового интерфейса (МЯИ) и библиотеки интерфейса. системы АПРОП. Реализации связей модулей проводится в 3 разделе на основе анализа ЯП.

2.7.3. Алгебраические системы обработки ФДТ

Система преобразования простых типов данных

Каждому типу данных Tαt языка lα ставится в соответствие алгебраическая система: Gαt = <Xαt , Θαt > ,

где Xαt - множество значений рассматриваемого типа; а Θαt - множество операций над объектами данного типа. Операции преобразования типов данных соответствует изоморфное отображение алгебраической системы Gαt в Gβd.

Рассматриваются все виды преобразований типов данных простых, структурных и сложных ТД. Доказывается изоморфизм алгебраических систем и при его отсутствии -ля неэквивалентность Xαt и Xβq .

К простым типам относятся перечислимые и числовые. Общее обозначение перечислимого типа имеет вид:

type T = {t1, t2, ... tn), где T – имя типа; X (x1, x2, …, xn) – множество имен всех

значений для типа T. Операции над перечислимыми типами включают бинарные операции отношения и унарные операции pred и succ, определяющие соответственно предыдущий и последующий элементы во множестве X.

Все операции относятся к множеству операций **Ω**, = <, ≤,>, ≥, ≠ в алгебраической системе:

G = <X, **Ω**>,

где X = X (x1, x2, ..., xn) множество типов значений **Ω**, = (≤, >, ≥, = , ≠) – множество операций (заменить одной операцией ≤), определяющее линейную упорядоченность элементов множества X.

Примерами перечислимых типов могут служить **булевый и символьный** типы. Множество значений булевого типа состоит из двух элементов – false и true.

Множество операций системы **Ω**, включающей операции булевой алгебры - &, V, ¬ образуют алгебраическую систему G boolean:

Gb = <Xb, **Ω** b>, Xb = {false, true}, (1)

В этой системе **Ω**b = {&, V, ¬, pred, succ, ≤}, Xb имеет тип (2, 2, 1, 1, 1;2) согласно арности соответствующих операций и предикатов. Булевый тип в ЯП записывается в виде type Tb = (false, true). Если он стандартный (Boolean), то это описание в текстах модулей может опускаться. Множество значений X символьного типа состоит из букв, цифр и специальных символов (знаков арифметических операций, знаков препинания и т. д.). Множество операций совпадает со множеством операций для любого перечисленного типа.

Алгебраическая система character символьного типа Gc имеет вид:

Gc = <Xc, **Ω**c>, Xc = {..., 'A' ..., 'X ' ..., '0', '1', '9'},

Ωc = {pred, succ, ≤}. (2)

Множество X упорядочено согласно внутреннему представлению символов для памяти ЭВМ. Алгебраическая система Gc имеет тип <1,1;2> согласно арности соответствующих операций и предикатов.

Символьный тип записывается следующим образом:

type Tc = (..., 'A', ..., 'X', …, '0', ..., '9'). Если он стандартный (CHAR), то это описание в модулях может быть опущено. Операция ord, присваивающая каждому символу его порядковый номер во множестве Xc, и char, определяющая по порядковому номеру символа его значение, являются по существу операциями преобразования типов и поэтому во множество **Ω**c не включается.

Для перечисленных типов характерны следующие аксиомы, которые в дальнейшем будут использоваться для анализа операций преобразования типов данных:

$X.min \in X, X.max \in X,$ (3)

$(\forall x \in X) \& (x \neq X.max) \Rightarrow succ(x) \in X.$

$(\forall x \in X) \& (x \neq X.max) \Rightarrow succ(x) \neq X.min$

Здесь X.min и X.max обозначают соответственно минимальный и максимальный элементы множества X.

Практическое использование **числовых типов** всегда содержит ограничения, определяемые архитектурой ЭВМ (конечное значение количества разрядов слова памяти для представления чисел) или явным описанием в модулях (для ограничения диапазона значений отдельных элементов). Поэтому все числовые типы без нарушения общности анализа могут быть рассмотрены как отрезки. В общей форме отрезок записывается в виде:

type T = (X.min … X.max). Здесь X.min и X.max обозначает соответственно минимальный и максимальный элементы отрезка. Для любого $x \in X$ выполняется условие x.min < x <x.max. Для стандартных числовых типов (INTEGER и REAL) приведенное выше описание в модулях может быть опущено. В этом случае элементы X.min и X.max не определены и зависят от конкретной реализации транслятора с ЯП на конкретной ЭВМ.

Над переменными целого типа intefer и типов, для которых целый тип является базовым, выполняются те же операции, что и в случае перечисленных типов. Кроме того,

добавляются операции целочисленной арифметики: унарный минус, +, -, ×, div (целочисленное деление) и mod - получение остатка от деления.

Алгебраическая система **целого** (integer) типа имеет вид:

Gi = <Xi, Ωi >, Xi = {Xi..min, Xi.max + 1, … Xi .max}, (4)

Ωi = {+, ×, div, -, ≤}.

Во множестве Ωi операция « - » соответствует унарному минусу. Остальные операции выражаются через операции, включенные в Ω.(real)имеет тип (2, 2, 2, 1; 2) согласно арности операций и предикатов. Форма записи целого и отрезков целого типа в ЯП имеет вид:

type Ti = (Xi.min ,…, Xi .max).

Над переменными **вещественного (real)** типа и типов, для которых вещественный тип является базовым, выполняются операции отношения и обычные арифметические операции для действительных чисел (унарный минус, +, -, ×, /).

Алгебраическая система **для real Gr** имеет следующий вид:

Gr = <Xr, Ωr>,

Xr = {x | Xr.min ≤ x ≤ Xr.max} (5)

Ωr = {+, ×, /, - , ≤ }.

Во множестве Ωr операция «–» соответствует унарному минусу. Алгебраическая система Ωr имеет тип (2, 2, 2, 1; 2) согласно арности операций и предикатов. Форма записи вещественного числа и отрезков вещественного типа в ЯП имеет вид:

type Tr = (Xr.min ,…, Xr max).

Порядок выполнения операций над любыми типами такой:

- все операнды приводятся к базовому типу;
- операция выполняется, как над объектами базового типа;
- для полученного результата выполняется обратный переход от базового к исходному типу.

Если результат принадлежит множеству значений данного типа, то операция выполняется верно. В противном случае результат операции не определен.

Система имеет вид: $(\forall x \in X) \Rightarrow T(T0(x) = x,$

$(\forall x1 \in X) \& (\forall x2 \in X) \Rightarrow (x1 \le x2) \equiv (T0(x1) \le.$

T0(x2)). (6)

Здесь T0 обозначает базовый тип для типа T. Операции T0(x) и T(x) определяют преобразование значения к соответствующему типу. В этих обозначениях выполнение арифметических операций числовых типов будет определяться следующим образом ($\oplus$, –), как любая двухместная арифметическая операция):

$(\forall X1 \in X) \& (\forall x2 \in X) \Rightarrow (x1 \oplus x2) \equiv T(T0(x1)$

$\oplus$ T0(x2)). (7)

Операции сравнения числовых типов выполняются по аксиоме (3).

Алгебраические системы преобразования структурных ТД

Отличительной особенностью структурных типов данных в сравнении с простыми является то, что они содержат несколько упорядоченных элементов, обработка которых проводится как над целыми объектами, так и на уровне отдельных элементов. Структурные типы строятся из базовых типов и отличаются функциями конструирования и механизмами обработки. В качестве основных структурных типов далее рассматриваются массивы и записи [201].

Массивы. Функция конструирования массива на основе базовых типов состоит в определении отображения из множества индексов на множество значений его элементов:

$M : I \rightarrow Y.$ (8)

Здесь I – множество индексов, Y – множество значений элементов массива. В общем

случае отображение M может не быть взаимно однозначным, если в различных элементах массива (элементы с разными индексами) содержатся одинаковые значения. Множество I является множеством значений перечислимого типа или отрезка целого типа. Элементы множества Y могут быть элементами любого типа, допускаемого в теории структурной организации данных.

Над массивами могут выполняться следующие операции:

- отношение для упорядоченных массивов (определяется как совокупность операций отношения для всех элементов массивов);
- сложение и вычитание однотипных массивов, т. е. массивов с одним и тем же множеством индексов (определяется как совокупность соответствующих операций над всеми элементами массивов с одинаковыми индексами);
- умножение двухмерных массивов по правилам умножения матриц.

Операция умножения накладывает ограничения на область значений индексов массивов, связанных с правилами умножения матриц, и не является общей для всех типов массивов. Операции сложения и вычитания выполняются только для числовых массивов. Поэтому они не входят в состав множества общих операций над массивами (array).

Алгебраическая система для ТД **array** имеет следующий вид:

$G\alpha = <X\alpha, \Omega\alpha>$,

$X\alpha = \{x \mid (\forall x1 \in X\alpha) \& (\forall x2 \in X\alpha) \Rightarrow I(x1) = I(x2)) \& (Y(x1) \cup Y(x2) \subset \overline{Y}(X\alpha))\}$,

$\Omega\alpha = \{\leq\}$. (9)

Здесь I{x) обозначает множество индексов для массива x, Y(x) - множество значений элементов массива x, $\overline{Y}$ (Xα) определяет множество значений элементов для всех массивов, принадлежащих рассматриваемому типу. Второе выражение (9) обозначает, что к данному типу принадлежат только те массивы, у которых множества индексов совпадают, а множество значений их элементов принадлежат одному и тому же множеству, характеризующему рассматриваемый тип. В обозначениях (8) отображение (8) примет следующий вид (I - постоянно для всех x):

$x : I \rightarrow Y(x), Y(x) \subset \overline{Y}(x)$. (10)

Множество Ωα состоит только из одного предиката. Поэтому алгебраическая система Gα фактически является алгебраической моделью типа (2). Тип данных массив в ЯП записывается в вид:

type Ta = array T (I) of T($\overline{Y}$),

где Ta – тип данных массив; T (I) – тип данных индексов массива; T ($\overline{Y}$) – тип данных для множества значений элементов массивов типа Ta.

Операции, определенные в (9), выполняются над массивами как над единым структурным значением. Кроме того, над элементами множеств I и Y могут выполняться операции, соответствующие их типам данных. Определим операцию селектора для элементов массива. Пусть $I' \subset I$. Через E обозначим вложение $I' \rightarrow I$. Отображение $E \bullet M : I' \rightarrow Y$ называется ограничением отображения M на x и обозначается M | I' [112–115]. Если I' состоит из одного элемента I' = {k}, то M|{k} будет определять элемент множества Y, соответствующий индексу k. Заменяя M на x (согласно определению массива), получим обозначение операции селектора для элементов массива x|{k}. В ЯП обычно элемент массива обозначается в виде x [k]. Рассматривались одномерные массивы, а многомерные массивы определяются рекурсивно. Данные типа T (Y) в описании массива Ta в свою очередь имеет вид:

type Ta = array T(I1) of T(Y1),

type T(Y1) = array T(I2) of T(Y2) эквивалентная запись:

type Ta = array T (I1 × I2) of T(Y2). (11)

В последнем предложении множество индексов массивов, принадлежащих типу Та, представлено в виде прямого произведения множеств значений для типов T(I1) и T(I2). Операция селектора будет определяться в виде {{x}|{i}}|{j}, если i $\notin$ I1 и j $\notin$ I2. В ЯП элемент массива будет обозначаться через x[i][j] или x[i, j].

Запись – структурный тип, как и массив, состоит из нескольких компонентов, которые могут быть разнородными, т.е. принадлежать различным простым или структурным типам. Функция конструирования записей представляет конкатенацию отдельных компонентов. Множество значений типа запись является прямым произведением множества значений ее компонентов. Ко множеству операций, выполняемых над записями, относятся только операции отношения. При этом предполагается, что сравниваться могут только однотипные структуры (типы компонентов сравниваемых записей и их порядок следования одинаковые) и для каждого компонента записи выполняемая операция отношения интерпретируется как соответствующая операция для типа, соответствующего данному компоненту.

Если запись состоит из n компонентов и m– компонент (т = 1, 2, ..., n) имеет тип Tvm и система имеет вид: Gvm = <Xvm, Ωvm >. Индекс vm соответствует одному из индексов для рассматриваемых в работе типов данных. Алгебраическая система **record** записи имеет вид:

Gz = <Xz, Ωz>, где

Xz = { x | (x=xv1 $\times$ . . . $\times$ xvn) & (xv1 $\in$ X v1) & . . . & (xvn $\in$ X vn)}, Ωz = {≤) (11)

Аналогично массиву, алгебраическая система Gz является моделью типа (2). Общая форма представления типа запись имеет вид:

type Tz = (Sv1:T v1; ... Svn: T vn), где, **...,** S_{v_n} – селекторы, а, ..., – типы данных для элементов записи.

Средствами ЯП **запись** (S_{v_1} , ...,) включает имена компонентов записи и имеет вид:

type T z = record,

$S_{v_1} : T^{v_1}$

:

$S_{v_n} : T^{v_n}$

end. (12)

Операции, введенные в (11), выполняются над записью, как над единым структурным значением. Для обработки отдельных компонентов вводится операция селектора, аналогично соответствующей операция для массива. Пусть I = {,}, I'⊂I.

Обозначим через E вложение I' → I. x - переменная типа запись и определяется согласно (11).

Введем в рассмотрение множество Xv = { X^{v_1} , . . .,}. Тогда между I и Xv существует однозначное соответствие M : I → Xv. Ограничение отображения M на I' обозначим через M | { S_{v_m} }. Если I состоит из одного элемента I' = { S_{v_m} }, то M | { S_{v_m} } будет определять S_{v_m} –ю запись компонента. Заменяя M на x, получаем

x |{ S_{v_m} }. В ЯП запись обозначается в виде x • S_{v_m} ,

где S_{v_m} имя соответствующего компонента. Проведенный анализ относится к фиксированным записям. Для вариантных записей, последовательность компонентов которых определяется специальным признаком, можно поступить следующим образом. Признак является переменной перечислимого типа с конечным множеством значений. Каждому конкретному значению признака соответствует определенный вид записи. Поэтому вместо одной вариантной записи рассматривается семейство фиксированных записей для каждого значения признака. Семейство конечное, так как конечно число

элементов перечислимого типа. Анализ вариантной записи будет сведен к перебору полученного семейства и обработке конкретной фиксированной записи. Поэтому в дальнейшем следует рассматривать как фиксированные записи.

Сложные типы данных в ЯП

Выше были рассмотрены основные структурные типы данных – массив, запись, которые встречаются в большинстве ЯП. Кроме них имеются и другие ТД: множества, объединения, динамические объекты, списки, последовательности, стеки, деревья и др.

Некоторые из этих типов являются стандартными в конкретных ЯП и реализуются при моделировании соответствующих структур и операций над ними. К особенностям описания сложных типов данных относятся работы [3, 4, 201]:

Некоторые сложные типы являются собственностью одного или малого числа ЯП и не имеют аналогов в других ЯП.

Анализ некоторых типов, моделируемых средствами определенного ЯП, может быть сведен к анализу базовых типов, рассматриваемых далее.

Реализация некоторых типов и выполнение операций над ними недостаточно формализованы. Чем сложнее тип данных, тем разнообразнее операции над его объектами и имеется формализованное представление этих операций в ЯП.

Множества. Аппарат множеств реализован в языке Паскаль [48]. Общая форма записи типа данных множество следующая:

typeT=powersetT0 (13)

где T определяет тип множества; T0 является базовым типом для элементов множества.

Обычно тип T это перечислимый или целый тип. Для типа T реализованы все основные операции над множествами как математическими объектами – объединение, пересечение, разность.

Операция включения, определения тождественности и нетождественности. Операции селектора задают выбор элемента типа T0 из объекта типа T. Операцией конструирования является формирование из одного или нескольких элементов типа T0 объекта типа T.

Алгебраическая система *powerset имеет* вид:

Gpz = <Xps, Ωps>,

где X= {X1, X2,…, Xn } –элементы множества,

Ωps = {+, ×, /, - , }.

Объединения. Общая форма объединения имеет вид [82]:

typeT=union(T^{v_1} ,..., T^{v_n}), (14)

где T – тип объединения; T^{v_1} , ..., T^{v_n} – базовые типы.

Алгебраическая система объединения (integration) вид:

Gin = <Xin, Ωin >,

где X= {X1, X2, Xn} –элементы множества, Ωins = {+, /, -,}.

В общем, на базовые типы ограничения не накладываются. Любой объект типа T имеет два элемента – значение и признак, по которому определяется один из типов, ..., для данного значения. Механизм реализации объединения подобен механизму реализации вариантных записей. Отличие состоит в том, что сам признак скрыт в отличие от признака вариантной записи, в которую он входит в качестве отдельного компонента. Все операции над объектами типа аналогичны операциям над вариантными записями.

Динамические объекты. Этот тип данных в различных вариантах реализован во многих ЯП: Паскаль, Ада, ПЛ/1, Си и др.

Общая форма записи для этого типа имеет вид:

type T = pointer toT0, (15)

где T – определяет; T0 – базовый тип.

Базовым может быть любой тип. Объект типа T представляет адрес объекта типа T0. Фактически ссылочный тип не является структурным, так как переменные этого типа

содержат только одно значение, как и объекты простых типов. Однако использование ссылочного типа отличается от использования простых типов. Операции над ссылочными типами не формализованы. Например, язык Паскаль допускает только одну операцию – настройку на элемент базового типа (аналогично операции присваивания). В то же время язык Си допускает над ссылочными типами операции адресной арифметики.

Списки. Списки являются конструкциями ЯП Лисп или могут реализовываться программным моделированием. Во втором случае элемент списка описывается как запись, содержащая одну или несколько компонентов ссылочного типа, которые обеспечивают связь между элементами списка. К последним могут быть применены операции, которые аналогичны операциям над фиксированными записями. Кроме них существуют операции, применяемые к целому списку: выбор начального элемента, получение остатка списка, соединение списков, их сравнение, инвертирование, поиск элементов (атомов) в списке и др. Необходимо отметить, что в ЯП Лисп эти операции принадлежат к средствам самого языка. Для языков, не имеющих стандартных средств обработки списков, аналогичные операции должны быть реализованы в виде отдельных процедур. Все это не позволяет определить фиксированное множество стандартных операций над списками, характерное для всех или большинства ЯП.

Последовательность имеет вид:

type T = sequenceT0, (16)

где T – тип *последовательности*; T0 – базовый тип.

Последовательность является одним из вариантов списка, у которого каждый элемент содержит только одну ссылочную переменную, что обеспечивает одностороннюю связь. Операции над последовательностями аналогичны операциям над списками. Одной из разновидностей последовательности является строка. Для строки каждый элемент кроме ссылочной переменной, содержит компонент символьного типа. Обработка символьных строк допускается в языке Снобол.

Стек (stek) представляется собой специальным образом организованную память с дисциплиной обработки LIFO (последним прошел – первым обработан). Обычно отдельный элемент стека принадлежит простому типу. Однако, используя средства программного моделирования, можно реализовать обработку стеков, содержащих элементы любых типов. На практике стеки реализуются в виде массивов или списков. В отличие от других дополнительных структурных типов, множество операций над стеками фиксировано и включает: инициализацию стека, занесение элемента в стек, выбор из стека, анализ элемента, находящегося на вершине стека. Операции над стеками могут быть реализованы па аппаратном уровне, стандартными средствами ЯП или программным моделированием.

Деревья. Деревья (tree) являются списковыми структурами и служат для представления графов или других аналогичных объектов. Множество операций над деревьями аналогично множеству операций над списками. Реализация этих операций зависит от конкретных видов приложений.

Кроме рассмотренных дополнительных структурных типов используются и другие – таблицы, файлы, всевозможные комбинации, перечисленных выше типов и др. Полный анализ этих типов данных и их преобразования не приводится.

2.7.4. Теория преобразования нерелевантных типов данных

Согласно рассмотренных выше ФДТ типов данных ЯП построено множество алгебраическтих систем $\sum = \sum 1, \sum 2, \sum 3$:

$\sum 1$ = {Gb, Gc ,Gi, Gr}

$\sum 2$ = {Ga, Gz, Gu, Web},

$\sum 3$ = {Gu ,Gp, Gse, Gtr…}, (17)

где $\sum 1$ включает наборы простых типов данных (t = b (bool), c (char), i (int), r (real));

$\sum 2$ – набор сложных (структурных) типов данных (t = a (array), z (record), u (union), e

(enum),…), как комбинации простых типов данных;

∑3 - набор сложных типов ∑3 = {Gu ,Gps, Gse, Gtr…} u-union, ps-powerset, in-integration, seqvance, tree и др.

Проведена изоморфное отображение данных по всем алгебраическим системам, входящих в состав всевозможных комбинаций между элементами каждого из множеств и между элементами различных и, где α≠β.

При строгом анализе значительное число отображений между системами $\cup$Xab и $\cup$ Xqb не рассматривается, что связано с отличиями некоторых множеств Ω_a^t и Ω_β^q и отсутствием изоморфнного отображения между определенными основными множествами и данными X_β^q . Чтобы не ограничивать множество операций преобразований ТД, рассмотрим следующее понятия изоморфизма преобразования данных.

Алгебраические системы Gα и Gβ – изоморфны, если существует изоморфизм между данными X_a^t и X_β^q .

Существуют изоморфизмы между X_a^t и X_β^q . Множество систем Ω_a^t и Ω_β^q различные. Если $\Omega_a^t \cap \Omega_\beta^q =$ Ω и Ω не пусто,, то рассматриваются отображения между модифицированными алгебраическими системами:

G = < X_a^t, Ω> и Gqp = < X_β^q , Ω>.

Между множествами X_a^t и X_β^q могут отсутствовать изоморфное отображение. Но на основе рассмотрения нескольких частных случаев возможного преобразования типов данных, доказывается изоморфизм.

Переходя к строгому анализу изоморфных отображений, отметим следующий очевидный факт: мощности алгебраических систем равны:

| Gta | = |Gqb | .

Под мощностью алгебраической системы Gta понимается мощность множества X_a^t [6]. Несмотря на тривиальность результата, этот факт может служить очень сильным критерием оценки правильности сборки модулей для случая, если в разных ЯП одинаковые типы данных имеют различные множества значений. Другим случаем этого критерия может служить перенос ПО с одного типа ЭВМ на другой с различными архитектурами и диапазонами целых и вещественных чисел. Для получения строгих результатов условие (19) всегда должно выполняться.

Анализируя множества Ω_a^t можно отметить, что они все содержат операции отношения. Поэтому на самих множествах X_a^t задано отношение порядка. Учитывая, что любые два элемента из X_a^t сравнимы, это отношение определяет линейный порядок. В этом случае изоморфное отображение должно сохранять отношение линейного порядка. В дальнейшем предполагается выполнение этого требования.

Пусть ∑ – множество алгебраических систем, построенных выше.

Сформулируем следующую лемму.

Лемма. Для любого изоморфного отображения φ между системами Gat и Gbq выполняются равенства φ ($X_{a.\min}^t$) =, φ ($X_{a.\max}^t$) =.

Доказательство. Для всех построенных алгебраических систем описании простых типов данных, множество значений ограниченны (числовые типы как отрезки). Согласно аксиомам, минимальные и максимальные элементы этих множеств им принадлежат. Структурные типы этих данных строятся из простых ТД с помощью конечного числа операций, а их множества значений также конечны.

Поэтому, $X^{q}_{\beta.\max}$ существуют и принадлежат множествам X^{t}_{a} и соответственно. Учитывая линейную упорядоченность этих множеств, выполнялось условие леммы. Если бы оно не выполнялось, то изоморфное отображение не сохраняло бы линейный порядок, что противоречит сделанному ранее выводу. Лемма доказана.

Чтобы не ограничивать множество операций преобразований ТД, рассмотрим следующие случаи изоморфизма:

1. Алгебраические системы G^{α} и G^{β}- изоморфны. Существуют изоморфизмы между и. Множества и совпадают.
2. Существуют изоморфизмы между и. Множества и различные. Если $\Omega^{t}_{a} \cap \Omega^{q}_{\beta} =$ Ω и Ω не пусто, то рассматриваются отображения между модифицированными алгебраическими системами:

$$G = < X^{t}_{a}, \Omega > \text{ и } GQ = < X^{q}_{\beta}, \Omega >. \qquad (18)$$

3. Между множествами и отсутствует изоморфное отображение, на основе рассмотрения нескольких частных случаев преобразования типов данных, принадлежащих этой группе.

Переходя к строгому анализу изоморфных отображений, отметим следующий очевидный факт: мощности алгебраических систем равны:

$$| G^{A} | = |G^{Q}| \qquad (19)$$

Под мощностью алгебраической системы G^{t}_{a} понимается мощность множества X^{t}_{a} [6]. Несмотря на тривиальность результата, этот факт может служить очень сильным критерием оценки правильности сопряжения модулей для случая, если в разных ЯП одинаковые типы данных имеют различные множества значений. Другим случаем этого критерия может служить перенос ПО с одного типа ЭВМ на другой с различными архитектурами и диапазонами целых и вещественных чисел. Для получения строгих результатов условие (19) всегда должно выполняться.

Анализируя множества Ω^{t}_{a} можно отметить, что они все содержат операции отношения. Поэтому на самих множествах X^{t}_{a} задано отношение порядка. Учитывая, что любые два элемента из X^{t}_{a} сравнимы, это отношение определяет линейный порядок. В этом случае изоморфное отображение должно сохранять отношение линейного порядка. В дальнейшем предполагается выполнение этого требования.

Пусть $\sum$ – множество алгебраических систем, построенных выше. Сформулируем следующую лемму.

Лемма 1. Для любого изоморфного отображения φ между системами $G_a{}^t$ и $G_b{}^q$ выполняются равенства $\varphi (X^{t}_{a.\min}) =, \varphi (X^{t}_{a.\max}) =$.

Доказательство. Для всех построенных алгебраических систем описания простых типов данных, множество значений ограниченны (числовые типы рассматриваются как отрезки). Согласно аксиомам минимальные и максимальные элементы этих множеств им принадлежат. Структурные типы этих данных строятся из простых с помощью конечного числа операций, а их множества значений также конечны. Поэтому, $X^{q}_{\beta.\max}$ существуют и принадлежат множествам X^{t}_{a} и соответственно. Учитывая линейную упорядоченность этих множеств, выполнялось условие леммы. Если бы оно не выполнялось, то изоморфное отображение не сохраняло бы линейный порядок, что противоречит сделанному ранее выводу. Лемма доказана.

Рассмотрим операцию преобразования между *перечислимыми* и типами на примере символьных типов. Пусть $G^{A} = G^{a}{}_{b}$, описывают два символьных типа в некоторых ЯП. Ранее были получены результаты о том, что изоморфное отображение должно сохранять порядок и $|G^{A}| = |G^{a}{}_{b}|$. Имеет место теорема.

Теорема 1. Пусть φ – отображение системы G^A в систему G^a_b множества $\sum_1$. Для того чтобы отображение φ было изоморфным, необходимо и достаточно, чтобы φ изоморфно отображало X^c_α и X^c_β с сохранением линейного порядка.

Необходимость. Пусть φ – изоморфизм. Тогда при отображении сохраняются все операции множества $\Omega = \Omega^c_\alpha = \Omega^c_\beta$, в том числе и операция отношения, которая определяет линейный порядок на множествах X^c_α и X^c_β.

Достаточность. Пусть φ изоморфно отображает X^c_α на X^c_β с сохранением линейного порядка. Необходимо проверить сохранность операций, указанных в (3.8). Операция отношения выполняется согласно упорядоченности. Рассмотрим операцию succ.

Согласно лемме 1 $\varphi(X^c_{\alpha.\min}) = X^c_{\beta.\min}$. Последовательно применяя операцию succ к данному равенству и учитывая линейную упорядоченность множества X^c_α на X^c_β ($x <$ succ(x)), получим, что для любого $x^c_\alpha \in X^c_\alpha$ и $x^c_\alpha \neq X^c_{\alpha.\max}$ из равенства $\varphi(x^c_\alpha) = x^c_\beta$ следует равенство:

$$\varphi(\text{succ}(x^c_\alpha)) = \text{succ}(x^c_\beta). \qquad (20)$$

Для операции pred рассмотрение аналогичное. Согласно лемме 1 $\varphi(X^c_{\alpha.\max}) = X^c_{\beta.\max}$. Последовательно применяя операцию pred, получим, что для любого $x^c_\alpha \in X^c_\alpha$ и $x^c_\alpha \neq X^c_{\alpha.\min}$ из равенства $\varphi(x^c_\alpha) = x^c_\beta$, где $x^c_\beta \in X^c_\beta$ следует равенство: $\varphi(\text{pred}(x^c_\alpha)) = \text{pred}(x^c_\beta)$. (21)

Теорема доказана. Она позволяет использовать в качестве операции преобразования любое изоморфное отображение, удовлетворяющее условиям теоремы, независимо от природы элементов множеств X^c_α и. Если в вызывающем модуле параметр имеет тип: type *TV* = ('A', 'B', 'C'), в вызываемом модуле формальный параметр описан в виде: type *TF* = (1, *2,* 3),
то в качестве операции преобразования можно выбрать любое изоморфное отображение вида: φ ('A') = 1, φ ('B') = 2, φ ('C') = 3.

Рассмотрим операцию преобразования между булевыми типами. Пусть $G^b_a = G^b_b$– алгебраические системы, описывающие эти типы. Справедлива следующая теорема.

Теорема 2. Любой изоморфизм φ между системами G^b_a и G^c_b является тождественным изоморфизмом*:* $\varphi(X^\beta_{\alpha.false}) = X^b_{\beta.false}$,

$$\varphi(X^b_{\alpha.true}) = X^b_{\beta.true} \qquad (22)$$

где через $x^\beta_{\alpha.false}(X^b_{\beta.false})$ и $x^\beta_{\alpha.true}(X^b_{\beta.true})$ обозначены соответственно элементы false и true *в множестве* $X^b_\alpha(X^b_\beta)$.

Доказательство. Согласно теореме 1 для отображения между G^b_a и G^c_b выполняются все операции при этом всегда справедливо неравенство $X^\beta_{\alpha.false} < X^b_{\alpha.true}$. Поэтому, учитывая, что φ сохраняет порядок, единственно возможным изоморфизмом является отображение (22). Теорема доказана.

В общем случае X^β_α и X^b_β – различные множества (например, в ЯП ПЛ/1 в качестве булевых переменных используются битовые строки). Если $X^\beta_\alpha = X^b$, то φ является тождественным автоморфизмом.

Рассмотрим операции преобразования для *числовых* типов. Введем стандартные обозначения для множеств целых чисел Z, рациональных *Q* и действительных (вещественных) *R.* Ранее предполагалось, что основные множества алгебраических систем,

соответствующие числовым типам, ограничены. Для дальнейшего анализа будем считать, что при необходимости границы числового диапазона могут быть расширены.

Теорема 2. Алгебраические системы G^t_a и G^t_b, соответствующие целым типам, имеют изоморфизм φ между ними. Изучим свойства этого изоморфизма. Для операции сложения выполняется равенство:

$$\varphi(x^i_\alpha + y^i_\alpha) = \varphi(x^i_\alpha) + \varphi(y^i_\alpha).$$

Полагая = *0*, получаем, что $\varphi() = \varphi(x^i_a) + 0$. Так как – произвольное, то это равенство выполняется только при $\varphi(0) = 0$.

Для операции умножения необходимо выполнить равенство:

$$\varphi(\cdot) = \varphi() \cdot \varphi(y^i_\alpha).$$

Полагая $y^i_\alpha = 1$, получаем, что $\varphi() = \varphi(x^i_\alpha) \cdot \varphi(1)$. Здесь $\varphi(x^i_\alpha)$ – целое число. Если оно не равно нулю, то выполним операцию целочисленного деления и получим, что $\varphi(1) = 1$. Окончательно можно записать: $\varphi(0)=0, \varphi(1)=1,$ (23)

где φ – изоморфизм между G^t_a и G^t_b.

Если $X^i_\alpha \subset Z$ и $X^i_\beta \subset Z$, то $\forall\ x^i_\alpha \in X^i_\alpha\ \exists\ x^i_\beta \in X^i_\beta : \varphi(x^i_\alpha) = x^i_\beta$.

Рассмотрим произвольное $x^i_\alpha < 1$. Применяя результаты (23), получаем: $\varphi(x^i_\alpha) = \varphi(x^i_\alpha - 1) + \varphi(1) = \varphi(x^i_\alpha - 1) + 1 = \ldots = \varphi(1) + x^i_\alpha - 1 =$.

Для *<0* и $|x^i_\alpha| => 0$, имеется следующий результат:

$$0 = \varphi(y^i_\alpha + (-y^i_\alpha)) = \varphi() + \varphi(-) = y^i_\alpha + \varphi(-y^i_\alpha).$$

Отсюда следует, что $\varphi(-) = -y^i_\alpha$ или $\varphi(x^i_\alpha) =$. Окончательно можно отметить, что для любого целого необходимо равенство:

$$\varphi(x^i_\alpha) = x^i_\alpha \qquad (24)$$

Представить изоморфное отображение φ между алгебраическими системами G^r и G^r_b, для соответствующих *вещественных* ТД, где X^r_α и X^r_β – подмножества *R*. Так как $Z \subset R$, то результат (24) справедлив и для подмножества вещественных чисел, совпадающего с *Z*. Если $x^r_\alpha \in Q$, т. е. $x^r_\alpha = m/n$, где *m* и *n* –целые числа, то представим следующее:

$m = \varphi(m) = \varphi(n\frac{m}{n}) = \varphi(n) * \varphi(\frac{m}{n}) = n * \varphi(\frac{m}{n})$, из чего следует:

$$\varphi(\frac{m}{n}) = \frac{m}{n}. \qquad (25)$$

Это соотношение выполняется для любых рациональных чисел. На основании известной теоремы из теории множеств (см., [6, 28]) Любое вещественное число *x* можно с любой заранее определенной точностью ε представить в виде рационального числа *m/n* так, что $|x - \frac{m}{n}| < \varepsilon$. Это тем более справедливо для представления вещественных чисел в ЭВМ, так как точность представления зависит от количества разрядов машинного слова памяти. Таким образом, для любых $x \notin R$ выполняется: $\varphi(x) = x$. (26)

Учитывая, что множества Z и *R* являются базовыми для основных множеств X^i_α и X^r_α алгебраических систем G^t_a и G^r_a, определенных в п. 3.4, можно доказать следующую теорему.

Теорема 3. Любой изоморфизм между алгебраическими системами соответствующими числовым типам, является тождественным автоморфизмом.

В доказательстве использовался тот факт, что 0 и 1 принадлежат основным множествам.

Это существенное предположение, так как некоторые числовые отрезки не содержат этих значений. Такие типы данных должны анализироваться в частном порядке.

Выше полученные результаты, следующие из анализа изоморфных отображений алгебраических систем, позволяют перейти к рассмотрению изоморфизма основных множеств и при условии отличия Ω^t_α и. Если $\Omega^t_\alpha \cap \Omega^q_\beta \neq 0,$ то возможны следующие виды преобразований ТД:
символьный – в целый; символьный – в булевой; целый – в символьный; целый – в булевый; булевый – в целый; булевый – в символьный.

В приведенных видах преобразованиях отсутствует вещественный тип. Это связано с тем, что множества значений символьных, булевых и целых чисел имеют дискретный характер, множество вещественных чисел по своей сути непрерывно (при реализации на ЭВМ это множество также дискретно, что связано с особенностями структуры основной памяти).

Рассмотрим преобразование символьного типа в целый и целого в символьной.

Пусть G^c_a и G^i_b– алгебраические системы, описывающие эти типы. Пересечение множеств операций $\Omega = \Omega^c_\alpha \cap \Omega^i_\beta$ имеет вид:

$$\Omega = \{\text{pred, succ}, \leq\}. \qquad (27)$$

Этот вид в точности совпадает со множеством операций для любого перечислимого типа. Поэтому мы можем воспользоваться результатами теоремы 1 и выбрать любое изоморфное отображение множеств X^c_α и X^i_β, сохраняющее линейный порядок. В качестве множества X^i_β может быть выбран любой отрезок целого типа, для которого

$$| X^c_\alpha | = X^i_{\beta.\max} - X^i_{\beta.\min} + 1, \qquad (28)$$

т.е., если выполняется условие (19) о равенстве мощностей основных множеств для алгебраических систем, особый интерес представляют отображения, ставящее каждому символу его порядковый номер в алфавите (функция ord), и отображение, определяющее по порядковому номеру символа в алфавите его значение (функция chr). В этом случае функции ord и chr – это операции преобразования символьных и целых типов.

Преобразование целого в булевый и булевого в целый выполняется с помощью алгебраических систем G^i_a и G^c_b, описывающих соответственно целый и булевый типы. Пересечение множеств операций $\Omega = \Omega^i_\alpha \cap \Omega^b_\beta$ совпадает с (27). Преобразование символьного в булевый и булевого в символьный практически редко используется. Для них справедливы формальные выводы, как и в случае преобразования между символьными и целыми типами. Возможно только применение операций для перечислимых типов. Множества значений каждого из типов для данного случая будут содержать по 2 элемента.

Системы преобразования и отображения структурных ТД

Рассмотрим операции преобразования для массивов и записей. Анализируя множества операций для соответствующих алгебраических систем (15) и (17), можно отметить, что при преобразовании должен сохраняться только линейный порядок. Возможны следующие виды преобразований: массив – в массив, запись – в запись, массив – в запись, запись – в массив.

Рассмотрим первое из них. Пусть G^b_a и G^i_b– две алгебраические системы, описывающие массивы. Пусть φ – их изоморфизм, в котором сохраняется линейный порядок. Это означает, что () & () & $(x^a_{\alpha_1} \leq x^a_{\alpha_2}) \Rightarrow \varphi (x^a_{\alpha_1}) \leq \varphi (x^a_{\alpha_2})$ (29)

Согласно (16), $x^a_{\alpha_1}$ и $x^a_{\alpha_2}$ – функции отображения. Выполнение отношения ≤ для функций означает выполнение этого отношения для всех элементов области определения этих функций.

Рассмотрим выражения для функций:

$\varphi(x^a_{\alpha_1})$ и $\varphi(x^a_{\alpha_2})$, из

$x^a_{\alpha_1} : I^a_\alpha \to Y(x^a_{\alpha_1}), Y(x^a_{\alpha_1}) \subset \overline{Y}(X^a_\alpha), \quad x^a_{\alpha_2} : I^a_\alpha \to Y(x^a_{\alpha_2}), Y(x^a_{\alpha_2}) \subset \overline{Y}(X^a_\alpha).$

Пусть выполняется: $\varphi_i: I^a_\alpha \to, \varphi_y: \overline{Y}(X^a_\alpha) \to \overline{Y}(X^a_\beta)$,

где φ_i и φ_y – изоморфные отображения, соответствующие множествам индексов i и значениям y элементов массива. Через E_1 обозначим вложение $Y(x^a_{\alpha_1}) \to \overline{Y}(X^a_\alpha)$, а через E_2 – вложение $Y(x^a_{\alpha_2}) \to \overline{Y}(X^a_\alpha)$. Тогда $\varphi(x^a_{\alpha_1})$ и $\varphi(x^a_{\alpha_2})$ будут определяться как ограничения отображения φ_y на соответствующие подмножества и обозначаться через $\varphi_y|Y(x^a_{\alpha_1})$ и $\varphi_y|Y(x^a_{\alpha_2})$ соответственно. Последнее неравенство в (29) будет эквивалентно $\varphi_y \mid Y(x^a_{\alpha_1}) \le \varphi_y \mid Y(x^a_{\alpha_2})$.

Расширим смысл обозначения в (13) и φ_i и φ_y будут обозначать изоморфные отображения между алгебраическими системами, для которых I^a_α, I^a_β, $\overline{Y}(X^a_\alpha), \overline{Y}(X^a_\beta)$ являются основными множествами. Тогда имеет место следующая теорема.

Теорема 4. Пусть G^a_a и G^a_b – две системы из $\sum_2$, соответствующие типам данных массив, и φ_i, φ_y – изоморфные отображения, сохраняющие линейный порядок.

Тогда изоморфизм φ между G^a_a и G^a_b полностью определяется отображениями φ_i, и φ_y. Это означает, что анализ отображения φ может быть сведен к анализу φ_i и φ_y и значительно упрощается задача построения изоморфного отображения.

Доказательство. Пусть имеются отображения φ_i и φ_y. Отображение φ_i сохраняет линейный порядок, и, следовательно, сохраняется взаимная упорядоченность отдельных элементов массивов относительно друг друга. Рассмотрим выражение (12). Оно достаточно для одного элемента массива.

Пусть $k \in I^a_\alpha$. Ему будет соответствовать $\varphi_i(k) \in I^a_\beta$. Пусть выполняется неравенство $x^a_{\alpha_1}(k) \le x^a_{\alpha_2}(k)$. Применим к обеим частям неравенства отображение φ_y в смысле (14). Упорядоченность не нарушается при применении операций конструирования структурных типов. Поэтому независимо от типа данных для множеств $\overline{Y}(X^a_\alpha), \overline{Y}(X^a_\beta)$ отображение φ_y также сохраняет линейный порядок. Отсюда справедливо неравенство: $\varphi_y \mid Y(x^a_{\alpha_1})(k)) \le \varphi_y \mid Y(x^a_{\alpha_2})(k))$, при этом каждому значению $\varphi_i(k)$ соответствует $\varphi_y \mid Y(x^a_\alpha)(k))$.

Теорема доказана.

Таким образом, φ является двух компонентной последовательностью $\varphi=(\varphi_i, \varphi_y)$. Преобразование массивов определяется преобразованиями ТД для индексов и множеств значений для элементов массивов, принадлежащих рассматриваемому типу.

Рассмотрим преобразование типа *запись – запись*. Пусть G^z_a и G^z_b– две алгебраические системы, описывающие записи. В теории структурной организации данных [151] множество значений для записей определяется как прямое произведение множеств значений их отдельных компонентов. Данный подход к описанию записей содержит существенный недостаток. Рассмотрим множество значений алгебраической системы для записи.

Пусть $n = 3$ и имеется три описания:

а) type $T^{z_1} = (: T^{v_3})$,

б) type $T^{z_2} = (: S' : T')$, type $T' = (: T^{v_3})$,

в) type $T^{z_3} = (S': T'; T^{v_3})$, type $T' = (: T^{v_2})$.

Несмотря на различия в описании трех типов записей,, T^{v_3} множества их значений совпадают относительно задачи преобразования данных. Поэтому имеется неопределенность при реализации операции преобразования вида запись – в запись. Для устранения этой неопределенности введем дополнительное ограничение, которое состоит в том, что между множествами типов данных обеих записей можно установить взаимно однозначное соответствие при условии:

– количество компонентов в обеих записях одинаково;

– для каждого из компонентов первой записи существует только один соответствующий компонент второй записи.

Операции отношения над записями будут выполняться с учетом выбранного соответствия (последовательности сравниваемых компонентов определяются данным соответствием). В этом случае анализ преобразования записи сводится к анализу преобразований типов отдельных компонентов. Если изоморфные отображения между отдельными компонентами сохраняют линейный порядок для самих компонентов, то с учетом сказанного доказывается следующая теорема.

Теорема 5. Пусть G^z_a и G^z_b – две алгебраические системы $\sum_2$, соответствующие типам данных запись, и $x^z_\alpha \in X^z_\beta$, $x^z_\beta \in X^z_\beta$.

Если между последовательностями компонентов и существует взаимно однозначное соответствие, то изоморфизм между G^z_a и G^z_b определяется изоморфными отображениями алгебраических систем, соответствующих записям.

Рассмотрим смешанные виды преобразований между структурными типами. Пусть G^a_a и G^z_b– алгебраические системы, описывающие массив и запись соответственно. Множества операций для этих систем совпадают. Поэтому определим условия изоморфного отображения для множеств значений этих типов. Как и в случае преобразования записей, для данного вида преобразования имеется некоторая неопределенность, которую можно устранить, введя следующее ограничение: количество элементов массива должно совпадать с количеством элементов записи.

Если рассматривать массив как запись, у которой множество селекторов совпадает со множеством индексов, а типы всех компонентов одинаковы. Для преобразования массива – в запись можно использовать результаты теоремы 5.

Преобразование вида *запись – в массив* требует, чтобы типы всех компонентов записи были одинаковыми, а сама запись была представлена в виде массива. Множество значений компонентов образует множество значений элементов массива *Y*. Множество индексов строится простым переупорядочиванием селекторов компонентов записи: *i*-й компонент записи ставится в соответствие *φ(i)*-й элемент во множестве *J* (*φ* – взаимно-однозначное отображение). Само множество *I* будет множеством значений перечислимого типа. В этом случае можно использовать теоремы 4.

$$\varphi(\text{succ}(x^c_\alpha)) = \text{succ}(x^c_\beta). \qquad (30)$$

Для операции pred рассмотрение аналогичное. Согласно лемме 1 $\varphi(X^c_{\alpha.\max}) = X^c_{\beta.\max}$. Последовательно применяя операцию pred, получим, что для любого $x^c_\alpha \in X^c_\alpha$ и $x^c_\alpha \neq X^c_{\alpha.\min}$ из равенства $\varphi(x^c_\alpha) = x^c_\beta$, где $x^c_\beta \in X^c_\beta$ следует равенство:

$$\varphi(\text{pred}(x^c_\alpha)) = \text{pred}(x^c_\beta). \qquad (31)$$

Теорема доказана. Она позволяет использовать в качестве операции преобразования использовать любое изоморфное отображение, удовлетворяющее условиям теоремы, независимо от природы элементов множеств X^c_α и X^c_β. Если в вызывающем модуле параметр имеет тип:

type *TV* = ('A', 'B', 'C'), в вызываемом модуле формальный параметр описан в виде: type *TF* = (1, *2,* 3),
то в качестве операции преобразования можно выбрать любое изоморфное отображение вида: φ ('A') = 1, φ ('B') = 2, φ ('C') = 3.

Рассмотрим операцию преобразования между булевыми типами. Пусть $G^b_a = G^b_b$– алгебраические системы, описывающие эти типы. Справедлива следующая теорема.

Теорема 6. Любой изоморфизм φ между системами G^b_a и G^c_b является тождественным изоморфизмом*:* $\varphi\,(X^{\beta}_{\alpha.false}) = X^{b}_{\beta.false}$,

$$\varphi\,(X^{b}_{\alpha.true}) = X^{b}_{\beta.true} \qquad (32)$$

где через $x^{\beta}_{\alpha.false}(X^{b}_{\beta.false})$ и $x^{\beta}_{\alpha.true}(X^{b}_{\beta.true})$ обозначены соответственно элементы false и true *в множестве* $X^{b}_{\alpha}(X^{b}_{\beta})$.

Доказательство. Согласно теореме 1 для отображения между G^b_a и G^c_b выполняются все операции при этом всегда справедливо неравенство $X^{\beta}_{\alpha.false} < X^{b}_{\alpha.true}$. Поэтому, учитывая, что φ сохраняет порядок, единственно возможным изоморфизмом является отображение (22). Теорема доказана.

В общем случае X^{β}_{α} и X^{b}_{β} – различные множества (например, в ЯП ПЛ/1 в качестве булевых переменных используются битовые строки). Если $X^{\beta}_{\alpha} = X^{b}$, то φ является тождественным автоморфизмом.

Рассмотрим операции преобразования для *числовых* типов. Введем стандартные обозначения для множеств целых чисел Z, рациональных *Q* и действительных (вещественных) *R.* Ранее предполагалось, что основные множества алгебраических систем, соответствующие числовым типам, ограничены. Для дальнейшего анализа будем считать, что при необходимости границы числового диапазона могут быть расширены.

Алгебраические системы G^t_a и G^t_b, соответствующие целым типам, и изоморфизм φ между ними. Изучим свойства этого изоморфизма. Для операции сложения выполняется равенство:

$$\varphi(x^i_{\alpha} + y^i_{\alpha}) = \varphi(x^i_{\alpha}) + \varphi(y^i_{\alpha}).$$

Полагая = *0*, получаем, что $\varphi(x^i_{\alpha}) = \varphi(x^i_a) + 0$. Так как x^i_{α} – произвольное, то это равенство выполняется только при $\varphi(0) = 0$.

Для операции умножения необходимо выполнить равенство

$$\varphi(x^i_{\alpha} \cdot y^i_{\alpha}) = \varphi(x^i_{\alpha}) \cdot \varphi(y^i_{\alpha}).$$

Полагая $y^i_{\alpha} = 1$, получаем, что $\varphi\,() = \varphi\,() \cdot \varphi(1)$. Здесь $\varphi\,()$ – целое число. Если оно не равно нулю, то выполним операцию целочисленного деления и получим, что $\varphi(1) = 1$. Окончательно можно записать:

$$\varphi(0)=0, \varphi(1)=1, \qquad (33)$$

где φ – изоморфизм между G^t_a и G^t_b.

Если Z и Z, то: $\varphi\,(x^i_{\alpha}) =$.

Рассмотрим произвольное $x^i_{\alpha} < 1$. Применяя результаты (33), получаем: $\varphi\,(x^i_{\alpha}) = \varphi(x^i_{\alpha} - 1) + \varphi(1) = \varphi(x^i_{\alpha} - 1) + 1 = \ldots = \varphi(1) + x^i_{\alpha} - 1 =$.

Для < 0 и $|x^i_{\alpha}| \Rightarrow 0$, имеется следующий результат:

$$0 = \varphi\,(y^i_{\alpha} + (-y^i_{\alpha})) = \varphi\,() + \varphi(-) = y^i_{\alpha} + \varphi(-y^i_{\alpha}).$$

Отсюда следует, что $\varphi(-) = -y^i_{\alpha}$ или $\varphi\,(x^i_{\alpha}) =$. Окончательно можно отметить, что для любого целого необходимо равенство:

$\varphi(x_\alpha^i) = x_\alpha^i$ (34)

Представить изоморфное отображение φ между алгебраическими системами G^r и $G^r{}_b$, для соответствующих *вещественных* ТД, где X_α^r и X_β^r – подмножества *R.* Так как $Z \subset R$, то результат (24) справедлив и для подмножества вещественных чисел, совпадающего с *Z.* Если $x_\alpha^r \in Q$, т. е. $x_\alpha^r = m/n$, где *m* и *n* –целые числа, то представим следующее:

$m = \varphi(m) = \varphi(n\frac{m}{n}) = \varphi(n) * \varphi() = n * \varphi(\frac{m}{n})$, из чего следует:

$$\varphi() = \frac{m}{n}. \quad (35)$$

Это соотношение выполняется для любых рациональных чисел. На основании известной теоремы из теории множеств (см., [6, 28]) Любое вещественное число *x* можно с любой заранее определенной точностью ε представить в виде рационального числа *m/n* так, что $|x - \frac{m}{n}| < \varepsilon$.

Это тем более справедливо для представления вещественных чисел в ЭВМ, так как точность представления зависит от количества разрядов машинного слова памяти. Таким образом, для любых *x* R выполняется:

$\varphi(x) = x.$ (36)

Учитывая, что множества Z и *R* являются базовыми для основных множеств и X_α^r алгебраических систем $G^t{}_a$ и $G^r{}_a$, определенных в п. 2.4, можно доказать следующую теорему.

Теорема 3. Любой изоморфизм между алгебраическими системами соответствующими числовым типам, является тождественным автоморфизмом.

При доказательстве используется тот факт, что 0 и 1 принадлежат основным множествам. Это существенное предположение, так как некоторые числовые отрезки не содержат этих значений. Такие типы данных должны анализироваться в частном порядке.

Выше полученные результаты, следующие из анализа изоморфных отображений алгебраических систем, позволяют перейти к рассмотрению изоморфизма основных множеств X_α^t и X_β^q при условии отличия Ω_α^t и Ω_β^q. Если $\Omega_\alpha^t \cap \Omega_\beta^q \neq 0$, то возможны следующие виды преобразований ТД: символьный – в целый; символьный – в булевый; целый – в символьный; целый – в булевый; булевый – в целый; булевый – в символьный.

В приведенных видах преобразованиях отсутствует вещественный тип. Это связано с тем, что множества значений символьных, булевых и целых чисел имеют дискретный характер, множество вещественных чисел по своей сути непрерывно (при реализации на ЭВМ это множество также дискретно, что связано с особенностями структуры основной памяти).

Преобразование символьного типа в целый и целого в символьной.

Пусть $G^c{}_a$ и $G^i{}_b$– алгебраические системы, описывающие эти типы. Пересечение множеств операций $\Omega = \Omega_\alpha^c \cap \Omega_\beta^i$ имеет вид:

$\Omega = \{\text{pred, succ}, \leq\}.$ (37)

Этот вид в точности совпадает со множеством операций для любого перечислимого типа. Поэтому мы можем воспользоваться результатами теоремы 1 и выбрать любое изоморфное отображение множеств X_α^c и X_β^i, сохраняющее линейный порядок. В качестве множества X_β^i может быть выбран любой отрезок целого типа, для которого

$$|X_\alpha^c| = X_{\beta.\max}^i - X_{\beta.\min}^i + 1, \quad (38)$$

т.е., если выполняется условие (19) о равенстве мощностей основных множеств для алгебраических систем, особый интерес представляют отображения, ставящее каждому символу его порядковый номер в алфавите (функция ord), и отображение, определяющее по порядковому номеру символа в алфавите его значение (функция chr). В этом случае функции ord и chr – операции преобразования символьных и целых типов. Рассмотрим преобразование целого в булевый и булевого в целый. Алгебраические системы G^{i}_{a} и G^{c}_{b} описывают соответственно целый и булевый типы. Пересечение множеств операций $\Omega = \Omega^{i}_{\alpha} \cap \Omega^{b}_{\beta}$ совпадает с (27). Поэтому в данном случае применимы предыдущие результаты и над целым и булевым типами возможны только операции как над перечислимыми типами.

Глава 3

Методы преобразования данных в АПРОП средствами Библиотеки межъязыкового интерфейса (БМИ)

Метод сборки разноязыковых программных модулей реализован в рассматриваемой здесь в системе АПРОП.

В 1975году академик В.М. Глушков высказал идею на ученом совете ИК АН УССР о том, что «программы будут собираться конвейерным способом, как автомобили собираются из готовых деталей на фабрике Форда через болты и гайки. При исследовании этой идеи сформировалось понятие конвейерной сборки программных элементов с помощью операции link, которая объединяет разнородные модули через интерфейс. Метод сборки обеспечивает связывание модульных объектов по данным, передаваемым между ними.

Метод сборки и библиотека интерфейса БМИ (64 примитивов преобразования неэквивалентных данных) реализованы в ОС ЕС или IBM 360. На основе системы АПРОП созданы на ВПК (В.В. Липаев, МНИИПА, Минрадиопрома СССР) комплексы РУЗА, ПРОМЕТЕЙ, ЯУЗА и специализированные ЭВМ для авиационной и космической промышленности (1979-1985). В 1985 СМ СССР наградил базовый коллектив Государственной премией. А все созданные системы и средства БМИ переданы в ЕрНУЦ. Система АПРОП передана в 52 организации СССР, включая прибалтийские, средне-азиатские, грузинскую, молдавскую республики и большие города в СССР.

Межмодульный интерфейс реализованный в ВПК, обсуждался на конференции «Интерфейс СЭВ-1987» и стал базовым понятием в технологии программирования. Автор получил грамоту Евроконференции «Интерфейс СЭВ» за определение межмодульного и межязыкового интерфейса.

Сборка производится в **операционной среде** ЯП с помощью вызова link других модулей, с соответствующим обращением в тексте описания CALL/RMI, которые обрабатываются компиляторами с ЯП и включают:

- данные для связи и управления модулей, находящихся как стандартные функции обработки данных в библиотеках ОС среды;
- системные данные ОС, списки идентификаторов функций, стеки точек передач и возврата управления;
- динамические данные из списка областей памяти ОС с регистрами.

Сборка разнородных объектов в ЯП проводился по **графовой структуре** приложения, систем [1-5]. Теория графов обеспечивает связывание модульных элементов с помощью математических операций (объединения, соединения, разности и др.) в сложные структуры (комплекс, агрегат, система и др.) и доказательство их достижимости. Эта теория внедрена в проектах ВПК поддержана академиком А.П. Ершова (ИПИ СО АН СССР), ученики которого развивали теорию графов для программирования трансляторов и операционных систем [7-14]. Начиная с 2003 года, теория графов начала применяться для моделирования сложных систем по методу объектного моделирования из объектов (ОМ) и компонентов [15] с применением новых операций сборки (assembler, config, make) согласно стандарта ISO/IEC 828 (1996, 2012, Configuration) b ISO/IEC GDT 12404 -2007. На основе графового описания программных структур проводится сборка объектов (компонентов) и проводится изоморфное отображения неэквивалентных общих 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 сборочного графового моделирования представлен в зарубежных и отечественных журналах в 2018-2020 и перспективе будет использоваться в таких прикладных областях, как медицина, генетика, биология и др.

Библиотека межъмодульного интерфейса (БМИ) является составной частью метода сборки, разработанной специально для реализации связей между модулями, записанными на ЯП ОС ЕС. Все макроопределения (МО) представлены на языке Ассемблера и функционально многие из них идентичны соответствующим макроопределениям ОС ЕС и ориентированы на использование для класса ЯП (Algol-60, Fortran, GL/1, Smalltalk, Snobol, Basic) ОС ЕС. Для таких макроопределений сохранена та же символика, с той лишь разницей, что добавляется в качестве последней буквы Р (CALLP, LINKP и т. п.).

По ориентации на выполнение макроопределения делятся на зависимые и независимые.

К зависимым макроопределениям относятся такие, которые связаны с системными таблицами АПРОП, т. е. полученные после генерации соответствующие макрорасширения в процессе своей работы обращаются к таблицам или служебным программам системы АПРОП. К зависимым макроопределениям, в частности, относятся макроопределения, обеспечивающие режим подзадач и управление моделированием работы агрегата. Поскольку применение макроопределений неавтоматизированным способом требует разработки таблиц с затратой большой работы пользователя на детальное их описание, то эти макроопределения рассматриваться не будут. Ими целесообразно пользоваться, обращаясь к средствам системы АПРОП.

К независимым относятся такие макроопределения, которыми можно пользоваться автономно, включая их в библиотеку макроопределений Ассемблера или располагая требуемые макроопределения в пакет заданий для ОС ЕС. В обоих этих случаях разрабатываются соответствующие управляющие операторы языка управления заданиями, обеспечивающие получение загрузочного модуля для выполнения. Именно такие элементы библиотеки БМИ и будут рассматриваться в этой главе.

В состав независимой части библиотеки БМИ входят: библиотека макроопределений (БМО) и библиотека загрузочных модулей (БЗМ).

В БМО размещены макроопределения управления и преобразования данных. Создание этой библиотеки обусловлено спецификой выходного кода компиляторов и регламентации регистров для обеспечения связей по управлению и по данным между вызывающим и вызываемым модулями.

Входящие в состав БМО макроопределения делятся на два вида: специальные и универсальные. К специальным макроопределениям относятся макроопределения, которые позволяют установить среду конкретного языка программирования, обеспечить построение различных структур программных агрегатов и передачи управления в них, обработать информационные векторы сложных типов и т.д.

К универсальным относятся макроопределения, с помощью которых обеспечивается преобразование типов данных и их организации (например, прямое и обратное транспонирование массивов и т. п.).

В БЗМ находятся загрузочные модули, которые специально разработаны как часто используемые подпрограммы для некоторых макроопределений библиотеки БМО. К ним относятся, например, модули вычисления адреса массива с размерностью более 2, модули проверки необходимости установления среды или освобождения динамической памяти и т. д. Обращение к модулям БЗМ формируется соответствующим макроопределением из БМО.

БМИ в данный момент обеспечивает объединение модулей, записанных в языках Ассемблера, Фортран и ПЛ/1. Для обозначения этих языков приняты следующие сокращения: AS —для языка Ассемблера, FT—для языка Фортран, PL — для языка ПЛ/1.

Эти символы послужили основой и для образования имен макроопределений БМИ. Так, ASPL используется для обозначения макроопределения связи по управлению модуля в языке Ассемблер с модулем в языке ПЛ/1, а знаки 0, 1, 2... следующие за парой этих символов, используются для обозначения макроопределении связи по данным. Каждое макроопределение начинается со слова MACRO и заканчивается MEND.

Описываемые ниже макроопределения библиотеки БМИ дают возможность пользователю самостоятельно создать модули-связки, в которые включается требуемая в

каждом конкретном случае последовательность макровызовов для обеспечения связей по управлению и по данным между разноязыковыми модулями. Каждое макроопределение содержит описание выполняемых им функций и формата макровызова. Тексты на языке Ассемблер для всех макроопределений приведены в приложении 1, а для загрузочных модулей — в приложении 2, которые опубликованы в первом издании книги [46].

3.1. Формирование имени модуля-связи

Имя модуля-связки формируется на основе имени вызываемого модуля и языка программирования, на которых написаны вызывающий и вызываемый модули. Формат имени:

<первые 5 символов из имени вызываемого модуля>#<L_1> <L_2>

Если длина имени вызываемого модуля 5 символов, то за основу берется полное имя. При этом L_1 — язык программирования, на котором записан вызывающий модуль, L_2 — язык программирования, котором записан вызываемый модуль. L_1 и L_2 могут принимать следующие знамения: A - для языка Ассемблера, F — для Фортрана, P – ПЛ/1/.

Например, если вызывающий модуль записан на языке ПЛ/1, вызываемый — на языке Ассемблера и имеет имя MOD1, то формат имени модуля-связки будет таким:

MODl # PA CSECT

Замечание. Исходные имена модулей пользователя не должны содержать символ #.

3.2. Макроопределения управления

Макроопределения управления обеспечивают связь по управлению одних исходных модулей с другими. Имена этих макроопределений образованы от соответствующих имен модулей ОС ЕС, выполняющих аналогичные функции. При этом последней буквой в образованном имени является буква P как символ принадлежности макроопределения к системе АПРОП.

К макроопределениям управления относятся:

- начало модуля-связки (SAVEP);
- формирование среды для модулей в языке Фортран (ASFT);
- формирование среды для модулей в языке ПЛ/1 (ASPL);
- вызов модулей из программ простой структуры (CALLP);
- динамический вызов модулей без образования подзадач (LINKP);
- возврат на вызывающий модуль (RETURNP);
- освобождение динамической памяти, выделенной модулем-связкой (FREEMNP).

3.2.1. Макроопределение начала модуля-связки

Макроопределение начала модуля-связки в модуле-связке записывается (после CSECT) первым.

В его функции входят:

- сохранение регистров вызывающего модуля;
- установка базового (9) регистра;
- выделение памяти под новую область сохранения регистров;
- подготовка старого и нового списков параметров;
- выделение памяти под новый список параметров;
- подготовка списка адресов нолей данных и выделение для него памяти;

- вызов модуля, который контролирует глубину вызовов модулей и установление среды ПЛ/1 (IGWTSB).

Понятие списка адресов полей данных дано ниже при описании макроопределения преобразования данных (см. п. 3.3).

Формат макровызова для этого макроопределения имеет вид:

SAVEP N

где N — число параметров в вызывающем модуле.

3.2.2. Формирование среды для модулей в языке Фортран

Макроопределение ASFT используется в том случае, когда из вызывающего модуля имеется обращение к модулю, записанном на языке Фортран.

Если модуль на языке ПЛ/1 или Ассемблер должен вызвать модуль на языке Фортран, то предварительно осуществляется вызов из библиотеки Фортрана программы IBCOM#, которая инициализирует код возврата и обеспечивает подготовку программ обработки прерываний. В задачу описываемого макроопределения входит подготовка для обращения к программе IBCOM#, т. е. это макроопределение выполняет следующие действия: занесение адреса IBCOM# в регистр 15 и передачу управления модулю IBCOM #.

Формат макровызова для данного макроопределения имеет вид: ASFT.

3.2.3. Формирование среды для модулей в языке ПЛ/1

Макроопределение ASPL обеспечивает вызов модулей в язык ПЛ/1 из других модулей. При этом предварительно моделируется среда ПЛ/1 (понятие среды см. в п.1.2.). На моделирование среды существенное влияние оказывает наличие в вызываемом модуле опции OPTIONS (MAIN). Если такая опция имеется и, кроме того, встречается впервые, то при вызове модуля в ПЛ/1 управление должно быть передано в его среду. В случае отсутствия такой опции управление должно быть передано на начало вызываемого модуля.

Так как среда ПЛ/1 замкнута (т. е. имеет одни вход и один выход), то в случае, когда цепочка модулей содержит несколько модулей, записанных на языке ПЛ/1, возникает необходимость контролировать наличие среды. Эту функцию выполняют модули IGWTSA, IGWTSB (описание см. в п. 3.4.4).

В функции макроопределения ASPL входит обеспечение моделирования среды, которое достигается выполнением следующие действий:

- анализ необходимости установления среды ПЛ/1; моделирование секции IHENTRY;
- анализ режима работы для вызова модуля IHESAP или IHETSA;
- моделирование секции IHEMAIN.

Макровызов к макроопределению ASPL должен содержаться в каждом модуле-связке, вызывающем модуль, написанный на языке ПЛ/1. Это дает возможность задать один раз среду, а затем обращаться к модулям на языке ПЛ/1, заданным как с опцией OPTION (MAIN), так и без нее.

Формат макровызова данного макроопределения имеет вид:

ASPL R, E, <имя первого выполняемого модуля>

Здесь R — задает режим, который при R = 0 является режимом без подзадачи и при R= 1 – с подзадачами:

E – идентифицирует точки входа для режима без подзадач IHEESAPB (E = 0) и IHESAPD (E=1).

В качестве примера рассмотрим макровызов ASPL 0, 1, X. При обращении на макроопределение будет сформирована среда для модуля X без режима подзадач (с точкой входа IHESAPD).

3.2.4. Вызов модулей из программ простой структуры

Макроопределение CALLP предназначено для организации передачи управления на модули, расположенные в программах простой структуры. Эту функцию данное макроопределение выполняет следующими действиями:

- засылка адреса нового списка параметров в регистр 1;
- засылка единицы в старший бит последнего слова нового списка параметров для обозначения конца этого списка;
- загрузка адреса вызываемого модуля в регистр 15;
- передача управления вызываемому модулю.

Формат макровызова к макроопределению CFLLP имеет следующий вид:

CALLP <имя вызываемого модуля>

3.2.5. Динамический вызов модуля без образования подзадачи

Макроопределение LINKP обеспечивает динамический вызов модуля без образования подзадачи. Эти функции выполняются следующей последовательностью действий:

- загрузка адреса нового списка параметров в регистр 1;
- засылка единицы в старший бит последнего слова нового списка параметров;
- обращение к программе супервизора для занесения вызываемого модуля в оперативную память и передачи ему управления.

Для обращения к макроопределению LINKP следует записать макровызов вида: LINKP <имя вызываемого модуля>

3.2.6. Возврат на вызывающий модуль

Макроопределение RETURNP по своим функциям соответствует макроопределению ОС ЕС, вызываемому по макрокоманде RETURN. Оно предназначено для восстановления содержимого регистров вызывающего модуля и передачи ему управления. Если вызываемый модуль является программой-функцией, то содержимое Регистра 0 для вызывающего модуля не восстанавливается, поскольку в этом регистре содержится результат выполнения функции.

Алгоритм данного макроопределения состоит в выполнении следующих действий, обеспечивающих:

- обращение к модулю IGWTSB для уменьшения счетчика глубины вызовов модулей на единицу;
- восстановление содержимого регистров 13, 14, 15;
- восстановление содержимого регистров 0—12, если вызываем, модуль не является функцией;
- возврат управления.

Формат макровызова имеет вид:

RETUR NP K

Здесь К=1, если вызывающий модуль является функцией и К=0 - в противном случае.

3.2.7. Освобождение динамической памяти, выделенной модулем-связки

В некоторых случаях (как, например, при создании ИВ массива символьных строк переменной длины) модуль-связка выделяет динамический участок памяти. Для

освобождения таких участков памяти необходимо использовать макроопределение FREEMNP. Это макроопределение вызывается только один раз для конкретного модуля-связки, и обращение к нему записывается перед макровызовом макроопределения RETURNP.

Макроопределение FREEMNP выполняет следующие функции: формирует обращение к модулю IGWFP (описание см. в п. 3.4.3), который освобождает динамические участки памяти; вызывает модуль IGWFP.

Формат макровызова на данное макроопределение имеет вид: FREEMNP.

3.3. Макроопределения преобразования

Макроопределения преобразования разработаны в соответствии с указанными ранее проблемами, связанными с отличиями в выходном коде компиляторов с разных языков программирования и отличиями языков программирования в средствах представления типов и структур данных.

Рассматриваемые макроопределения предназначены:

1) для изменения адресов, задаваемых идентично разными компиляторами для таких типов данных, как массивы, символьные строки и т. п.;

2) для необходимого преобразования типов данных, переданных в качестве параметров при обращении на вызываемый модуль.

Рассмотрим более подробно эти задачи.

1) При объединении модулей, написанных на языке Фортран с модулями, написанными на других языках, возникает необходимость изменить расположение элементов массивов. Это вызвано тем, что компилятор с языка Фортран располагает массивы в оперативной памяти таким образом, что при обращении к последовательным элементам первым изменяется самый левый индекс, а компиляторы с других языков располагают массивы так, что первым меняется самый правый индекс. Для нормальной работы связываемых модулей необходимо изменить расположение элементов массивов; для двумерных массивов это преобразование будем называть транспонированием. Поскольку действия по такому преобладанию выполняет пользователь, то в БМО помещены макроопределения преобразования транспонирования TRAN (см. п. 3.3.6), которые значительно сокращают объем работ пользователя.

2) Необходимость преобразования полей данных из одного вида представления в другой возникает тогда, когда параметры в выдающем и вызываемом модулях отличаются по типу данных.

Обработка параметров происходит в два этапа: сначала происходит преобразование информационных векторов, на втором этапе - обработка внутреннего представления данных. При обработке ИВ и заполнении нового списка параметров происходит вычисление фактических адресов передаваемых величин, которые заносят в список адресов полей данных. Этот список используется на втором этапе, а также при обратном преобразовании перед возвратом явления в вызывающий модуль.

Поскольку компиляторы с языков Фортран и Ассемблер обеспечивают одинаковое внутреннее представление для используемых в них данных, а компилятор с ПЛ/1 — для всех типов данных (кроме простых переменных) формирует ИВ (см. 1.2.2), то любая комбинация модулей с модулями на ПЛ/1 предполагает либо создание, либо по имеющемуся ИВ формирование адресов данных.

Макроопределения преобразования имеют следующие символьные имена:

ASPLNM —для создания ИВ по адресам данных;

PLASNM — по ИВ формирование адреса данного.

Здесь M обозначает размерность, а N — признак типа данных, которые распределены следующим образом:

0 – для простых переменных;

1 – для арифметических массивов;

2 – для символьных и битовых строк постоянной н переменной длины;

3 – для массивов символьных и битовых строк постоянной длины;

4 – для массивов символьных и битовых строк переменной длины и битовых строк переменной;

5 – для структур и массивов структур.

Каждое макроопределение адрес очередного элемента старого списка параметров размещает на регистре 2, адрес очередного элемента повою списка параметров — на регистре 3; адрес очередного элемента списка полей данных, куда в результате работы данного МО помещается адрес поля соответствующего параметра, — па регистре 5.

При выходе из макроопределения преобразования в регистры 2, 3 и 5 заносятся адреса следующих элементов соответствующих списков. Таким образом, каждое макроопределение подготавливает необходимую информацию для работы последующего.

Первоначальное формирование содержимого регистров 2, 3 и 5 осуществляется с помощью раннее описанного SAVEP.

3.3.1. Преобразование адресов простых переменных

Как уже было сказано выше, внутреннее представление простых переменных для всех языков программирования одинаково, отличаются они только длиной и способом описания в языке (см. п. 1.2). Поэтому в функции рассматриваемых макроопределений преобразования адресов простых переменных входит перепись адреса соответствующей переменной из старого списка параметров в новых: и в список адресов полей данных. Конкретно макроопределение PLAS0 предназначено для связи ПЛ/1 — Ассемблер (Фортран), а ASPL0 — соответственно для Ассемблер (Фортран) — ПЛ/1.

Оба макроопределения выполняют аналогичные действия, которые состоят в следующем:

- занесение адреса параметра в регистр 6;
- запись адреса в новый список параметров;
- запись адреса в список адресов полей данных;
- формирование содержимого регистров 2, 3, 5 для последующего их использования.

Обращение к этим макроопределениям имеет соответственно следующий вид:

```
ASPL0
PLAS0
```

3.3.2. Преобразование адресов арифметических массивов

При передаче арифметического массива в качестве параметра и модуля, написанного на языке ПЛ/1, в модули па языках Фортран и Ассемблер требуется с помощью ИВ, созданного компилятором ПЛ/1 (см. структуру ИВ в п. 1.2), вычислить фактический адрес первого элемента массива (элементы массива могут быть целый, с плавающей точкой и комплексные). С этой целью разработаны макроопределения PLAS1, PLAS11, PLAS12, каждое из которых предназначено соответственно для вычисления фактического адреса 3 (и более) -мерного, 2-мерного и 1-мерного массивов. Специальное выделение макроопределений для одной и двух размерности вызвано тем, чтобы сэкономить занимаемую ими память.

Действия, обратные этим макроопределениям, выполняют макроопределения ASPL11 и ASPL1. С их помощью формируется ИР для одномерных и n-мерных массивов соответственно.

Формирование по ИВ адреса массива.

Макроопределение PLAS11. Это макроопределение выбирает из ИВ величину для элемента массива для передачи его модулю на языке Фортран.

Данное макроопределение осуществляет:

- загрузку адреса информационного вектора в регистр 6; вычисление K_1*HP_1;
- получение виртуального адреса;
- внесение адреса массива в новый список параметров и список адресов полей данных;
- формирование содержимого регистров 2, 3, 5 для дальнейшего использования.

Для обращения на это макроопределение требуется указать его имя, а адрес массива, требуемый для обработки, берется из поля данных, сформированного макровызовом SAVEP.

Макроопределение PLAS12. Данное макроопределение осуществляет:

- загрузку адреса информационного вектора в регистр 6;
- вычисление величины K_2*HP_2;
- вычисление K_1*HP_1;
- сложение полученных результатов;
- получение виртуального адреса;
- формирование фактического адреса массива в новый список параметров в список адресов полей данных;
- формирование содержимого регистров 2, 3, 5 для дальнейшего использования.

Макроопределение PLAS12 используется при формировании адреса двумерного массива по информационному вектору, находящемуся в поле данных и содержащему коэффициенты K_1 и K_2.

Формат макровызова этого макроопределения: PLAS12. Это макроопределение решает общую задачу получения адреса массива n-мерной размерности.

Для получения этого адреса выполняется следующее:

- определяется размерность массива;
- если массив одномерный или двумерный, то вызывается соответственно макроопределение PLAS11 или PLAS12;
- формируется вызов модуля IGWV3, который вычисляет виртуальный адрес для массивов любой размерности; осуществляется вызов модуля IGWV3;
- адрес массива заносится в новый список параметров и адресов данных;
- формируется содержимое регистров 2, 3, 5.

При определении размерности макроопределение имеет выход на макроопределении PLAS11, PLAS12, в том случае, когда размерность массива равна 1 или 2.

Формат макровызова макроопределения PLAS1:

PLAS1 S

В нем S — задает информацию о размерностях массива в том виде, в котором она присутствует в описании вызывающего модуля на языке ПЛ/1. Например, для массива A (10, 20, 30) макровызов макроопределения PLAS1 имеет вид:

PLAS1 (10, 20, 30)

Информация о размерностях массива может иметь явное указание нижней границы. Например, для массива B (0: 10, 2:5, 30) макровызов запишется:

PLAS1 (0: 10.2:5,30)

Макрогенератор Ассемблера, используемый при формировании макрорасширения, определяет число элементов списка по запятым.

Создание ИВ по адресу массива. Для обеспечения передачи массивов из модулей на Фортране или Ассемблере в модули ПЛ/1 разработаны макроопределения ASPL11 и ASPL1, в зад., которых входит создание ИВ для арифметического массива соответственно с одной или с любой размерностью.

Алгоритм работы этих макроопределений основан на метод и построения ИВ, заложенной в компиляторе ПЛ/1. Для работы рассматриваемыми макроопределениями от пользователя не требуются знания структуры информационных массивов (символы!' битового, арифметического) и особенностей передачи типов данных из модулей, написанных на языке Фортран, на модули, написанных на языке ПЛ/1 (см. п. 1.2).

Ниже описываются макроопределения ASPL11, ASPL1.

Макроопределение ASPL11. Данное макроопределение обеспечивает построение ИВ для одномерного массива, выполняя последовательность следующих шагов:

- занесение адреса ИВ в регистр 7 и выделение памяти под ИВ;
- загрузку адреса передаваемого массива в регистр 6 и списки адресов полей данных;
- вычисление виртуального адреса и занесение его в первое слово ИВ;
- занесение адреса ИВ в новый список параметров;
- формирование содержимого регистров 2, 3, 5 для дальнейшего использования.

Формат макровызова для макроопределения ASPL11 имеет вид:

ASPL11 N, L

В нем N —верхняя граница размерности массива (нижняя граница массива принимается равной 1), а L — длина одного элемента массива в байтах.

Макроопределение ASPL1. Это макроопределение предназначено для построения ИВ массивов, число размерностей которых больше единицы. При своей работе данное макроопределение обеспечивает макровызов макроопределения ASPL11 в том случае, когда задан одномерный массив.

Алгоритм создания ИВ состоит в следующем:

- определение размерности массива и, если она равна единице, вызов ASPL11, иначе выполняется следующий шаг;
- занесение адреса ИВ в регистр 1;
- вычисление коэффициентов для ИВ;
- формирование виртуального адреса и загрузка адреса массива в регистр 6;
- занесение адреса массива в список адресов полей данных, а виртуального адреса в ИВ;
- занесение адреса ИВ в новый список параметров; формирование содержимого регистров 2, 3, 5.

Формат макровызова макроопределения ASPL1;

ASPL1 N, L

Здесь N — информация о размерностях массива; L — длина одного элемента массива в байтах.

3.3.3. Преобразование символьных строк

Если в модулях, написанных на языке Фортран и Ассемблер, используются массивы, содержащие символьные величины, то для передачи их как параметров используются макроопределения преобразования адресов символьных строк и массивов символьных строк.

Понятие «символьная строка» относится к модулям, написанным на языке ПЛ/1, и отсутствует в языке Фортран. При этом не исключается использование массива, содержащего в качестве элементов символьные величины. Для символьных строк компилятор с ПЛ /1 строит ИВ (см. п. 1.2.2). В языке Фортран в ряде простых случаев (пересылка величин, вывод на печать) можно использовать строки, описанные как простые переменные или арифметические массивы. Для обеспечения доступа к таким величинам аналогично арифметическим массивам разработаны макроопределения, обеспечивающие по ИВ вычисление адреса символьных строк, и наоборот.

Формирование на основе ИВ адреса строки. Данное макроопределение выбирает из ИВ строки фактический адрес поля и помещает его в новый список параметров и в список адресов полей данных.

Алгоритм макроопределения заключается в следующем:

- занесение адреса ИВ в регистр 6;
- занесение адреса поля строки в регистр 6 и в новый список параметров, а также в список адресов полей данных;
- формирование содержимого регистров 2, 3, 5 для дальнейшего использования.

Если поле строки не было выравнено, а в вызываемом модуле элементы в описании массива расположены по границе полного слова или полуслова, то при выполнении возможны ошибки.

Формат макровызова данного макроопределения:

PLAS2

Также как и при макровызове PLAS1 адрес поля символьных строк берется из поля данных, сформированного макровызовом SAVEP.

Создание ИВ строки для поля памяти. Данное макроопределение обеспечивает создание ИВ для поля памяти, которое в вызываемом модуле описано, как символьная строка.

Алгоритм заключается в следующем:

- занесение адреса ИВ в регистр 6 и формирование ИВ (структура ИВ описана в п. 1.2.2);
- занесение адреса ИВ в новый список параметров;
- занесение адреса поля строки в ИВ и в список адресов полей данных;
- формирование содержимого регистров 2, 3, 5.

Формат макровызова данного макроопределения:

ASPL2 BL, HL

В нем BL означает максимальную длину строки, а HL — текущую длину строки.

Данное макроопределение разработано для случая, когда величины BL и HL равны.

Замечание. В случае битовых строк могут использоваться макроопределения PLAS2 и ASPL2. При этом поля BL, HL должны содержать длину в битах. Память под битовую строку следует выделить кратно одному байту. Так для строки длиной 9 бит выделятся два байта. Эквивалентный тип данных для битовой строки можно определить на основании длины поля, заданного в байтах.

3.3.4. Преобразование адресов массивов символьных и битовых строк

Массивы с постоянной длиной элементов. ИВ для массивов символьных и битовых строк имеют структуру, одинаковую со структурой ИВ арифметических массивов. Отличие в том, что последним элементом ИВ являются два поля, в которых находится число баи: отведенных под один элемент массива.

Например, для описания массива символьных строк на языке ПЛ/1:

DCLX(10,20) CHAR (5);

ИВ имеет вид:

V.A.	
100	
5	
10	1
20	1
5	5

Все величины, участвующие в ИВ для массива символьных строк, вычисляются аналогично ИВ для арифметического массива. Следовательно, преобразование адресов массивов символьных строк постоянной длины может осуществляться с помощью макроопределений преобразования адресов арифметических массивов. Для удобства работы пользователя с такими типами данных при составлении модулей-связок выделены следующие макроопределения: PLAS3, PLAS31, PIAS32, ASPL3, ASPL31. Первые три отличаются от соответствующих макроопределений преобразования адресов арифметических массивов только названием. Последние два, кроме названия отличаются еще и тем, что при построении ИВ добавляются поля, содержащие длину одного элемента массива. На содержательном уровне и функционально эти модули совпадают.

Следовательно, макроопределения PLAS3, PLAS3I, PLAS32 предназначены для формирования по заданному ИВ адресов массивов символьных и битовых строк для n-мерных, одно- и двумерных массивов, а макроопределения ASPL3 и ASPL31 - для обратного преобразования n-мерных и одномерных массивов символьных и битовых строк.

Макровызов всех описанных макроопределений аналогичен макровызову соответствующих макроопределений преобразования адресов арифметических массивов.

Все элементы массива символьных строк постоянной длины следуют друг за другом и представляют в памяти единый участок. Поэтому данным массивам в модулях, написанных на языках Фортран и Ассемблер, можно поставить в соответствие обыкновенные массивы. Так, описанию на языке ПЛ/1:

```
DCL X(10) CHAR(8);
```

можно поставить в соответствие описание на языке Фортран: INTEGER* 4 Y(20) либо * INTEGER* 2 Z(40)

Описание необходимо выбирать так, чтобы общие длины соответствующих массивов совпадали, т. е. для предыдущего случая можно использовать в языке Ассемблера одно из таких описаний:

```
        DS CL80
Или     DS 10CL8
        DSXL80
```

Информационный вектор у массивов битовых строк постоянной длины и у массивов символьных строк постоянной длины имеет одинаковую структуру. В обоих случаях элементы массива следуют в памяти один за другим, но в случае битовых строк каждый элемент дополняется до целого байта. Так, для массива B, описанного в виде:

```
DCL B(8) BIT(9)
```

отведено не 9 байт (8x9), а 16 байт (каждый элемент занимает два байта).

Массивы символьных и битовых строк переменной длины. Для массивов символьных строк переменной длины имеют два уровня. Первый уровень задает ИВ массива строк постоянной длины, при этом в поле текущей длины содержится нуль и признак типа массива. Если указанное поле отлично от нуля, то массив — постоянной длины. В случае равенства текущей длины нулю, массив содержит строки переменной длины. Вторым уровнем являются ИВ для элементов символьных или битовых строк. В качестве примера рассмотрим следующий фрагмент программы:

```
DCL X(2,2)          CHAR (6) VAR;
X(1,1) = ´MOD´;     X(1,2) = ´SEG´;
X(2,1) = ´PROG´;    X(2,2) = ´COMPX´;
```

Информационный вектор будет иметь следующую структуру:

VA.	
16	
	8
2	1
2	1
8	0

Адрес элемента x(1,1)	
6	3
Адрес элемента x(1,2)	
6	3
Адрес элемента x(2,1)	
6	4
Адрес элемента x(2,2)	
6	5

M	O	D			
S	E	G			
P	R	O	G		
C	O	M	P	X	

Исходя из приведенной структуры все массивы символьных или битовых строк переменной длины имеют максимальную длину поля в ИВ, равную 6.

Для формирования адресов массивов строк переменной длины по ИВ разработаны макроопределения PLAS4I, PLAS4 н ASPLI.

Макроопределение PLAS41. Данное макроопределение определяет адрес одномерного массива строк по его ИВ. Алгоритм этого макроопределения состоит из следующих последовательно выполняемых шагов:

- загрузка адреса ИВ массива символьных строк на регистр 6;
- вычисление адреса ИВ строки для первого элемента массива и занесение его в новый список параметров адресов полей данных;
- формирование содержимого регистров 2, 3, 5.

Память под массив строк переменной длины выделяется исходя из максимальной длины каждого элемента, которые расположен друг за другом, поэтому доступ к полю массива производится через адрес первого элемента. Так как вызываемому модулю доступно все поле массива строк переменной длины, то в первом варианте макроопределении принято ограничение, состоящее в том, что длины одного элемента массива в вызывающем и вызываемом модулях должны совпадать. Это ограничение принято в связи с тем, что при разных длинах элементов массива для нормального выполнения вызываемого модуля потребовалось бы переписывать массив и изменять длину каждого его элемента. Учитывая, что при этом потребовалась бы и дополнительная память, данное макроопределение PLAS41, а также PLAS4 формирует адреса массивов строк переменной длины вызываемому модулю без изменения их содержимого.

Макроопределение PLAS4. Данное макроопределение вычисляет адрес массива строк переменной длины любой размерности. По своим функциям данное макроопределение совпадает с макроопределением PI.AS3 и в случае одномерного массива строк обращается на PI.AS41. Если число размерностей больше 1, то выполняется следующее:

- формируется команда вызова модуля IGWV3 (см. п. 3.4.1); обеспечивается вызов модуля IGWV3;
- с помощью IGWV3 определяется фактический адрес ИВ первого элемента массива, на основе которого вычисляется адрес первого элемента массива и занесение его в новый список параметров и список адресов полей данных;
- формируется содержимое регистров 2, 3, 5.

Результатом работы макроопределения является получение адреса начала массива и его размещение в новом списке параметров и адресов полей данных.

Формат макровызова макроопределения PLAS4:

PLAS4 M

Здесь M — информация о размерностях массива.

В качестве примера рассмотрим описание массива DCL X(2,2) CHAR(6)VAR. Для этого описания адрес массива будет вычислен с помощью макровызова PLAS4(2,2).

Макроопределение ASPL4. Данное макроопределение разработано для получения ИВ массива строк переменной длины и любой размерности, при этом максимальная длина всех элементов массива должна быть одинаковой. Макроопределение ASPL4 выполняет следующие функции:

- определяет число размерностей массива;
- формирует обращение к модулю IGWV4 (описание см. в п. 3.4.2), который формирует ИВ массива строк переменной длины;
- вызывает модуль IGWV4;
- обеспечивает занесение в новый список параметров признака о необходимости освобождения динамического участка памяти, выделенного модулем IGWV4;
- формирует регистры 2, 3, 5 для дальнейшего использования,

Формат макровызова макроопределения ASPL4:

ASPL4 S, L

Здесь S —список верхних границ размерностей (нижние границы принимаются равными единице), L— длина элемента массива.

Пример записи макровызова:

ASPL4 (3,10,20),6

для описания массива DCL X(3. 10, 20) CHAR(6)VAR; ,длина элементов которого равна шести символам.

Обработка структур. Структуры, содержащиеся в языке ПЛ/1 задаются конкатенацией ИВ и адресных констант своих элемент. Если элемент является простой переменной, то ИВ структуры содержит его адрес. Для сложных типов данных ИВ содержит соответствующие ИВ отдельных элементов.

Пусть, например, имеется описание структуры в языке ПЛ/1:

```
DCL     1 A,
        2 B (10),
        2 C (5) CHAR(4),
        2 D,
        3 E BIN FIXED(31),
        3 F CHAR (80);
```

Для нее компилятор ПЛ/1 сформирует ИВ следующего вида:

V.A. (B)	
4	
10	1
V.A. (C)	
4	
5	1
4	4
Адрес поля E	
Адрес поля F	
80	80

При обработке ИВ структуры будут использоваться те же макроопределения преобразования адресов, которые были описаны выше. Каждый элемент структуры будет обработан соответствующим макроопределением. Для данного примера необходимо записать следующую последовательность макровызовов:

PLAS1 (10)
PLAS3 (5)
PLAS0
PLAS2

Формирование этих макровызовов выполняет макроопределение PLAS5.

После выполнения этих макровызовов в новом списке параметров будут находиться четыре адреса для соответствующих полей. Поэтому при передаче структуры через параметры (вызываемый модуль написан на языке Ассемблера и на Фортране) в вызываемом модуле должны быть предусмотрены переменные и массивы для всех элементов структуры. Если данная структура передается модулю, написанному на Фортране, то среди параметров вызываемого модуля должны находиться три массива (A, L, K) и одна прочая переменная X. Им соответствует, например, следующее описание:

SUBROUTINE FI (…A.L.X.K…)
DIMENSION A (I0), L(5), K(20)

В модуле-связке для параметров типа структура должна быть выделена требуемая память для построения нового списка параметров с учетом всех элементов структур и проведена перепись адресов параметров и адресов составляющих структур в новый список параметров с помощью описываемого ниже макроопределения PLAS5.

Макроопределение PLAS5. Данное макроопределение переписывает адреса простых переменных и ИВ, являющихся элементами ЦВ структуры, н новый список параметров. Ему соответствует следующий алгоритм.

Подготавливаются регистры 8 и 2 для обработки структуры, производится проверка на тип данных и в зависимости от этого осуществляется соответствующая обработка.

Тип данных = 0. Занесение в новый список параметров адреса переменной.

Тип данных = 1. Занесение в новый список параметров адреса ИВ массива.

Тип данных = 2. Занесение в новый список параметров адреса ИВ строки.

Тип данных = 3. Занесение в новый список параметров адреса МВ массива строк постоянной длины.

Тип данных = 4. Занесение в новый список параметров адреса ИВ массива строк переменной длины.

Содержимое регистра 2 восстанавливается тля дальнейшего использования в модуле-связке. Обработка данных производится с помощью макроопределений PLAS0, PLASI, PLAS2, PLAS3, PLAS4. Модуль-связка должен столько содержать вызовов макроопределения PI.AS5 для каждой структуры, сколько фактических полей она содержит.

Формат макровызова рассматриваемого макроопределения:

PLAS5 S,T,X,P

Здесь T — тип данных, принимающий значения 0,1,2,3,4 (см. п. 3.3):

X — информация о размерностях для массивов;

P — длина одного элемента передаваемых данных;

S — признак операций, выполняемых данным макроопределением (их может быть три), каждой из которых соответствует: подготовка регистров 2 и 3; обработка очередного элемента структуры; восстановление регистра 2.

Для каждой передаваемой структуры первое макроопределение из группы макроопределений PLAS5 должно выполнить операцию 1, а последнее — операцию 3. Поэтому S может принимать следующие значения (в скобках указана последовательность операции):

S = 2 (2);
S = 3(1 + 2);
S=6(2+4);
S = 7(1 +2 + 4).

Для структуры, описанной на с. 92, последовательность макроопределении должна быть следующей:

```
PLAS5        3,1, (10),4
PLAS5        2,3, (5),4
PLAS5        2,0,(1),4
PLAS5        6,2,(1),80
```

3.3.5. Изменение порядка параметров при обращении к вызываемому модулю

Если порядок следования параметров в вызываемом и вызывающем модулях различен, то для его изменения разработано макроопределение SCMOVE. Необходимость изменения порядка параметров поясним на следующем примере.

Пусть даны два модуля FI и F2 на языке Фортран.

```
SUBROUTINE      F1
DIMENSION       A(10)
…
CALL            F2(A,B,C)
RETURN
END
SUBROUTINE      F2(X,Y,Z)
DIMENSION       Y(10)
…
RETURN
END
```

В вызывающем модуле заданы массив А и простые переменные В и С, а в вызываемом модуле массив Y является вторым параметром. Для нормального выполнения модуля F2 необходимо изменить порядок параметров: А сделать вторым, а В — первым.
За тем при возврате в модуль F1 должен быть восстановлен порядок.

Макроопределение SCMOVE. Оно строит новый список параметров в соответствии с измененным порядком, затем этот список принимается за «старый» список параметров, а под новый отводится необходимая память.

Эти действия выполняются в следующем порядке:

- проверяется список параметров, и если он старый, то переписываются параметры из старого списка в новый, а если он новый, выполняется следующий шаг;
- изменяется порядок следования параметров, и если необходимо, выделяется память под новый список параметров;
- формируются старый и новый списки параметров.

Формат макровызова данного макроопределения:

SCMOVE L,N1,N2,S

Здесь L —указывает на перепись из старого списка параметров в новый (при L = 0 перепись не происходит. L должно равняться числу переписываемых параметров);

N1 и N2 — номера соответственно в списке передаваемых параметров) вызывающего модуля и в списке параметров вызываемого модуля;

S — информирует о необходимости построения нового списка параметров (при S=0 список не строится, иначе S должно соответствовать числу параметров в вызываемом модуле).

Для рассматриваемого в этом пункте примера модуль-связка должен содержать следующие макровызовы:

```
SCMOVE 3, 1, 2, 0
SCMOVE 0, 2, 1,3
```

3.3.6. Транспонирование матриц

Макроопределение TRAN предназначено для изменения расположения элементов массива при работе с модулями, написанными на языке Фортран. Массивы в модулях на Фортране расположены в оперативной памяти так, что при обращении к двум послед овальным элементам меняется первым самый левый индекс. В модуле на языке ПЛ/1 и Ассемблер массивы располагаются по строкам.

Макроопределение TRAN выполняет следующие функции:

- выборку необходимого адреса массива из списка полей данных;
- определение числа размерностей массива n;
- определение и формирование имени программы преобразования (т. е. IGWTn);
- подготовку списка параметров;
- вызов программы преобразования.

При передаче массивов модулям, написанным на языке Фортран, для каждого массива требуется дважды применить макроопределение TRAN: одни раз перед вызовом модуля и другой раз после него.

Макровызов имеет вид:

TRAN M,L,N,P=K

Здесь M — информация о размерностях массива;

L — длина одного элемента;

N — порядковый номер данного массива в строке передаваемых параметров;

P=l, если массив передается из модуля на Фортране и P=0, если массив передается в модуль на языке Фортран.

Макровызов макроопределения TRAN должен стоять после формирования списка полей данных, т. е. после макровызова макроопределений ASPL1, PLASI.

Пример 16.

Пусть в модуле на языке Фортран имеется оператор CALL

P1 (A, B, C), в котором передается массив REAL*4 C (10,10), и этот оператор вызывает модуль P1 на языке ПЛ/1.

Схематически модуль-связка для рассматриваемого примера имеет вид:

```
CSECT
  …
TRAN(10,1O),4,3,P = l
  …
CALLP P1
 …
TRAN (10,10),4,3,P = 0
…
END
```

Пример 17.

Пусть теперь модуль на языке ПЛ/1 вызывает модуль на язык Фортран (P2), и в нем используются те же передаваемые параметры. Структура модуля-связки в этом случае будет следующей:

```
CSECT
…
TRAN(10,1O),4,3,P = 0
…
CALLP P2
…
```

TRAN (10,10),4,3,P = 1
…
END

3.4. Библиотека загрузочных модулей

В связи с тем, что размер модуля-связки не должен превышать 4096 байт, так как в каждом модуле-связке устанавливается только один базовый регистр (9), то для выполнения некоторых функций преобразования, требующих для своей реализации больших числа команд, составлена библиотека загрузочных модулей. Каждый загрузочный модуль вызывается из соответствующего макрорасширения модуля-связки, в котором был помещен макровызов макроопределения, использующего готовые модули из библиотеки БЗМ.

В данном описании содержится описание модулей, вычисляющих адрес массива по его ИВ для случая, когда число размерностей больше 3, а также модулей, выполняющих транспонирование. При ручном составлении модулей-связок пользователь имеет дело с модулями БЗМ, поскольку они являются вспомогательными, и обращаются на них макроопределения, описанные в п. 3.3. Но для общего представления модули БЗМ кратко описываются, а в приложении 2 в [46] приведена их распечатка.

3.4.1. Модуль вычисления адреса массива по ИВ

Этот модуль имеет символическое имя IGWV3, обращение к нему формируется в макроопределениях PLAS1, PLAS3, PLAS4, когда число размерностей превышает 2. Входным параметром модуля служит число размерностей массива, передаваемое в регистре

1) Адрес ИВ определяется текущим содержимым регистра

2) После выполнения модуля IOWV3 фактический адрес массива находится на регистре 1.

Данный модуль является реентерабельным.

3.4.2. Модули, используемые при транспонировании массивов

Эти модули должны вызываться в случае, когда среди данных предаваемых модулю, написанному на языке Фортран, содержатся массивы с числом размерностей больше или равно 2. Обращение на модули нормируется макроопределением TRAN, описанным в п. 3.3.7.

Учитывая сложность задачи транспонирования при большом числе размерностей и несовпадении границ размерностей массива, разработка модулей транспонирования проводилась следующим образом. Для каждого числа размерностей составлялся корневой модуль, который производит обращение к группе модулей, каждый из которых производит транспонирование в одном из конкретных случаев, зависящих от соотношения размерностей массива. Общая блок-схема загрузочных модулей транспонирования представлена на рис 23. Имя каждого корневого модуля $IGWT_n$, где n — чисто размерностей массива. Имена остальных модулей $IGWT_{nm}$, где m определяет конкретный случай транспонирования.

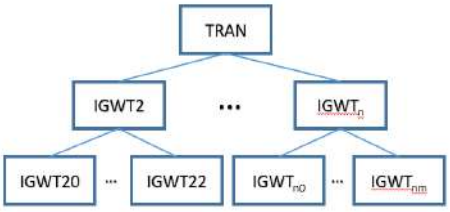


Рис. 23. Общая схема программы транспонирования (TRAN) массивов

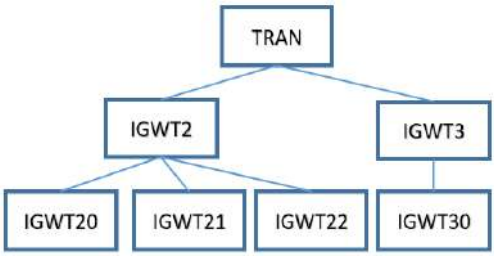


Рис. 24. Схема программы транспонирования для массивов 1-3 размерности

В данном описании рассматриваются загрузочные модули, используемые при транспонировании массивов с размерностью 2 и 3; блок-схема приведена на рис. 24.

Модули IGWT2 и IGWT3 являются управляющими и обеспечивают в соответствии с входными параметрами выбор модулей транспонирования. При этом модуль IGWT20 производит транспонирование двумерного массива A (n,m) при n>m, модуль IGWT21 — при n<m и модуль IGWT22 — при n=m.

Модуль IGWT30 производит транспонирование трехмерного массива A (n, m, k) при n = k и m — любом (в частности, n = m = k).

Параметром для всех модулей транспонирования служит поле, адрес которого передается в регистре 1. Структура поля следующая: T_n

Адрес массива, 4 байта	Буфер для обмена, 16 байт	Длина элемента, 2 байта	Число разномерностей, 2 байта	Размерность 1, 2 байта	…	Разномерность 3, 2 байта

Учитывая, что в модулях, написанных на языке Фортран, длина одного элемента не превышает четырех слов, под поле обмена (буфера) отводится 16 байт.

Все модули транспонирования — реентерабельные.

3.4.3. Модуль освобождения динамических участков памяти

Модуль IGWFP служит для освобождения динамических участков памяти, выделенных в модуле-связке. Обращение из него формируется макроопределением FREEMNP. На регистре 4 данному модулю передается адрес нового списка параметров, каждый элемент которого проверяется на наличие признака освобождения динамического участка. Если признак присутствует, то выделенная память для данного параметра освобождается.

3.4.4. Модули контроля установки среды ПЛ/1

К данным модулям относятся модули IGWTSA и IGWTSB. В их задачу входит проверка необходимости установки среды ПЛ/1. Основные функции выполняет модуль IGWTSA, а модуль IGWTSB служит для согласования передачи управления от модуля-связки к IGWTSA.

Принцип работы модуля IGWTSA состоит в следующем. Каждое из макроопределений ASPL, находящееся в модулях-связках обращается к модулю IGWTSA с целью проверки признака наличия среды ПЛ/1. Если соответствующий признак нулевой, то среда ПЛ/1 устанавливается, в противном случае необходимость установки игнорируется. Кроме макроопределения ASPL, к модулю IGWTSA обращаются макроопределения SAVEP и RETURNP.

При вызове модуля IGWTSA из макроопределения SAVEP происходит увеличение на единицу счетчика глубины цепочки модулей, выполняющихся в данный момент в среде ПЛ/1. Если среда не уставлена, то счетчик не меняется.

Макроопределение RETURNP формирует обращение для уменьшения описанного счетчика на единицу. Когда значение счетчик, становится равным нулю, то сбрасывается признак установки средь ПЛ/1, и цепочка модулей может выполняться повторно.

При входе в модуль IGWTSA регистр 1 содержит код запроса, по которому модуль обрабатывает соответствующий запрос модуля- связки и выполняет конкретные функции. Результатом работы является формирование признака о необходимости установки среды ПЛ/1.

3.4.5. Модуль формирования ИВ массива строк переменной длины

Для выделения динамического участка памяти под ИВ массива строк переменной длины и формирования этого ИВ предназначен модуль IGWV4, обращение к которому осуществляется из макроопределения ASPL4.

Параметрами для работы IGWV4 служит вектор характеристик размерностей массива, формируемый макроопределением ASPL4. По этому вектору формируется запрос на выделение динамического участка памяти. Затем по вектору характеристик размерностей в выделенном участке памяти формируются и вторичные ИВ для массива строк переменной длины. Результатом работы модуля является адрес поля следующей структуры:

Смещение	Назначение поля
-8	Адрес выделенного участка памяти
-2	Полуслово, содержащее длину выделенного участка памяти, в байтах
0	ИВ массива строк переменной длины

Глава 4

Использование средств БМИ для создания модулей-связок

В связи с тем, что у пользователя ЯП ОС ЕС возникает необходимость связывать модули, записанные, по крайней мере, в двух входных языках, то в данной главе даются рекомендации по применению макроопределений БМИ для самостоятельной разработки модулей-связок. Именно с этой точки зрения в гл. 3 описана соответствующие средства БМО.

При автоматизированном создании агрегатов средствами системы АПРОП в БМИ имеется значительный набор макроопределений и готовых подпрограмм в загрузочном виде, которые выполняют функции обеспечения общих наборов данных и областей управления подзадачами и т. д. Такого рода элементы БМИ связаны с системными таблицами и поэтому описание таких элементов, а также указанных таблиц для использования без системы АПРОП требует изложения большого объема материала, что предполагает и увеличен»' нагрузки на пользователя. В таких случаях целесообразнее использовать систему АПРОП.

БМИ будет развиваться в направлении реализации средств, описания ФДТ приведенных в разделе средств 1.4 -1.7.

4.1. Правила написания модулей-связок

Модуль-связка пишется для пары модулей: вызывающего и вызываемого. При составлении модуля-связки используются все прав» ла написания программ на языке Ассемблера ОС ЕС, а также формат макровызовов макроопределений БМИ, которые описан и в гл. 3. При этом имя модуля-связки должно составляться согласно принятому соглашению в данном описании.

Первым предложением модуля-связки должно быть:

<имя модуля-связки> CSECT,

а последним запись конца END.

Для сохранения содержимого регистров вызывающего модуля, установления базового регистра содержимого регистров вызывающего модуля и возврата **в** него управления используется макроопределение RETURNP.

Перечисленные выше четыре предложения являются обязательными при описании модуля-связки. Учитывая специфику построения модулей связок, макроопределения SAVEP и RETURNP выполняют ряд дополнительных функций. Так, SAVEP выделяет память под новый список адресных параметров и список, содержащий адрес полей параметров с последующей обработкой. Макроопределение RETURNP в зависимости от того, является вызываемый модуль подпрограммой или функцией, соответственно восстанавливает содержимое регистра 0 или оставляет без изменений.

Другие группы макроопределении могут присутствовать или опускаются по мере необходимости, но порядок их следования должен сохраняться.

4.1.1. Использование макроопределений преобразования адресов

Эти макроопределения обрабатывают ИВ в соответствии с описанием типов данных в вызывающем и вызываемом **модулях.**

Пример 18.

Пусть даны модули: F и P. Вызывающий:

```
SUBROUTINE F DIMENSION
A(10,101.B(-10)

CALL P#FP(A .C.B). RETURN

End P;
```

Вызываемый:

```
P:   PROC(X.Y.Z) OPTIONS(MAIN);
DCL X(I0.I0).Y.Z.(40);

END P;
```

Модуль- связка будет содержать следующие предложения:

```
P#FP   CSECT SAVEP 3 ASPL1 (I0.I0M
       ASPL0 ASPL I 10) 4
       RETURNP0 END
```

Как уже отмечалось выше, если между типами **связей** Ассемблер—ПЛ/**1** и Фортран —ПЛ/**1,** а также ПЛ/**1** — Ассемблер. ПЛ/**1** — Фортран принципиальных различий при передаче данных существует, то используются одни и те же макроопределения.

4.1.2. Использование макроопределений для передачи структур

Макроопределение PLAS5 применяется в случае, когда модулям на языках Фортран нлн Ассемблер передается структура в языке ПЛ/1. При этом вызываемый параметр должен содержать параметры, соответствующие каждому полю структуры. Следующий пример иллюстрирует использование макроопределения PLAS5.

Пример 19.

Пусть даны два модуля: вызывающий PL (на языке IIЛ/1 и вызываемый F (на языке Фортран). Вызывающий модуль:

```
PL:   PROC OPTIONS (MAIN). DCL IA. 2 B CHAR(80).
       2 C.
       3 D.
       3 E(I0);
       'CALL F#PF (Al END PL;
```

Вызываемый модуль:

```
SUBROUTINE F (x.y.z)
 DIMENSION X(20).Z(I0)
  RETURN END
```

В модуле F параметрам x, y, *z* соответствуют поля B, D, E. Модуль-связка будет содержать следующие предложения:

```
F#PF    CSECT
        SAVEP 3
        PLAS5 3.2.(1 (.80
        PLAS5 2,0.(1).4
        PLAS5 6.1.(10) .4
        RETURNP0
```

Необходимо отметить, что число принимаемых параметров в вызываемом модуле больше числа передаваемых параметров в вызывающем.

Память, отведенная макроопределением SAVEP, используете" для построения расширенного списка параметров.

Если порядок следования параметров в вызываемом и вызывай' щем модулях различается, то применяется макроопределение SCMOVE.

Пример 20.

Рассмотрим два модуля FI и F2, написанные на языке Фортран.

```
 SUBROUTINE FI DIMENSION A
 (10)
CALL F2#FF A. B (x,y., c)
RENURN
END


SUBROUTINE F2 (x.y.z) DIMENSION
y (10)
RETURN END
```

В вызывающем модуле A — массив, B и C — простые переменные. Для нормального выполнения модуля F2 необходимо изменить порядок параметров A сделать вторым, а B — первым.

Модуль-связка будет содержать следующие предложения:

```
F2#FF    CSECT
SAVEP 3 SCMOVE 3.1.2.0
SCMOVE 0.2.1.3

RETURN 0

END
```

4.1.3. Использование макроопределений преобразования данных и транспонирования массивов

При передаче массивов данных из Фортрана в ПЛ/1 и наоборот возникает необходимость произвести транспонирование массивов, поменять столбцы в строки, а затем, наоборот. С этой целью используется макроопределение TRAN.

Пример 21.

Пусть даны модули A1 и A2, записанные в языке ПЛ/1 и Фортран соответственно. В вызывающем модуле A1 содержится CALL A2#PF для выполнения действий по объединению A1 и A2.

Вызывающий:

```
A1:     PROC OPTIONS (MAIN);
        DCL K(4,10), M(5,5);
        DO J=1 TO 4; DO L=1 TO 10;
        K (J,L0 = (J-1)*10+L;
        END; END;
        DO J=1 TO 5;
        DO L=1 TO 5;
        M(J,L) = (J-1)*5 + L;
        END; END;
        CALL A2#PF(K,M);
```

```
            END A1;
```

Вызываемый:

```
            SUBROUTINE A2(K,M)
            INTEGER*2 K(4,10), M(5,5)
            WRITE (6,1) ((K(J,L)=1,10), J=1,4)
            WRITE (6,1)  ((M(J,L), L=1,5), J=1,5)
            RETURN
      1     FORMAT (4013)
      2     END
```

Модуль-связка A2#PF:

```
            A2#PF   CSECT
            SAVEP2
            PLAS1 (4,10)
            PLAS1 (5,5)
            TRAN (4,10),2,1,P=0
            TRAN (5,5),2,2,P=0
            ASFT
            CALL A2
            TRAN (4,10),2,1,P=1
            TRAN(5,5),2,2,P=1
            RETURNP 0
            END
```

Модуль A2#PF с макрорасширениями имеет вид:

```
      A2#PF CSECT
            SAVEP 2
            DS    OF
            STM   14,12,12(13)
            BALR  9,0
            USING *,9
            LR    2,1
            BAL   15,*+8
            DC    V(IGWTSB)
            L     15,0(15)
            LA    1,4
            BALR  14,15
            XR    1,1
            BALR  14,15
            CNOP  0,4
            ST    13,*+12
            BAL   1,*+76
            DC    18F'0'
            ST    1,8(13)
            LR    13,1
            CNOP  0,4
            BAL   4,*+20
            DC    H'2'
            DC    H'4'
            DC    F'0'
            DC    2F'0'
            BAL   5,*+12
            DC    2F'0'
```

```
ST      5,4(4)
LR      15,5
LA      3,8(4)
PLAS1   (4,10)
L       6,0(2)
L       7,8(6)
MH      7,18(6)
L       1,4(6)
MH      1,14(6)
AR      7,1
A       7,0(6)
ST      7,0(3)
ST      7,0(5)
LA      2,4(2)
LA      3,4(3)
LA      5,4(5)
PLAS1   (5,5)
L       6,0(2)
L       7,8(6)
MH      7,18(6)
L       1,4(6)
MH      1,14(6)
AR      7,1
A       7,0(6)
ST      7,0(3)
ST      7,0(5)
LA      2,4(2)
LA      3,4(3)
LA      5,4(5)
TRAN    (4,10),2,1,P=1
L       15,4(4)
CNOP    0,4
BAL     1,*+36
DC      V(IGWT2)
DC      A(0)
DC      4A(0)
DC      H'2'
DC      H'4'
DC      H'10'
MVC     4(4,1),0(15)
L       15,0(1)
LA      1,4(1)
BALR    14,15
TRAN    (5,5),2,2,P=1
L       15,4(4)
CNOP    0,4
BAL     1,*+36
DC      V(IGWT2)
DC      A(0)
DC      4A(0)
DC      H'2'
DC      H'2'
```

```
DC      H’5’
DC      H’5’
MVC     4(4,1),4(15)
L       15,0(1)
LA      1,4(1)
BALR    14,15
ASFT
CNOP    0,4
BAL     15,*+8
DC      V(1BCOM#)
L       15,0(15)
BAL     14,64(15)
CALLP   A2
LA      1,8(4)
LH      6,0(4)
LA      6,1(6)
SLL     6,2
AR      6,4
MV1     0(6),X’80’
CNOP    0,4
BAL     15,*+8
DC      V(A2)
L       15,0(15)
BALR    14,15
L       15,4(4)
TRAN    (4,10),2,1,P=1
L       15,4(4)
CNOP    0,4
BAL     1,*+36
DC      V(IGWT2)
DC      A(0)
DC      4A(0)
DC      H’2’
DC      H’2’
DC      H’10’
DC      H’4’
MVC     4(4,1),0(15)
L       15,0(1)
LA      1,4(1)
BALR    14,15
TRAN    (5,5),2,2,P=1
L       15,4(4)
CNOP    0,4
BAL     1,*+36
DC      V(IGWT2)
DC      A(0)
DC      4A(0)
DC      H’2’
DC      H’2’
DC      H’5’
DC      H’5’
MVC     4(4,1),4(15)
```

```
        L       15,0(1)
        LA      1,4(1)
        BALR    14,15
        RETURNP     0
        CNOP    0,4
        BAL     15,*+8
        DC      V(IGWTSB)
        L       15,0(15)
        LA      1,8
        BALR    14,15
        L       13,4(13)
        LM      14,15,12(13)
        LM      0,12,20(13)
        BR      14
        END
```

Пример 22.

Аналогично предыдущему примеру рассмотрим модуль-связку для случая, когда вызывающий модуль FT задан на языке Фортран, а вызываемый PL – на языке ПЛ/1.

Вызывающий:

```
        SUBROUTINE FT
        READ (5,L)K
        DO 2 L = 1,5
        CALL PL(K)
        FORMAT (14)
        RETURN
        END
```

Вызываемый:

```
PL:     PROC(K) OPTIONS(MAIN);
        DCL K FIXED BIN(31);
        PUT LIST(K);
        END PL
```

Модуль-связка PL#FP имеет вид:

```
PL#FP   CSECT
                SAVEP 01 ASPL0(I),4 IRAN
                (I).4,01,P= 1 ASPL 0.I.PL CALLP PL
                TRAN(I).4.01.I' = 0 RETURN 0
                END
```

Комплексный пример

В заключение приводится пример, охватывающий все перечисленные выше группы макроопределений.

Пример 23.

Вызывающий модуль GVN написан на языке ПЛ/1, вызываемый GVN1 – на языке Фортран.

```
GVN:    PROC OPTIONS (MAIN);
        DCL 1A,
        2B CHAR(12);
        2D,
        3E (10,10).
        3F;
        …
        CALL GVN1#PF(A);
        …
        END GVN
```

```
      SUBROUTINE GVN1(X,Y,Z);
      DIMENSION X(3), Y(10,10)
      …
      RETURN
      END
```

При ручном составлении модуля-связки исходный модуль FT должен быть

```
      END GVN
      SUBROUTINE GVN1 (X,Y,Z);
      DIMENSION X(3), Y(10,10)
      …
      RETURN
      END
```

Модуль-связка будет иметь вид:

```
GVN1#PF    CSECT
           SAVEP 3
           PLAS5 3,2,(1),12
           PLAS5 2,1,(10,10),4
           PLAS5 6,0,(1),4
           TRAN (10,10),4,2,P=0
           ASFT
           CALL GVN1
           TRAN (10,10),4,2,P=1
           RETURN 0
           END
```

При ручном составлении модуля-связки исходный модуль FT должен быть откорректирован посредством замены оператора вызова CALL PL(K) на оператор PL#FP.

4.1.4. Использование макроопределений установки среды

Среда языка Фортран устанавливается макроопределением **SFT.** Среда языка ПЛ/**1** устанавливается обращением на макро-пределснне ASPL.

Пример 24.

Пусть вызывающий модуль написан на языке Фортран, а вызы-мый — на языке ПЛ/**1** с именем PL, тогда в модуле-связке AS небходнмо обратиться к макроопределению установки среды:

```
      AS     CSECT
             ASPL 0.1.PL
             END
```

В случае, когда устанавливается среда ПЛ/**1,** нет необходимо-дополнительно передавать управление вызываемому модулю, поскольку это делается автоматически по установлении среды.

4.1.5. Использование макроопределений передачи управления

Для передачи управления в простой структуре используется макроопределение CALLP, для построения динамической структуры LINKP. При использовании макроопределения LINKP необходимо - помнить, что вызываемый модуль должен быть отредактирован отдельно от вызывающего.

Пример 25.

Исходные модули

```
PL:   PROC OPTIONS (MAIN):
          CALL FORT#PF;
          . . .
          END PL;
          SUBROUTINE FORT
          . . .
          END
```

Модуль-связка

```
FORT#PF CSECT
. . .
CALLP FORT
. . .
END
```

Заключение

В главе 4 приведены элементы БМИ для формирования модулей-связей при сборке ресурсов, представленных в разных ЯП. Библиотека БМИ включает набор макроопределений преобразования данных и транспонирования матриц, задаваемых в ЯП (Algol, Fortran, PL/1 и др. Приведены их тексты на языке Ассемблера.
В следующей главе рассматриваются современные ЯП (C, C++, Basic, Pyton, Ruby, ADA…) и новые операции конфигурационной сборки разных модулей, объектов, компонентов и аспектов.

Литература

1. Лебедев В. Н., Соколов А. П. Введение в систему программирования ОС ЕС. М., Статистика, 1978, 187с.
2. Дал У., Дейкстра Э., Хоор К. Структурное программирование. М., Мир. 1975, 246с.
3. Лаврищева Е.М,, Грищенко В.Н. Связь разноязыковых модулей в ОС ЕС.-1982..-137с.
4. Редько В.Н. Композиции программ и композиционное программирование. – Программирование, 1978, № 5, с. 3–24.
5. Загацкий Б. А. Автоматизация реакторных расчетов. М., Атомиздат, 1974,-. 101с.
6. Глушков В. М., Вельбицкий И. В. Технология программирования и проблемы ее автоматизации. – УсиМ, 1976, № 6, с. 75–93.
7. Modular and structured programming techniques, Data Processing 17(1975), 5, p. 310–316.
8. Глушков В. М., Капитонова Ю.В., Летичевский А. А. О применении метода формализованных технических заданий к проектированию программ обработки структур данных. – Программирование, 1978, № 6, с. 31–43.
9. Система автоматизации производства программ (АПРОП). Киев, РФАП, 1976. - 134с.
10. Грищенко В. Н., Лаврищева Е. М. О создании межъязыкового интерфейса для ОС ЕС ЭВМ. – УСиМ, 1978, № 1, с. 34–41.
11. Лаврищева Е. М. Вопросы объединения разноязыковых модулей в ОС ЕС 78, № 1, с. 22–27.
12. Алферова И. А., Лихачева Г. В., Шураков В. В. Математическое обеспечение ЭВМ. М., Статистика, 1974.
13. Гальченко О. Н. О сопряжении программ, написанных на языках ПЛ и Фортран-4. – УСиМ, 1979, № 6, с. 69–71.
14. Орлов Б. Н., Пономарев А. С. Система генерации модульных программ. – Кибернетика, 1980, № 2, с. 82–84.
15. Браун П. Обзор макропроцессоров. Пер. с англ. М., Статистика, 1975, 77с.
16. Филина Л. Н. Вопросы связи модулей, транслированных с языков Фортран, ПЛ/1, Ассемблер, в ОС ЕС. – Программирование, 1980, № 3, с. 39–43.
17. Жоголев Е. А. Принципы построения многоязыковой системы модульного программирования. – Кибернетика, 1974, № 4, с. 1–5.

18. Жоголев Е. А. Технологические основы модульного программирования. – Программирование, 1980, № 2, с. 44–49.
19. Кахро М. И., Мяннисалу М. А., Саан Ю. П., Тыугу Э. Х. Система программирования ПРИЗ. – Программирование, 1976, № 1, с. 38–46.
20. Ершов А. П. Проектные характеристики многоязыковой системы программирования. – Кибернетика, 1975, № 5.
21. Лаврищева Е. М. Об автоматизированном изготовлении программных агрегатов из разноязыковых модулей. – УСиМ, 1979, № 5, с. 54–60.
22. Лаврищева Е. М. Методика модульного изготовления программных агрегатов. – Кибернетика, 1980, № 1.
23. Лаврищева Е.М.Технология программирования индустриальных систем. Семинар Новосибирск -78. Перспективы развития программирования. С.1-5.

Глава 5

Теория и практика обработки обмениваемых общих типов данных GDT при сборке ресурсов Интернет

Основы построения сложных систем из готовых ресурсов модульного типа метода модульного программирования

Первоначально в 1975году академик В.М. Глушков высказал идею на ученом совете ИК АН УССР после посещения IFIP-1974 о том, что «программы будут собираться конвейерным способом, как автомобили собираются из готовых деталей на фабрике Форда на основе болтов и гаек». При исследовании этой идеи сформировалось понятие конвейерной сборки программных элементов с помощью операции link, которая объединяет разнородные модули через интерфейс. Метод сборки обеспечивает связывание модульных объектов по данным, передаваемым между ними.

Сборка производится в операционной среде ЯП с помощью вызова link других модулей, с соответствующим обращением в тексте описания CALL/RMI/WSDL, которые обрабатываются компиляторами с ЯП и включают:

— данные для связи и управления модулей, находящихся как стандартные функции обработки данных в библиотеках операционной среды;

— системные данные ОС, списки идентификаторов функций, стеки точек передач и возврата управления;

— динамические данные из списка областей памяти ОС с регистрами и т. д.

Операции обращения в ЯП задают связь модулей, функций или подпрограмм из библиотек/репозиториев и описывается следующей функцией:

$$CP = K_1 + K_2,$$

где K_1 — коэффициент вызова; K_2 — коэффициент перехода от среды ЯП вызывающего модуля к среде ЯП вызываемого.

Для стандартного вызова модуля $K_1 = 1$, а для нестандартного — $K_1 = 1+ a$ ($a > 0$), где a зависит от количества нестандартных вызовов. К таким вызовам относятся: способ задания точки входа в вызываемом модуле; передача управления; адреса возврата в вызывающий модуль; передача списка параметров; метод сохранения и восстановления регистров для вызывающего модуля.

Коэффициент K_2 зависит от количества операций, необходимых для перехода от среды вызывающего модуля к среде вызываемого и наоборот. Аналитического выражения для K_2 не существует, но можно указать такие параметры - количество библиотечных модулей, входящих в среду; количество выполняемых ими функций; количество структур данных, входящих в среду; общий объем памяти, занимаемой программно-информационными средствами среды; ЯП вызывающего и вызываемого модулей. Если вызывающий и вызываемый модули написаны на одном *ЯП,* то $K_2 = 0$. Для остальных случаев $K_2 > 0$.

Виды интерфейсов для связи, сборки модулей в ЯП

Под *интерфейсом* понимается взаимосвязь модулей для совместного функционирования друг с другом. В зависимости от способа реализации, интерфейсы могут быть встроенными, внешними и внутренними. *Встроенный интерфейс* содержится в ОС ПО ЭВМ. Внешний интерфейс содержит описание внешних данных для передачи вызываемым модулям и включает описание паспортных данных модуля. *Внутренний интерфейс* является локальным и задается аналогично внешнего в описании текста любого модуля.

Интерфейсные данные, которые передаются между модулями, описываются операциями ЯП - CALL/RPC/RMI и языками WSDL, SPL, SBL. Spapol, RDF, XML и др.

Межъязыковый интерфейс включает описание типов передаваемых данных средствами различных ЯП, определяемых соответствующими системами программирования. Данный интерфейс обеспечивает:

1. переход от среды одного ЯП к среде я другого и обратно;
2. передачу управления между разными модулями;
3. доступ к общим данным, находящимся во внешней памяти сервера;
4. передачу данных из списка фактических параметров вызывающему модулю и необходимое преобразование к представлению, согласующемуся с ЯП описания параметров вызываемого модуля.

Преобразование данных состоит в устранении отличий в задании языковых и форматов представления данных системами программирования с ЯП. Оно выполняется перед и после выполнения вызываемого модуля (т. е. осуществляется прямое и обратное преобразование).

Межмодульный интерфейс включает описание данных в операторе link в параметрах передаваемого модуля в ЯП для обработки. *Связь пар разноязыковых модулей* сводится к устранению отличий в:

1) языковых средствах описания передаваемых параметров;
2) формальном описании отдельных модулей на разных ЯП;
3) способах представления модулей системами программирования ЯП.

1. *Отличия в языковых средствах* ЯП состоят в неодинаковости синтаксического и семантического представления типов данных ЯП и их функциональных возможностей. К ним относятся:

— способы конструирования новых типов данных, которые отсутствует в языках Фортран, ПЛ/1, Кобол и имеются в языках Паскаль, Ада, Симула-67, Модула-2, Си, Альфард, CLU и др.;

— некоторые предопределенные типы, которые отсутствуют в ЯП (например, символьного типа нет в ЯП Фортран);

— представление некоторых предопределенных типов, отличающиеся в разных ЯП (логический, boolean тип в языке ПЛ/1 задается как битовая строка);

— динамические типы данных отсутствуют в Фортране и Коболе и имеются в языках Паскаль, ПЛ/1, Ада, Симула-67 и др.;

— внешние отличия (ПЛ/1 допускает последовательную, индексно-последовательную и прямую организацию файлов, а Фортран не поддерживает индексно-последовательной организации файлов);

— дескрипторы для представления структурных типов данных имеются в некоторых ЯП и не требуются в Фортране и Коболе;

— представление некоторых структурных типов отличается в различных ЯП (массивы в Фортране задаются по столбцам, в других ЯП — по строкам) и др.

2. *Задача объединения* модулей через интерфейсного посредника связана с описаниями модулей и вызваны несоответствием описания данных, предаваемых через формальные и фактические параметры типа CALL (a, b) и состоит в следующем:

— отличное описание типов данных и областей значений переменных, индексов массивов и т. п;

— одному формальному параметру сопоставляется несколько фактических и наоборот;

— Разный порядок следования передаваемых параметров и др.

3. *К проблемам, связанным с реализацией* систем программирования с ЯП, относятся:

— наличие среды функционирования ОС и особенностей передач управления на разных платформах;

— различия внутреннего представлении однородных типов данных для систем программирования в разных архитектурах;

— различия в структуре и организации внешней памяти для однородных файлов.

Метод сборки разноязыковых модулей в рамках ОС ЕС ЭВМ реализует многие моменты отличий в представлении данных и преобразует друг к другу отличающиеся типы данных с помощью функций преобразования, приведенных в главе 4, реализованных на ЕС (IBM) Грищенко В.Н. [5, 6, 9, 10]. Метод сборки включает в себя математические формализмы определения связей (по данным и по управлению) между объектами сборки и реализации интерфейсных модулей-посредников для каждой пары объединяемых модулей.

Сущность метода сборки пары разноязыковых модулей состоит в определении взаимно однозначного соответствия между задаваемым 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 сборки для данной пары модулей считается неразрешимой. Передаваемые данные описываются формальными средствами ЯП, которые представлены в формальной теории типов данных (ФДТ).

Метод сборки реализован в 1976–1982 г. в рамках ВПК СССР в системе АПРОП и в описан в ряде статей и в этой книге «Связь разноязыковых модулей в ОС ЕС» [1-5], в которой представлена разработанная библитека из 64 интерфейсных примитивных функций для преобразования неэквивалентных данных для ЯП (Алгол, ПЛ/ 1, Фортран, Кобол**,** Ассемблер) в среде ОС ЕС (IBM 360).

Система АПРОП и библиотека 64 функций отработана в рамках ВПК (В.В. Липаев, МНИИПА, Минрадиопрома СССР). На основе АПРОП были созданы специализированные ЭВМ и программно-технические комплексы ПРОМЕТЕЙ, РУЗА, ЯУЗА и др. Все средства автоматизации программно-технических комплексов и слеланных технических приборов переданы в ЕрНУЦ 1984). Выполненные работы по ВПК были награждены премией СМ СССР по теме –Технология разработки бортовых систем (1985). Система АПРОП и сборка разноязыковых модулей передана по актам внедрения в 52 организации СССР. В частности, система АПРОП внедрена в ГДР (1971-1981) и там сделана АС производства аппаратуры для медицины и биологии.

Апробация конвейерной сборки по ТЛ проведена также в АИС «Юпитер-470» для военно-морского флота СССР (1982–1991). В рамках этого проекта создано шесть ТЛ (АСПИ, АСУ, АСНИ и др.). Эти ТЛ стали способом генерации отдельных программ по разным направлениям автоматизации АИС-470. На ТЛ в АИС было сгенерировано более 500 программных модулей обработки данных для АСНИ. Отдельные средства автоматизации программирования задач АИС-470 для флота, флотилии, подводных лодок, кораблей внедрены в Ленинграде, Мурманске, Владивостоке, Мурманске, Одессе (1991).

Новый вариант метода сборки после развала СССР. В настоящее время интерфейс сохраняет свою актуальность и выступает в качестве главной доминанты взаимодействия компонентов, объектов, сервисов в современных глобальных сетевых средах Интернет. Появились и новые ЯП спецификации интерфейса: интерфейс API (Application programs Interface), интерфейс IDL (Interface Definition Language), научный интерфейс SIDL (Scientifical IDL), WSDL Internet и др., а также средства обеспечения взаимосвязи разных информационных и интеллектуальных ресурсов ресурсов по ТЛ на примере многочисленных сформировавшихся современных фабрик программ.

Эти виды интерфейсов вошли в практику работы с программами на ЯП и используются и для современных ЯП с новыми фукциями преобразования. На конференции «Интерфейс СЭВ» 1987 обсуждались разные виды интерфейса в техниике, экономике, прикладных и технических системах ПС, ПТС. Лаврищева Е.М. и ее соавторы получили грамоты конференции «Интерфейс СЭВ» за определение межмодульного и межязыкового интерфейса. Эти интерфейсы и другие интерфейсы использовались в разных странах СЭВ при создании программно-технических продуктов разного назначения. Они реализованы для вычислительных сред (IBM, VS, NET, Intel, Unix, General Electric, Oberon, .Net, JavaEE, Grid, Etics и др.).

Научно-технические разработки по методу сборки под руководством В.В. Липаева привели к созданию в нашей стране сборочного программирования:

- Глушков В.М., Стогний А.А., Лаврищева Е.М. b lh/Система АПРОП. - К.: 1976. - 231с.
- Лаврищева Е.М., Грищенко В.Н. Связь равноязыковых модулей в ОС ЕС –М.: Финансы и статистика, 1982. – 127 с.
- Лаврищева Е.М., Грищенко В.М. Сборочное программирование. – Киев: Наук.думка, 1991. – 213С.
- Лаврищева Е.М., Грищенко В.Н. Технология сборочного программирования. Основы индустрии программных систем. - 2009. - 431с.
- Липаев В.В, Позин Б.А., Штрик А.А. Технология сборочного программирования. –М.: Радио и связь, 1992. – М.: Синтег. - 271с.

Создание технических и программных систем из модульных объектов (объект, компонент, сервис, программа) в ЯП осуществлялись с использованием стандартов по ЯП, ЖЦ и типам данных, рассматриваемых ниже.

5.1. Описание общих типов данных и метода конфигурации объектов

Общие типы данных в стандартах

Это стандарта ISO/IEC General Data Types (GDT)-1999, ISO/IEC General Pupoused DataTypes (GPD) - 2007, 2012;

IEEE 828 Configuration program object to structures: 2007, 2012. (Config, build, make, assemble).

К стандарту GDT относятся: ЯП, среды, системные интерфейсы ПО и Фундаментальные типы данных (1996); Типы данных используются представления и обеспечения совместимости данных при обмене; генерируемые, неструктурированные типы данных (GPD) и средства обеспечения взаимодействия объектов и типов данных (2012).

Далее представлены стандартные средства описания типов данных разной структуры (простые, структурные, сложные и неструктурированные), которые используются в ФДТ и GPD при сборке. Приводится описание стандарта конфигурирования (Configuration) сложных программно-технических систем из готовых информационных и интеллектуальных ресурсов, работающих со стандартными типами данных (ТД) и средствами обеспечения безопасности этих ресурсов и систем из них, защиты данных ресурсов, оценки надежности и качества ПС и ПТС.

Приведено формальное описание стандарта ISO/IEC 12207 Life Cycle (ЖЦ ПО, Систем) 2006, 2012 и подход к автоматизации вариантов данного стандарта по онтологическому описанию в языке DSL (Domain Spesific Language) инструментария Семантик Веб для описания функциональной структуры разных предметных областей знаний. Сделано описание онтологии домена ЖЦ стандарта описание ISO/IEC 12207 Life Cycle и представлено на конференции в Лондоне в 2015:

- Lavrischeva Ekaterina. Ontological Approach to the Formal Specification of the Standard Life ycle, "Science and Information Conference-2015", Jule 28-30, London, UK, www.conference.thesai.org.- p.965-972.
- Ekaterina M. Lavrischeva. Ontology of Domains. Ontological Description Software Engineering Domain—The Standard Life Cycle, Journal of Software Engineering and Applications, 2015, 8, p.1-15. Published Online July 2015 in SciRes.http://www.scirp.org/journal/jsea.

На конференции в Лондоне комитет IEEE предложил сделать патент на онтологию ЖЦ на анг. языке. Но в ИСП РАН была зарегистрирована программа ЖЦ - Гос.регистрация программы ЖЦ - Свидетельство №2018615442. - 2018. «Метод онтологического моделирования домена ЖЦ стандарта ISO/IEC 12207.

5.2. ТД в стандартах ISO/IEC 11404 GDT-1996 и GPD-2007

В 1996 году вышел стандарт "Информационные технологии – языки программирования, среды и системные интерфейсы ПО – Языки типов данных".

Начиная с 1996 года этот стандарт используется для обеспечения семантики типов данных ЯП, баз данных и других систем, которые используют типы данных (ТД). Пересмотр этого стандарта произошел через 7 лет под названием "Информационные технологии — типы данных общего назначения (GDT) для стандартизации и обеспечения совместимости данных».

Согласно стандарта ISO/IEC 11404 GDT ТД состоит из трех основных понятий: пространства значений, набора свойств и характеристических операций.

Пространство значений включает набор значений для каждого ТД, которые имеют определение:

– аксиоматических ТД из фундаментальных понятий,

– совокупности пространства значений,

– совокупности произвольных значений из некоторых уже определенных пространств.

Каждый ТД содержит свойства из шести характеристик:

– равенство между элементами пространства значений,

– наследование данных из области значений,

– конечное, бесконечное (счетное), или бесконечное (несчетное) множества значений пространства,

– точность по отношению к модели вычислений значений элементов области пространства,

– пространство концептуальных значений как точные или приближенные.

Например, вещественные числа могут только быть приближенными на любой платформе компьютера.

Другой особенностью ТД являются характеристические операции над значениями из пространства значений. Например, ТД целое число стандарта ISO/IEC 11404 GDT :1996 имеет следующие характеристические операции:

Equal(x, y: integer) – boolean is true if x and y определяют одно и тоже значение и false в противном случае,

Add(x,y: integer) – является операцией integer;

Multiply(x, y: integer) – является математической мульти операцией integer;

Negate(x: integer) – является значением y в такой операции Add(x, y) = 0 как integer;

NonNegative(x: integer): boolean есть true if x = 0 or x может быть разработан путем итерации 1, т.е. if x = Add(1, Add(1, ... Add(1, Add(1,0)) ...)) else false ;

InOrder(x,y: integer): boolean = NonNegative(Add(x, Negate(y)));

Quotient(x, y: integer) – integer, где 0 < y, является верхней границей множества всех integers z, таких, что Multiply(y,z) ≤ x;

Remainder(x, y: integer) – integer, where 0 ≤ x and 0 < y, = Add(x, Negate (Multiply(y, Quotient(x,y))).

Данный стандарт GDT:1996 предоставляет богатую библиотеку генераторов ТД (например, запись, массив и т. д.), которые могут быть использованы для создания сложных структур данных (поддерживаемых многими ЯП).

Ниже приведен простой пример использования нотации этого стандарта:

```
Type employee record = new record ( Name: character string, // employee name
Marital status: state (single, married),
exemptions: integer, // number of exemptions for tax deduction
pay rates: array (0..20) of pay_rate_type, // an array of records),
```

type pay_rate_type = new

record

(code: characterstring, // pay code

wage: scaled(10,4), // hourly wages to 4 decimal digits),

5.2.1. Язык ТД стандарта ISO/IEC 11404 GPD-1996

Для установки взаимосвязей и обеспечения взаимодействия между разнородными компонентами в разных ЯП современных сред, разработан стандарт ISO/IEC 11404 GDL:1996, который предоставляет возможность независимо от ЯП специфицировать разные сущности и типы данных для их передачи между компонентами. Этот стандарт содержит описание всех типов данных, механизмы их агрегации, упорядочение и преобразование на внешнем и внутреннем уровнях представления компонентов. Это альтернатива известным языкам описания интерфейсов – IDL, API, RPC и он занял доминирующее положение в программировании до появления WSDL.

Основа стандарта ISO/IEC 11404 – язык LI (Language Independent), метод генерации новых типов данных, механизмы преобразования типов данных ЯП в LI-язык, и наоборот. Стандарт предлагает специальные правила и операции генерации примитивных типов данных и типа объединение LI-языка в более простые структуры данных ЯП. Параметры интерфейса определяются средствами языков IDL, RPC и API.

Независимые от ЯП типы данных стандарта разделены на примитивные, агрегатные и сгенерированные. В этот язык включено семейство и генератор типов данных в форме XML-документов. Все существующие типы ЯП и общие типы данных ориентированы на генерацию других типов данных. Описание типов данных задается в разделе «declaration» обших типов данных, объявленных типов данных, объявленных генератором.

Раздел объявления типов данных включает описание, переименование существующих типов. Каждый тип данных имеет шаблон, который содержит в себе описание и спецификатор типа данных, значение в пространстве значений, синтаксическое описание и операции над типами данных. Для объявленного типа данных задается шаблон, который содержит синтаксическое описание, спецификатор типа данных, значение в пространстве значений и операции над типами данных.

Средствами языка LI описываются параметры вызова и интерфейсы, необходимые при обращении к стандартным сервисам и готовым программным компонентам. LI-язык предполагает такие виды преобразования данных:

– внешнее преобразование типов данных ЯП в LI-тип данных;

– внутреннее преобразование с LI-типа данных в тип данных ЯП;

– обратное внутреннее преобразование.

Внешнее преобразование типов данных и генераторов типов данных это:

а) обеспечение связи каждого примитивного типа данных с одним из LI-типом данных;

в) определение связи путем преобразования допустимого значения внутреннего типа данных в эквивалентное значение соответствующего LI-типа данных;

с) определение существующего значения для любого внутреннего типа данных с целью преобразования в LI-тип данных и извлечение этого значения.

Внешнее преобразование определяет аномалии при идентификации внутренних типов и дает гарантию того, что интерфейс между программными компонентами адекватно задаются сервисным средством.

Внутреннее преобразование связывает внутренний примитивный или сгенерированный тип данных с LI-типом данных с конкретным внутренним типом данных ЯП. Это преобразование для каждого LI-типа значений данных имеет такие свойства:

а) примитивное или сгенерированное преобразование при наличии этого типа данных в

ЯП;

в) преобразование определяет отношение между допустимым значением этого типа и эквивалентным значением соответствующего внутреннего типа ЯП;

с) преобразование внутреннего типа данных определяется значением после преобразования какого-то значения LI-типа данных.

Обратное внутреннее преобразование LI-типа данных заключается в преобразовании значений внутреннего типа данных в соответствующее значение LI-типа ЯП при наличии соответствия и отсутствия двусмысленности. Это преобразование – коллекция обратных преобразований LI-типа данных.

Предложенные в стандарте рекомендации, а также средства описания типов данных и методов их преобразования – общие. Новый вариант стандарта ISO/IEC 11404 -2007 (General Purpose Datatypes) дает общее описание типов данных, которые могут использоваться в новых ЯП, и иметь программную поддержку для всех типов данных современных формальных средств описания данных.

5.2.2. Трансформация ТД ISO / IEC 11404 GDT- 1996 и GPD 2007, 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 стандарте **ISO / IEC 11404:** GPD 2007, 2012 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 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 Big Data.

Вариант теории преобразования ТД GDT с Big Data рассмотрены в [14].

Таким образом, выше дано описание нового объектно-компонентного метода (ОКМ) [15] 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 [1-5]. 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 внедрена в проектах ВПК Липаева В.В. [5] и при поддержке академика А.П. Ершова в ИПИ СО АН СССР, ученики которого развивали теорию графов для программирования трансляторов и ОС ПО [7-14].

Начиная с 2003 года, теория графов начала применяться для моделирования сложных систем по методу объектного моделирования (ОМ) и компонентов [15] с применением операций конфигурационной сборки (assembler, config) согласно стандарта ISO/IEC 828: Configuration 1996, 2012. На основе графового описания программных структур реализуется сборка объектов и компонентов с применением аппарата изоморфного отображения неэквивалентных типов данных (структурных, неструктурных и генерируемых), передаваемых между отдельными объектами создаваемых программ.

Приведенные средства преобразования типов данных стандартов ISO/IEC 11404 GDT-1999?, 2001 и ISO/IEC 11404 GPD-2012 используются в парадигмах сборочного программирования.

5.2.3. Типы данных стандарта GPD 11404-2007, 2012

Общие ТД GPD (Genaral Purpose Datatypes) 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 языка спецификации 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 каждого 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 $a \leq b$ или $b \leq a$ или оба этих условия;

2) для любых двух значений (*a, b*), если $a \leq b$ и $b \leq a$, то $a = b$;

3) для любых трех значений (*a, b, c*), если $a \leq b$ и $b \leq c$, то $a \leq c$.

Запись $a < b$ используется для нотации: $a \leq b$.

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 $b \leq a$, 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 $s \leq U$. 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 $L \leq s$. 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) конечным;

2) точным и бесконечным;

3) приближенным (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 стандарта GPD

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 в GPD

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. **Aactual-type parameters** соответствует номеру и типу formal-type parameters объявленных в type-declaration. Таким образом, каждый aactual-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.

Ttype-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 ТД к реальным параметрическим.

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.

5.2.4. Типы данных стандарта GPD ISO/IEC 11404-2007, 2012

К общим типам данных (GPD - Genaral 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.

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 – результат генерации типов данных, с которыми работал 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;

в) 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.

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.) на рис. 25.

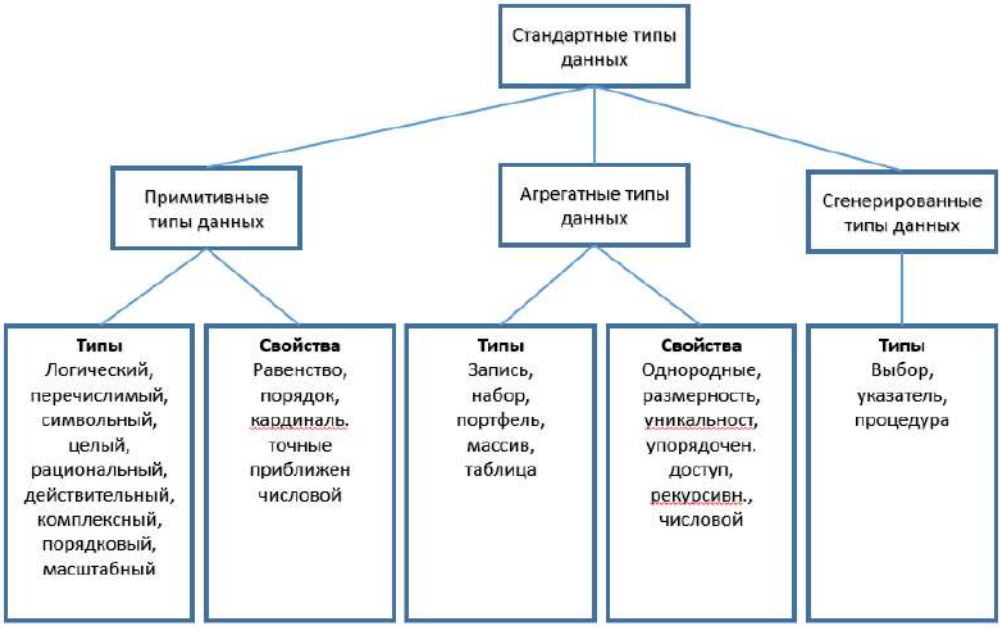


Рис. 25. Типы данных стандарта GPD

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.26) 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.

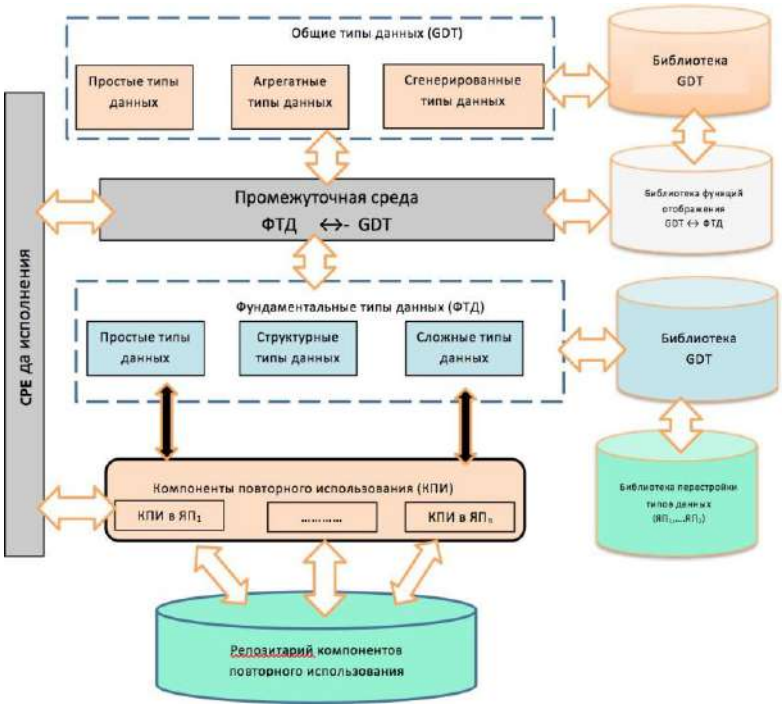


Рис. 26. Схема преобразования ТД стандарта GPD

5.2.5. 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 не реляционные данные, которые используют одну из трех архитектур:

- реляционные данные находятся в БД, а большие не реляционные данные двоичных объектов (BLOB), которые находятся в файловых системах или на файловых серверах;
- не реляционные данные из хранилищ, предназначенных для BLOB-данных;
- реляционные и не реляционные 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.

5.2.6. 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 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 [16].

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 в другое приложениеи 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 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.

СМВ— 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.

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 переносиз 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.

СМВ— 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.

5.2.7. Неструктурированные данные стандарта ISO/IEC 11404 GPD-2012

В результате проведенных нами исследований неструктурированных данных из класса больших данных (Big Data) и стандарта GPD-2012 приводится их определение.

Неструктурированные данные GPD – это совокупность данных, которые включают:

1. Структурированные ТД или метод доступа;
2. Наполовину структурированные данные, которые имеют один ТД или метод доступа;
3. Неструктурированные данные, включающие наборы данных неодинаковой природы (напрмер, сбор космических данных, авиационных, недр земли и др.).

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;

- не реляционные данные из хранилищ, предназначенных для BLOB-данных;

- реляционные и не реляционные данные, что находятся в БД.

Эти данные используются разными приложениями при:

- создании, загрузки, обновлении и удалении неструктурированных данных путем использования транзакционной согласованности между источниками неструктурированных данных;

- индексировании неструктурированных данных и использовании при вычислениях в Интернет;

- 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 использовании BLOB-данных в БД из/в централизованное хранилище снижаются затраты и быстродействие.

При разработке приложений, работающих с Big Data, используются средства анализа и описания данных [17–20]:

– A/B testing, crowdsourcing data fusion;
– integration genetic algorithms machine learning,
– natural language processing,
– signal processing simulation и visualisation,
– Parallel Processing Massiv,
– search-based applications data mining и др.

Большие данные могут быть представлены как tensors, которые управляют вычислениями, как например, полилинейное обучение подпространств (multilinear subspace learning). Технологии обращения к большим данным, включают массив параллельно-обрабатывающей (MPP) базы данных и извлечение данных на основе приложения, распределенные файловые системы, распределенные базы данных и инфраструктуры (приложения, хранение и вычисляющие ресурсы) Интернет.

К инструментальным средствам для работы с большими данными относятся:

– Oracle Designer,
– Oracle Developer, включающий Oracle Forms Oracle Discoverer и Oracle Reports,
– Oracle JDeveloper,
– NetBeans,
– Oracle Application Express,
– Oracle SQL Developer,
– OEPE, Pach for Eclipce.

Следует отметить, что рассмотренные подходы к обработке структурных и неструктурированных ТД являются техническими. Теоретической основы их обработки еще не сформировались. Теория создается при практических работах с большими данными.

5.2.8. Преобразования типов данных в среде JAVA

Несколько студентов МФТИ писали магистерские работы по тематике сборки специальных прикладных приложений приложений (Газазулина Р.И., Гарипов И.Т., Ларин Д.М. Иванов А. и др.) с использованием средств язык JAVA и системы JAVAEE в Интернет. В результате здесь представляется отработанная с их помощью сборочная методология.

В программах на этом языке используются операторы вызова удаленных методов RMI, позволяющие создавать распределенные приложения и обеспечивать их взаимодействие в

Интернет среде. Виртуальная машина работает с byte-кодами компонентов в других ЯП и, таким образом, обеспечивает взаимодействие компонентов в ЯП JAVA и C++, JAVA и PASCA и др.

К формальным средствам в системе JAVA относятся:

– оператор вызова удаленных методов RMI;

– сетевой обмен данными между удаленными компонентами;

– виртуальная машина для интерпретации битовых кодов элементов компиляторов C++ в среде JAVA.

Преобразование форматов данных. На каждой платформе компьютера где размещен транслятор с JAVA можно использовать соглашения о кодировании символов (например, ASCII, EBCDIC) о форматах целых чисел и чисел с плавающей точкой (например, IEEE, VAX, IBM и др.). Для представления целых типов (short, long) используется дополнительный код, для типов float и double – стандарт ANSI / IEEE, а для char - множество значений ISO Latin/1 [5, 133].

Порядок расположения байтов зависит от структуры процессора платформы (Big Endian или Little Endian), в частности от старшего к младшему байту и от младшего к старшему (например, в сетях Ethernet байты кодируются с младшего по значению бита). При этом существуют процессоры, которые поддерживают обе возможности (UltraSPARC, PowerPC) в зависимости от требований ПО. При передаче данных с одной платформы на другую учитывается расхождение в задании порядка байтов.

Для форматирования данных используется XDR–стандарт, который обеспечивает преобразование данных, передаваемых на другие платформы (SUN, VAX, IBM). На одной платформе программы, написанные на разных ЯП, могут использовать одни и те же данные в XDR–формате, хотя компиляторы их выравнивают по-разному. Кодирование (code) или декодирование (decode) данных выполняется с помощью XDR-процедур к виду представления в C++.

Кодирование – это преобразование локального представления в XDR–представлении и запись полученного результата в XDR–блок. *Декодирование* – это чтение из XDR-блока и превращение данных в локальное представление платформы, которая его принимает. Библиотека содержит процедуры форматирования для *простых и сложных типов данных* XDR–стандарта. Они используются для создания новых процедур, поддерживающих специфические локальные типы данных и сложные структуры. Для выравнивания данных в памяти используется стратегия, основанная на выборке из адреса данного, кратного действительному размеру в байтах (2, 4, 8, 16). Данные выравниваются по наибольшей длине, необходимой для размещения значений базовых типов. В результате в памяти могут появляться «дырки», которые должны устраняться. При обработке сложных структур данных компиляторы оптимизируют эти поля с помощью специальных процедур и функций из состава системных средств ОС.

Эти процедуры обеспечивают анализ форматов представления данных и необходимое их преобразование к формату платформы, которая их принимает. Соответствующий компилятор генерирует к ним обращение, базируясь на стратегиях выравнивания и размещении данных для платформ клиента и сервера. Принимающая платформа обрабатывает полученные данные, а потом декодирует их обратно к виду формата платформы, которая отправила эти данные.

Наиболее общим подходом к оптимизации процедур преобразования форматов данных – это методы интерпретации и компиляции. При *интерпретации* переданный код формата данного более компактный, а процедура преобразования работает медленнее. В процессе компиляции форматы и структуры данных существенным образом увеличиваются, хотя соответствующие процедуры работают намного быстрее.

Таким образом, преобразование форматов данных для передачи по сети включает взаимно обратные процедуры: кодирование и декодирование.

Средства интеграции и сборки компонентов в среде JAVA

Средства интеграции компонентов в JAVA. Основные типы компонентов в языке JAVA – это проекты, формы (AWT компоненты), beans компоненты, COBRA компоненты, RMI компоненты, стандартные классы–оболочки, JSP компоненты, сервлеты, XML–документы, DTD документы и файлы разных типов и др. [25]. Интерфейс является частью спецификации названных компонентов и способствует проведению интеграции компонентов в среде системы JAVA.

Шаблон развертывания представляет собою скрытую часть и необязательную часть абстракции компонента, который может быть повторно использован в одной или многих средах и для этого он имеет несколько шаблонов отладки. К спецификации компонента могут добавляться новые шаблоны интеграции или изменяться старые шаблоны. В некоторых классах ПИК параметры интегрирования в новую среду включаются в интерфейс компонента, что ограничивает способность компонента адаптироваться к этим средам и тем самым уменьшается круг задач, в которых он может повторно использоваться.

Проекты как средство композиции компонентов. Создание нового проекта состоит в задании конфигурации системы с помощью компонентов JAVA и обеспечении их взаимодействия следующими действиями:

– скомпилировать разные файлы с разными JAVA компонентами одной командой;

– установить основной компонент (класс) в проекте, который задает шаблон связей других компонентов в проекте;

– установить уникальную конфигурацию для каждого отдельного проекта,

– поддержать соответствующую файловую систему,

– установить уникальные типы компилирования, выполнения и отладки;

– подключить к работе иерархию окон.

Базовые операция проекта – это создание нового проекта, импорт компонентов из другого проекта, создание новых компонентов с помощью “Мастера шаблонов”, компиляция, выполнение и отладка группы подключенных к проекту компонентов как единой композиции. Проект обеспечивает разработку, сохранение и корректировку шаблона для поддержки взаимодействия разных типов компонентов при решении одной задачи. К шаблонам повторного использования относятся:

– *BlankAntProject,* определяющий первоначальный бланк проекта, в котором не содержится ни одного класса или пакета, но разрешается подключение классов и пакетов схему проекта;

– *SampleAntProject* предназначен для конфигурирования общей схемы проекта в виде иерархии системы файлов, ее корневого узла с последующим добавлением в нее компонентов, выполнять или доработки отдельных компонентов.

– *CastomTask* обеспечивает создание нового проекта, начиная с формирования первоначального класса в нем.

Классы – основа JAVA, порождаются с помощью ключевого слова Extends, после которого указывается тип компонента (например, JApplet). В проектах используются основной класс, с которого начинается выполнение проекта, и вторичный класс. К основному классу относится Class, Main, Empty (пустой класс), как шаблон типа:

– *exception* для создания класса, его исключений и сообщений об ошибках, которые могут быть обнаружены в программе;

– *persistence* Capable для отображения реляционной схемы БД без подключения к MySQL;

– *interface* – шаблон для создания нового JAVA интерфейса, который можно использовать любом классе через ключевое слово implements.

Для построения классов с помощью шаблонов используются стандартные классы–оболочки (Boolean, Character, BigInteger, BigDecimal, Class), а также класс строчных

переменных, класс–коллекция (Vector, Stack, Hashtable, Collection, List, Set, Map, Iterator) и класс–утилита (Calendar – работа с массивами и со случайными числами).

Формы. Интерфейсы компонентов содержат методы работы с графическими объектами и классы, реализующие эти методы. Они подключаются к AWT библиотеке классов, каждый из которых описывает отдельный графический компонент, применяемый независимо от других элементов. В AWT имеется класс Component, в котором графический компонент – это экземпляр этого класса. При выводе графического элемента на экран он размещается в окне дисплея, как потомок класса Container. Библиотека AWT содержит формы, каждая из которой представляет собою контейнер для размещения графических элементов интерфейса пользователя, а также систему классов Abstract Window Toolkit для построения абстрактного окна.

AWT форма построена на базе “тяжелых” интерфейсов (peer–интерфейс), а Swing формы – на базе “легких” интерфейсов. В разных средах AWT компоненты имеют вид, соответствующий данной среде, а Swing компоненты сохраняют этот вид (“plaf” – Pluggable Look and Feel) за счет того, что они разработаны средствами языка JAVA независимо от платформы. Все упомянутые окна применяются как контейнеры, к которым можно добавлять более простые графические элементы интерфейса (кнопки, меню и т.п.). Интеграция простых компонентов осуществляется на панели графики, изменения которых выполняются автоматически.

Аплет – это небольшая программа, доступная на Интернет сервере, автоматически устанавливается и выполняется веб–браузером или Appletviewer пакета JDK (Java developer Kit). Аплеты не выполняются JAVA интерпретатором, а работают в консольном режиме. После компиляции аплет подключается к HTML файлу, использующий тег <applet>. Компонент в языке JAVA Applet поддерживается набором стандартных методов инициализации, запуска, подключения аплета в требуемый веб контекст для работы с URL адресами и объектами типа Image и др.

Взаимодействие компонентов в системе JAVA

Для обеспечения взаимодействия разных типов компонентов используется механизм вызова удаленного метода RMI, который дополняет язык JAVA стандартной моделью EJB (Enterprise Java Beans) компании SUN. К ней подключены классы языка JAVA, определения их атрибутов, параметров среды и свойств группирования компонентов в прикладную программу для выполнения на виртуальной машине JVM. Механизм развертывания JAVA–компонентов типа beans на сервере базируется на программах в исходном языке, а сервер создает для них оптимальную среду для выполнения задач EJB.

Для реализации и повторного использования КПИ типа *beans* в системе разработаны следующие шаблоны Java for Forte:

– *Beans* для создания нового компонента и формирования каркаса компонента с простыми свойствами и возможностью автоматического изменения этих свойств;

– *BeanInfo* для интеграции beans компонентов и обеспечения взаимодействия;

– *Customizer* для создания панели, на которой размещаются элементы, которые со временем можно использовать для управления конфигурацией beans компонентов;

– *Property Editor* для создания класса, который используется во время проектирования и редактирования свойств beans компонентов.

Таким образом, в среде системы JAVA содержится богатый набор средств поддержки компонентного подхода и решения проблем интеграции и взаимодействия их в разных средах, совместимых с системой JAVA.

В настоящее время имеются механизмы обеспечения *связей языков Java* и *C++*. Они основываются на преобразовании Java классов, библиотеке классов C++ и языке описания интерфейса Java Native Interface. Основа взаимодействия компонентов в этих языках – аппарат установления соответствия класса интерфейса в C++ с Java-классом интерфейса.

Транслятор C++ продуцирует Java-прокси в C++класс, результат которого – код Java–C++ является интероперабельным. Реализация системы конвертирования релевантных типов данных между разно языковыми компонентами (Java↔C++, C++↔Java и др.) и при условии, что они расположены на разных платформах или гетерогенных средах, основывается на дополнительных описаниях компонентов и соответствующих доработках в компиляторах с этих ЯП для настройки к особенностям платформ гетерогенной среды.

5.2.9. Преобразование типов данных в TypeCode

TypeCode - средство обеспечивает преобразование значений базовых типов с помощью набора базовых примитивов в процедурах преобразования сложных типов и форматов представления данных:

– дополнительный код для представления целых чисел;

– числа с плавающей точкой (стандарт ANSI / IEEE);

– символы ISO Latin / 1;

– значения базовых типов, выравниваемых независимо от типов или данных, которые реализуется современными компиляторами;

– дополнительные базовые типы языка IDL (64-разрядный целый тип – signed и unsigned, как тип двойной точности и др.).

Процедуры преобразования типов реализуются путем интерпретации TypeCode для каждого типа языка IDL и интерфейса с методами доступа к сохраненной информации. Он содержит поле со значением типа TCKind и дополнительные параметры, которые отвечают этому значению. Тип TCKind определяет множество видов TypeCode базовых типов (tk_long, tk_float) и вид конструирования составных типов (tk_struct, tk_sequence).

Для каждого типа данных, определенного пользователем, константы TypeCode порождаются компилятором по спецификации в языке IDL.

Преобразование данных осуществляется с помощью процедур encoder ( ) и decoder ( ), интерпретатора TypeCode.

Параметрами процедуры encoder( ) – это данные для преобразования, которое отвечает TypeCode, и указатель на буфер.

Таким образом, достоинством метода преобразования на основе TypeCode, есть компактность выполненного кода, унифицированность использования и единообразие при работе с данными любого типа и произвольной сложности. Интерпретатор позволяет превратить данные, типы которых не известны при компиляции или во время выполнения.

Типизированные функции преобразования данных

Преобразование сложных данных осуществляется с помощью функций отображения типов, описанных в IDL. Трансформация данных типа *struct*, например, включает последовательное преобразование всех ее полей, в порядке, указанном в спецификации функции в языке IDL, в язык C++. Эту функцию выполняет компилятор IDL, порождая файлы отображения в соответствующие конструкции C++, набор вспомогательных процедур, необходимых для обращения к брокеру ORB. Для каждого типа IDL имеются соответствующие процедуры их преобразования в C ++.

Функции преобразования базовых типов учитывают информацию о границе выравнивания и размере данных, совпадающими с методами класса CDR, а также реализуют преобразования составных типов, имеющих вложенную структуру, с помощью inline–подстановок. Данные типа array преобразуются специальными функциями и процедурами для простых типов данных.

Типизированные функции и процедуры такие, как кодирование и декодирование, имеют симметричные структуры с точностью до базовых процедур и других специфических действий, используемых для отображения типов данных языка IDL в тип данных языка C++.

5.3. Метод сборки интеллектуальных и информационных ресурсов Интернет

5.3.1 Конфигурационная сборка ресурсов по стандарту IEEE 828: Configuration

Этот стандарт 2012 обеспечивает конфигурирование информационных, сервисных и системных ресурсов. Его основу составляет:

- управление конфигурацией (Configuration Management);
- аудит конфигурации (Audit Configuration);
- базис конфигурации — BC (Configuration Baseline);
- элементы конфигурации (Configuration Item);
- компоненты, ГОР, входящие в описание моделей ПС и СПС - 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.

Конфигурационный аудит

Аудит – это ревизия или проверка перед выпуском очередной версии ПО или перед сдачей системы заказчику. Кроме того, в обоих случаях аудиторская работа большей частью связана с рассмотрением и оценкой документации – данных, сводок, отчетов. Данный аудит производится непосредственно перед выходом новой версии продукта, его части, т.е. практически всегда исходит из ответственности момента по тем или иным обязательствам перед заказчиком. Конфигурационный аудит включает работу по двум взаимосвязанным направлениям:

– функциональный аудит конфигурации для подтверждения соответствия фактических характеристик конфигурации/единиц продукта предъявляемым требованиям;

– физический аудит конфигурации, как подтверждение взаимного соответствия документации и фактической конфигурации продукта.

Функциональный аудит - это не верификация или валидация продукта путем тестирования, а проверка того, что тестирование проведено в установленном объеме, результаты документированы и подтверждают соответствие характеристик продукта предъявляемым требованиям. При этом все изменения реализованы, критичные дефекты устранены, а по всем выявленным отклонениям от конфигурационного базиса приняты адекватные решения. Он обеспечивает сверку выпущенного продукта согласно документов конфигурационного базиса, а также проверка того, что данная конфигурация построена в соответствии с установленными процедурами и из корректных версий соответствующих компонентов. Конфигурационный аудит проводится независимыми экспертами, например - представителями службы качества.

При конфигурационной сборке ресурсы используется модель систем и модель характеристик MF (Model Feature). КПИ хранятся в репозиториях или библиотеках систем в Интернет. Они выбираются их библиотек, адаптируются и интегрируются (собираются) в прикладную систему.

Основную роль в этих процессах выполняет конфигуратор (рис.5) http://7dragons.ru/ru). Конфигуратор управляет созданием варианта готового продукта и сохраняет его в

репозитории. М*одель среды конфигуратора* включает: графовую структуру системы из ресурсов; модель вариантов системы; конфигурационную модель и операции сборки КПИ; верификация моделей и ресурсов; оценку качества КПИ и систем.

5.3.2. Сборка информационных и интеллектуальных ресурсов Интернет

Сущность интеллектуальных и информационных ресурсов

Интеллектуальные ресурсы (ИНР) – это компоненты повторного использования (КПИ и reuses), отображающие знания специалистов из разных областей знаний. Любая функция предметной области знаний 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 - это информационные объекты, данные которых относятся к фундаментальным данным FDT и общим типам данных GPD стандарта ISO/IEC 11404, а также могут оперировать с большими объемами данных (Big Data). КПИ, ИНР взаимодействуют с модулями, объектами или сервис-компонентами Интернет через интерфейс в языках IDL, API, ABI, WSDL и др.

Reuse — это готовый интеллектуальный объект, компонент, сервис, реализующий функцию некоторой предметной области знаний. Специфицируется в языке WSDL, IDL и может многократно использоваться, если удовлетворяет требованиям задач предметных областей. Reuses имеют код (implementation), интерфейс и схему развертки (deployment) для выполнения. КПИ, Reuses имеют общие переменные, которые входят в **классы** или **множества**.

Интерфейс — это спецификатор информационной части ИНР — КПИ, reuses для обращения к другим ресурсам и обмена данными между ними. Интерфейс содержит оператор вызова метода или функции (RPC/RMI/WSDL) с параметрами, которые управляют внешними переменными экземпляра класса (выборка значения *get-метод,* присвоение значения *set-метод, Home-интерфейс* в языке JAVA и др.) для их взаимодействия. Интерфейс в языке WSDL содержит:

- *название* функции (метода) и *ID* — идентификатор ресурса;
- *описание (спецификацию)* функции средствами ЯП;
- *параметры* (входные и выходные) для передачи данных другим КПИ;
- описание КПИ в ЯП (C, C++, Java, Python, Ruby и др.);
- *необязательные атрибуты* (дата, состояние, версия, право доступа, автор, срок использования и т.п.).

Информационные ресурсы (ИР) – это данные разного объема, в том числе и Big Data, представлены в Базах Данных, файлах, каталогах**,** таблицах, контейнерах 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. Языки JAVA, WSDL включают следующие 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) последовательности байт (xsd:base64Binary, xsd:hexBinary);
5) дата и время (xsd:time, xsd:date, xsd:g);
6) объекты (xsd:anySimpleType).
7) множества, последовательности, массивы и др.

Интеллектуальные ресурсы - ИНР включают: модели данных, операции хранения, доступа и обработки данных. Выполняется доступ к данным и взаимодействие с ресурсами

типа entity в модели EJB. При работе с большими объемами данных ETL (Extract Transform Load) переносит данные из одного приложения в другое через клиент-серверную архитектуру и проводит извлечение данных из внешних источников. После извлечения проводится анализ и трансформация данных к виду требований архитектуры среды Интернет.

Сервисные ресурсы Интернет. Это - *Web-сервисы,* основанные на открытых стандартах Интернет, и широко поддерживаются на платформах Unix, Intel, VS.Net, IBM, Linux и др. Они 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 надежность функционирования Web-сервисов и систем. К средствам создания прикладных систем в Интернет относятся сервис-ориентированная архитектура SOA (Service Oriented Architecture) и сервисно-компонентная архитектура SCA (Service-Component Architecture).

SOA и SCA задают функциональность (Functions) и качество сервисов (Quality service) на транспортном уровне (transport layer) для обмена внешними данными на коммуникационным уровне (service communication layer). Описываются в ЯП сервиса и интерфейса (service description layer) со средствами безопасности и защиты (авторизации, аутентификации)*.

В Интернет среде имеется реестр сервисов, который содержится описание сервисов в языках Семантик Веб (SOA, SCA, SMC и др. и обеспечивает регистрацию, поиск и вызов сервисов по запросам поставщиков и потребителей сервисов.

Системные ресурсы сети Интернет. К ним относятся**:** клиент-серверная архитектура сети Интернет: клиент, сервер приложений (систем) и сервер БД. На уровень *клиента* обрабатываются простые сервисные ресурсы: интерфейс, операции с данными, алгоритмы шифрования, взаимодействие и безопасность для передачи серверу приложений и т.п. Клиентская часть отвечает за отправку интерфейса и обработку приложений, которые записаны на ЯП (C, C++, Python, Ruby, Java, Basic и др.), и выполняют прикладные функции с обработкой возникающих аварийных ситуаций и отказов выполнения.

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 выполняется анализа данных большого объема и манипулирования данными. *Клиент* работает с Интернет-браузером (Chrome, Firefox, MS Internet Explorer, Safari, Opera и др.), посылая сообщения серверу через интерфейс следующего вида: $WebAppInterface = \{Request^p\}$, где $Request^p$ — р-й запрос. Обработку запросов производит сервер: 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.).

* Лаврищева Е.М., Рыжов А.Г. Подход к моделированию систем и сайтов из готовых ресурсов.- XX Всероссийская конференция «Научный сервис в сети Интернет» (17-22 сентября 2018. –М.: ИПМ им. Келдыша.-2018.-321-345.-URL https://keldych.ru/abrau/2018/thesis/50pdf/, doi 10.20948/abrau-2018-50/

5.3.2.1. Основные операции сборки в среде Интернет

К основным операциям вызова объектов сборки относятся CALL/RPC/RMI/ WSDL для современных ЯП и к операциям сборка систем из готовых ресурсов относятся:

- llnk, make, building, assembling ($C1 \cup C2 \Rightarrow C3$) – виды сборки;
- Config ($C1 \cup C2 \cup C3 \Rightarrow C4$) – конфигурационная сборка;

- Reing (Cn) – реинженерия ресурса;
- Revers (Сявляются) – реверсная инженерия ресурса;
- Berification and validation (V&V) КПИ;
- client-server protocols Internet для обработки обмениваемых ресурсов;
- Creat classes (Cj) – образование классов ресурсов;
- depl - инсталляции (развертывания);
- del – удаление ресурса.
- addim – добавления реализации ресурса;
- addIn – добавления интерфейса ресурса;
- replIm – замещение реализации ресурса,
- replIm – замещение интерфейса и др.

К основным процессам технологии сборки относятся[1]:

1. Проектирование графа G из модульных объектов.
2. Выделение функциональных элементов или их поиск в Библиотеках Интернет
3. Трансляция и анализ ошибок в ресурсных компонентах, КПИ
4. Разработка функционального модуля, отладка, тестирование и запись в Библиотеку.
5. Определение интерфейсов для новых модулей (внешних,межмодульных, пользовательских).
6. Конструктивный контроль КПИ.
7. Доработка КПИ и занесение в Библиотеку.
8. Сборка КПИ (link) по маршруту (1, 2, 3) графа системы.
9. Комплексная отладка и тестирование системы из КПИ.
10. Проверка системы на контрольных данных.
11. Оценка качества по стандартам ISO/IEC 9126.
12. Документирование собранной системы.
13. Генерация руководства пользователя для эксплуатации
14. Проверка надежности и качества готовой системы.

Сборка модулей в ЯП в системе АПРОП для первых ЯП рассмотрена в гл.3, и задавалась оператором link (M_1, M_2) модулей. На данный момент времени при стандарте конфигурационной сборки ресурсов Интернет EEE 824 -1996 производится с помощью новых операциий – make, build, cmake, config, sassembly и др. с использованием книги

Липаев В.В., Позин Б.А., Штрик А.А. Технология сборочного программирования М.: Радио и связь, 1992.-271с.

5.3.2.2. Современные способы сборки сложных систем в Интернет среде

1). Способ сборки make реализован в системах BSD, GN5.5.1 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). См. рис.27

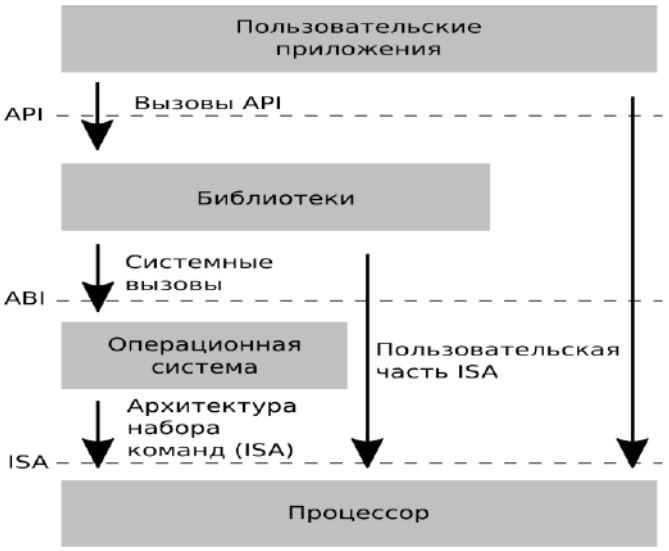


Рис. 27. Схема обработки операцией make

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.28. Пример сборки модулей в C++ и Fortran

2). Способ сборки через интерфейс Stub-клиент и Skeleton-сервера. В языке IDL (Interface Definition Language) дается описание интерфейсов ресурсов 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 АПРОП в Интернет (рис.29)

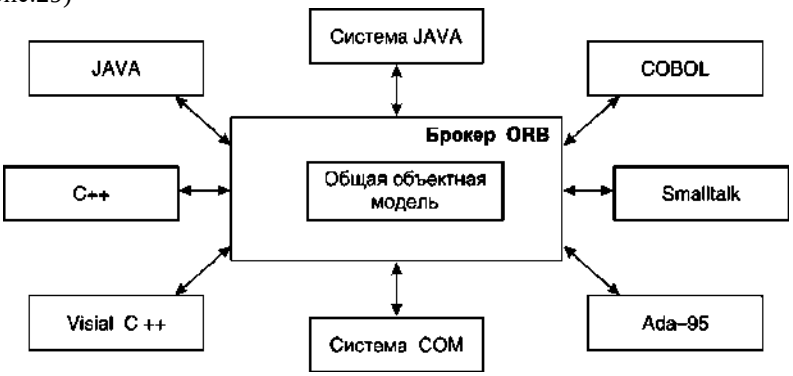


Рис. 29. Функциональная схема брокера ORB

3). Способ сборки в среде .Net основана на обработке передаваемых данных между модулями с помощью системы CTS (Common Type System, рис.30). 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.

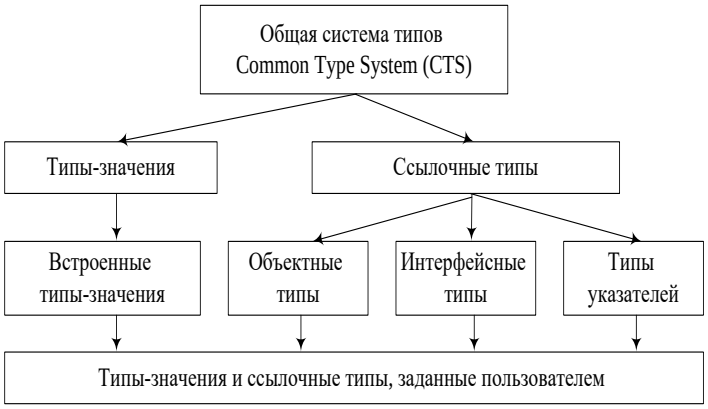


Рис.30. Схема сборки CTS

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 исключительная ситуация 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. Рис 31. обеспечивает обработку и управление программными, сервисными и др. Web ресурсами глобальной сети. Сборка разнородных сетевых и системных ресурсов в системы, приложения и пакеты, работает с данными разного объема в системах ETICS GRID (рис.31).

Ресурсы описываются в разных современных ЯП. Базовые сущности систем, пакет и приложений, а также связи описываются в CIM (Common Information Model).

Описание модели данных основано на следующих базовых положениях:

1) каждый компонент содержит описание сведений (имя, лицензия, URL репозитория и т. п.) и глобального 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;

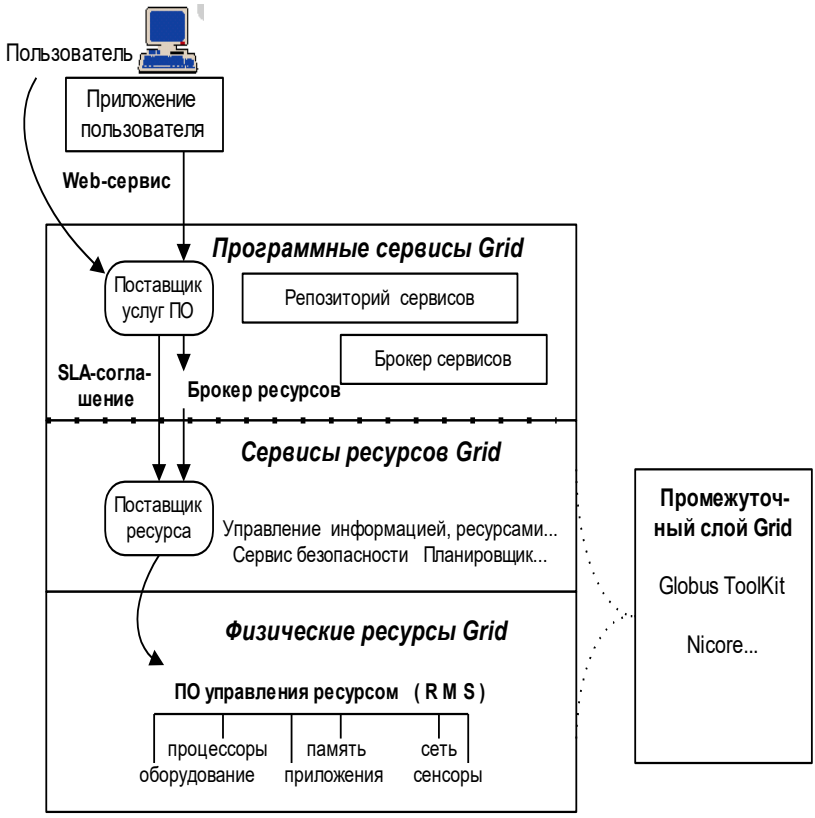


Рис.31. ETICS GRID

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.

В системе ETICS, как и Grid, 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.

Система ETICS по функциям соответствует 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. 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 (appfab) 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. 5.1) [24].

Таблица 5.1. 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 систем в соответствии с требуемыми сервисных ресурсов моделейдля SOA в Visual Studio Industry Partners (VSIP). При этом используются модели Guidance Automation eXtensions (GAX) и Guidance Automation Toolkit (GAT) в Visual Studio. GAX — для исполнения VSIP с помощью 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-служб. WCF находятся у разработчиков фабрики ПО GotDotN. В ее состав входят процессы предприятия и пакеты документаций и рекомендаций для приложения типа Global Bank, Global Fиbliications с использованием модели SCA, SOA в IBM WebSphere.

Средства описания ресурсов для создания ПТС

Глобальная сеть Интернет WWW (World Wide Web) обеспечивает интеграцию мультимедийных объектов, данных и разнообразных инструментальных сред через аппарат гипертекстов. Суть этой технологии состоит в том, что в тексте выделяется слово-ссылка (так называемая гиперссылка – фрагмент текста или изображения), с помощью которого выполняется переход к нужному документу или его разделу [26].

К средствам описания разных ресурсов Интернета относятся:

– язык гипертекстовой разметки HTML (HyperText Markup Language);

– универсальный способ адресации ресурсов в URL (Universal Resource Locator);

– протокол обмена гипертекстовой информацией HTTP (HyperText Transfer Protocol);

– универсальный интерфейс шлюзов CGI (Common Gateway Interface).

Язык HTML – это средство описания структуры документа и связей его с другими

документами. Этот язык предназначен не для форматирования документа, а для его функциональной разметки, создания мультимедийной структуры.

Язык URL предоставляет универсальную форму адресации информационных ресурсов с помощью указателя на ресурс баз данных WWW. В нем задается HTTP протокол и полное имя машины или адрес порта/путь, в который принимается документ, как гипертекст. Протокол передачи файлов ftp используется для доступа к локальным файлам.

Протокол HTTP – это совокупность правил для общения навигатора браузера с веб–сервером. Он предназначен для обмена гипертекстовыми документами с учетом специфики информации, которая реализуется с помощью ASCII-команд. При этом используются общий интерфейс CGI (Common Gateway Interface), как универсальный интерфейс шлюзов, разработанный для расширения возможностей Интернета в целях подключения любого внешнего ПО, которое вносится в среду WWW как средство интерактивного диалога с пользователем для создания интерактивных форм и динамических документов.

Программа-клиент (браузер) выполняет функции интерфейса с пользователем и обеспечивает доступ ко всем информационным ресурсам (ИР) Интернета. Клиент выполняет разные функции: запуска внешних программ на серверах для работы с документами в форматах, отличных от HTML.

Сервер протокола HTTP обеспечивает доступ к БД документов в формате HTML, управляемым сервером, программы которого разработаны в стандарте спецификации CGI. База данных HTML-документов является частью файловой системы, которая содержит текстовые файлы в формате HTML и связанные с ним другие ресурсы, в том числе и графические. Программы, которые обеспечивают взаимодействие сервера с серверами других протоколов или с распределенными на сети серверами баз данных, принимают данные от сервера и выполняют соответствующие действия (например, получение текущей даты, реализацию графических ссылок, доступ к локальным базам данных или производятся некоторые расчеты). На основе языка XML создаются: объектная модель документа DOM (Document Object Model); средства определения типа документа DTD (Document Type Descpiption); каркас описания ресурса RDF (Resource Definition Framework).

Semantic Веб – задает веб данные, которые управляются и поддерживаются приложениями. Semantic Веб обеспечивает интеграцию распределенных метаданных, используя описание объектов (или ресурсов), включающих описание атрибутов и значений. Ресурсы задаются URI идентификаторами, как Web-страницы задаются URL. Каждый атрибут ресурса вместо коротких имен использует URI–идентификаторы. Это связано с необходимостью представлять распределенную информацию в Web, децентрализовано описывать некоторый ресурс в RDF документе, указывать ссылку URI на значение, аналогично тому, как в HTML указывается ссылка на другую веб-страницу. При этом описание ресурса может считаться не полным и окончательным, поскольку в Веб системе может содержаться и дополнительная информация об ресурсах в URI, либо в нескольких RDF-документах.

RDF – язык представления метаданных о веб-ресурсах в WWW, их сущностях, идентифицированных средствами веб-идентификаторов (URI), о данных приложений, которые обмениваются данными в обобщенной среде.

Каждый ресурс описывается в терминах свойств и их значений, а также задаются в графическом виде с помощью вершин и дуг графа, описываемых в XML (RDF/XML) и URI

Вершинами графа могут быть субъекты и объекты. В качестве субъекта может выступать предикат, содержащий свойства. Объект может выступать в роли субъекта в другой тройке графа RDF. Объединение троек (субъект, объект, свойство) в графе означает их конъюнкцию. Квалифицированное имя Qname в XML обеспечивает запись идентификаторов в тройках и RDF/XML через префикс, которому присваивается URI и локальное имя. Иными словами, URI образует словарь терминов, совокупность взаимосвязанных терминов – URIref, как общие префиксы URI. URIref дает возможность отличать свойства, определяемые одним человеком, от свойств, определяемых другими

(или в другом контексте). Например, name может означать ФИО человека, или название сценария, или переменной в некоторой программе. Так как свойства являются ресурсами, то о них можно записать дополнительную информацию (например, описание на русском языке свойства "name"). Создается предложение в RDF, в котором свойство URIref выступает в качестве субъекта. Значения свойств могут быть структурированными и содержать информацию типа: город, район, улица и др.

RDF представляет бинарные связи. Для задания n-арных связей вводится дополнительная сущность, которая разрешает встроенные типы данных (исключая URIref rdf:XMLLiteral для представления литерального значения XML-контента. Данные, представленные с помощью литерала, могут быть заданы и URI. Литералом может быть только объект и предикат. Простой литерал – это строка с факультативной меткой языка, а типизированный литерал – это строка с URI, т.е. со значением из пространства значений, полученное отображением «лексика – значение» в строку литерала (т.е. лексическое значение).

RDF-схема обеспечивает описание словарей классов и атрибутов. Стандартизация RDF-словарей позволила включать свойства метаданных для конкретных предметных областей В стандарте зафиксировано использование терминов (свойств, значений и т.п.), что дает возможность прикладным программам интегрироваться между собою и обмениваться понятной для них информацией. При получении данных из другой системы текущая программа может найти себе свойства, регламентированные стандартом, и соответственно правильно проинтерпретировать их содержание и семантику. Такая особенность называется *семантической интероперабельностью* в Semantic Веб.

Стандарт W3C (World Wide Web Consortium) обеспечивает спецификацию веб-сервисов и взаимную увязку стандартов спецификации для описания веб-сервисов. Инструментом для согласования стандартов является XML-схема, в форматах которой представляются стандарты и соответствующий набор тегов. Виртуальный способ интеграции веб-сервисов в прикладное приложение обеспечивает взаимодействие сервисов между собою через распределенные вызовы объектов в интегрированном международном стандарте передачи сообщений между разными сервисами, удаленных вызовов процедур с помощью стандартного протокола коммуникации SOAP (Simple Object Access Protocol). Информация представляется как XML-схема, в которой элемент XML определяется в виде заголовка и тела. Заголовок определяет цель сообщения, а тело – его содержание, типы параметров сообщения, ответа и т.п.

Интерфейс веб-сервиса – это коллекции точек взаимодействия (end points), через каждую из которых можно получить от веб-сервиса определенную услугу средствами обмена сообщениями стандарта WSDL (Web Services Description Language) и форматом XML. Ресурсы описываются в стандарте WSDL в виде набора именованных интерфейсов и типов данных, изоморфных простым типам данных. На этих идеях и концепции был построен портал IT-Портал «E-Science», который предоставляет средства для выбора в Интернет сервисных ресурсов, данных из СУБД Интернет и построение разных вариантов сайтов в другой организации.

5.3.3. Общий сборщик интеллектуальных и информационных ресурсов в Интернет

В качестве общего вывода по рассмотрению операций сборки для общих Интернет сред можно сказать, 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 имеют схожие операционные способы сборки разных

вариантов ресурсов: модулей в современных 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 общей. Способ сборки зависит от данных, которые передаются при обмене данными между связываемыми разнородными информационными, сервисными, интеллектуальными ресурсами с учетом типов данных и форматов их в разных средах.

Рассмотренные средства генерации общих типов данных стандарта ISO/IEC 11404 GPD – 2012 требуют создания набора новых примитивных функций для неструктурированных данных, полученных при исследовании космоса, океанов, недр земли с помощью нестандартных типов (портфелей, контейнеров, шаблонов, указателей, неструктурных данных и др.) и использоваться в названных способах сборки разнородных ресурсов с преобразованием неэквивалентных данных в средах Интернет.

Для создания общего «сборщика» готовых ресурсов в прикладные и технические (макроконвейерные) системы, необходимо на базе конфигурационной сборки сделать следующее:

1. Определить формальный вариант операции сборки разнородных ресурсов.
2. Создать библиотеку примитивных функций для всех формальных типов данных и дорабатывать примитивные функции для новых ТД согласно стандарта GPD.
3. Создать анализатор операции сборщика и взаимодействия (компиляторных, системных, прикладных, системных, технических) с учетом всех типов ресурсов.
4. Проводить анализ соответствия типа ресурса и взаимодействия с другими ресурсами через интерфейс и осуществлять обращение к соответствующим программам преобразования обмениваемых данных для генерации соответствующего преобразования по теории преобразования типов данных стандарта GPD.
5. Управлять сборкой конфигурации архитектуры системы из ресурсов при выполнении всех процессов, определенных в стандарте IEEE 828-2009 Configuration.
6. Выдавать сведения о результатах сборки (ошибок и их устранения), оценивая надежности и качества созданной системы для выдачи сертификата на созданный продукт. и завершения работы сборщика.
7. Обеспечить размещение готовых ресурсов в библиотеках и хранилищах Интернет из разных предметных областей знаний (медицина, физика, биология генетика, математика, химия 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) вариантов конфигурационного файла с получением качественных решений функциональных задач приложений в разных предметных областях знаний.

Краткое назначение конечной продукции, технологии или услуг, которые будут производиться с применением полученных результатов.

5.3.4. Технология сборки общего назначения на ТЛ

Описание основных информационных ресурсов и ТД

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 определяется двумя способами: множеством всех значений, принадлежащих типу; предикатом, определяющим принадлежность объекта к

типу.

Тип данных – это описание данных в программном ресурсе 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. ТД задает множество значений, набором операций, которые применяются к таким значениям, а также операциями реализации, хранения значений и выполнения операций над ними. К средствам описания ТД относятся:

1. FDT (Fundamental Data Type) - Фундаментальные типы данных (см.1.2.7.).
2. GDT (General Data Types) - Общие типы данных,
3. Big Data – Большие данные.

Проблема преобразования ТД в программах на ЯП

Под преобразованием данных в программах на ЯП будем понимать способы определения видов отличий в представлении ТД в разных ЯП, методы преобразования и отображения типов и форматов передаваемых данных к соответствующему формату виду ресурса. Проблемы преобразования ТД в ЯП связана с несоответствие количества параметров в списке формальных и фактических параметров и несоответствием значений типов фактических и формальных параметров, передаваемых данных на ЭВМ сети; отсутствие обратных преобразований ТД параметров.

Для преобразования ТД каждый элемент данных Di = {dij) j=1,t определяется тройкой: именем Ni j, типом Ti j и значением этого типа Vij.

Функции преобразования ТД в ЯП задаются отображениями:

FNi k : Ni → No , FTi k : Ti → Tk , FVi k : Vi → Vk,

Отображения FNik, FTik, FVik должны содержать одинаковое количество элементов. В задачу преобразования FNik входит упорядочение имен переменных в описании связываемых между собой программных модулей.

Отображение FTik базируется на множестве T = (X, Ω), где X – множество значений, которые принимают переменные множества V и Ω – множество операций и предикатов преобразования типов.

ISO/IEC 11404 General Data Types - 1996, 2007 данных.

Алгебраическая системы для ТД GDT имеет вид:

Gα t = <Xαt, Ωαt >,

где t – ТД,, Xα t – множество значений переменных ТД,

Ωα t – множество операций над этими ТД.

Для простых, сложных и неструктурированных ТД ЯП разработаны классы алгебраических систем:

Σ1 = <G α b , Gα c , Gα i , Gα r >,

Σ 2 = <Gα a , Gα z , G α u , Gα e>,

Σ 3 = <Gs k , Gs n , Gg t >,

где Σ1 - простые ТД: t = b (bool), c (char), i (int), r (real)),

Σ2 – сложные, структурные ТД: t = a (array), z (record), u (union), e (enum), как комбинации простых ТД;

Σ3 - сложные (s), неструктурированные (n) ТД (портфель, контейнер, протокол и т.п.) и сгенерированные из них простые данные (см. раздел 5.3.2).

5.3.5. Обработки ТД в Microsoft.Net

В .Net реализованы ТД с помощью системы общих типов CTS (Common Type System), включающей:

Встроенные типы-значения, Объектные типы, Интерфейсные типы, Типы указателей, Типы-значения, Ссылочные типы.

CTS включает: типы-значений и ссылочные типы, заданные пользователем; - отображает типы значений, ссылочные и встроенные; формат данных компьютера описывается в CLR (Common Language Runtime) для подпрограмм и обработки целочисленных данных с плавающей запятой, строки биты;

- предоставляет язык спецификации CLS классов, структур, интерфейсов, встроенных ТД, перечислений и др.;- переводит данные на любом ЯП Microsoft.Net в промежуточный язык MSIL для конвертирования и взаимодействия объектов между собой.

- MSIL преобразуется в специфический код CPU для выполнения программ на разных архитектурах компьютеров.

Cloud Computing и обработка данных в Интернет

Это парадигма для распределенного широкомаштабного компьютера, в котором содержится набор абстрактных виртуальных, динамически масштабированных управляющих машин, памяти, платформ и сервисов, задаваемых через Інтернет, как - генетические хранилища (время, 100% точность восстановления, в 1 г – 2000 ТБ);

- IBM 154 TB (размер в 62 раза больше, чем LTO-6, технология Nanocubic).

ClCloud computing предоставляет для информационных технологий средства SaaS (Software as a Service) для удаленного применения сети VPN (Virtual Private Network) с многочисленными WEB-сервисами с высоким уровнем надежности, безопасности и средств виртуализации IaaS, PaaS и SaaS Cloud содержит такие виды услуг: testing-as-a-service, ecurity-as-a-service, database-as-a-service, process-as-a-service, application-as-a-service, platform-as -a-service и UC-as-a-service.

Сервисные услуги Cloud Computing в Internet

Услуги охватывает такие вычисления, как сохранение данных, интеграцию, безопасность, организацию процессов, коммуникации и управления. Категории сервис-провайдеров, которые позволяют реализовать пользователю требуемые сервисы в рамках своей архитектуры.

Cloud computing содержит 15 видов услуг для вычислений:

- сохранение данных (storage-as-a-service)
- базы данных (database-as-a-service)
- информации (information-as-a-service)
- процесса (process-as-a-service)
- приложений (application-as-a-service = software-as-a-service)
- платформа (platform-as-a-service)
- интеграции (integration-as-a-service)
- информационные технологиb (IT-as-a-service).

Пользователь может использовать весь спектр услуг у одного провайдера и у других провайдеров.

5.3.6. Система компиляции и сборки ресурсов в Java WWW

Процесс компиляции и сборки ресурсов в API выполняется в JavaE через операцию Make, а Генератор сборочных скриптов в среде GNU Build с помощью Cmake и в MSBuild (.NET), Apache Ant (Java), Apache Maven (Java, C#, Scala), NAnt (.NET), Scons в ЯП (C, C++, Java, Fortran,Tex). ABI - Application Binary Interface включает: набор инструкций процессора: структуру регистрового и файла, организацию стека, способы доступа к памяти и т.п.; размер, расположение, выравнивание базовых ТД, с которыми работает процессор компьютера; соглашение о вызовах (правила передачи аргументов и возвращаемых значений процедур) на стек, регистры процессора; правила задания системных вызовов кодом приложения; бинарный формат объектных файлов, библиотек и исполняемых файлов.

Веб сервисы Интернет задает технологию взаимодействия электронных устройств, программ в сети с помощью Web API интерфейса с помощью средств сервиса:

- SOAP (Simple Object Access Protocol) – протокол обмена структурированной информацией через компьютерные сети, реализованный с помощью веб сервисов XML, RPC.

- WSDL (Web Services Description Language) – язык описания интерфейса веб сервиса SOA, SCA и их функций в XML.

- REST (Representation State Transfer) – архитектурный подход, предоставляющий способ взаимодействия компьютерных систем по сети (Client-Server, Stateless).

Текстовые форматы: XML, JSON, YAML.

К бинарным форматам компьютеров относятся:

– XDR (External Data Representation).

– CDR (Common Data Representation).

– Protocol buffers (gRPC) и др.

Интеллектуализация системных знаний

К современным средствам представления знаний в терминах дескриптивной логики относятся: FaCT (на LISP), FaCT++ (на C++), CEL (2005), KAON-2 (2005, на Java), MSPASS (на C) и Pellet (на Java) и др.

Необходимой составляющей инфраструктурой уровня представления знаний являются прикладные сервисы, ориентированные на коллективное решение проблем среды (Problem-Solving Environments, PSE) мирового сообщества. В процессе создания моделей знаний ПрО используется система KBS (Knowledge-based systems), CommonKADS и др., обеспечивающие построение библиотек знаний, содержащих элементы решения задач в ПрО, ькоторые затем могут повторно использоваться и в других областях знаний.

Программа на современных ЯП – это некоторый формальный или математический текст, который используется системами управления для описания информации в терминологии (System Quirk – SQ, http://www.computing.surrey.ac.uk/SystemQ).

Система поддерживает создание и ведение терминов в терминологическую базу данных (ТБД) и организацию коллекций текстов на компьютере с помощью инструментов Virtual Corpus, KonText, Ferret, Grid и др.

К языкам описания модульных элементов относятся: API, HTML, XML, OWL, JSON и операции работы с этими элементами через link, make, config, weaver, assembling, которые входят в операции общего сборщика технических (макроконвейерных) и конвейерной сборки прикладных систем из готовых ресурсов Интернет в недалекой перспективе.

Глава 6

Онтология – инструмент представления знаний

Онтология - один из инструментов Семантик Веб для представления семантики разных предметных знаниях (математика, физика, биология, медицина и др.)[1, 2].

В основе представления онтологических знаний о некоторой ПрО лежит понятийная база, совокупность концептов (понятий) и отношений между ними, классификация понятий и их таксономия в виде тезаурусов, а также методы создания онтологий прикладных областей знаний.

Онтология в Семантик Веб (www.semanticweb.com/ontology) является инструментом построения концептуальной модели (КМ) предметной области/ домена. КМ включает структуры данных, релевантные классы объектов, их связи и отношения (теоремы, ограничения). Языками описания онтологий являются OWL, FODA, ODM, DSL и др. Проведена онтология домена SE ЖЦ ISO/IEC 12207 -2007 в языке OWL, средствами Protégé 2.3 с выходным результатом в языке XML. «Подход к автоматизации ЖЦ» докладывался на Международной конференции «Science and Information-2015», www.conference.thesai.org, july 28-30. London UK, p.965-972. Комитет IEEE предложил сделать патент на описание средства автоматизации ЖЦ.

Онтология домена ЖЦ ISO/ IEC 12207

Объектом моделирования домена ЖЦ являются основные процессы стандарта ISO/IEC 12207 ЖЦ (требования, проектирование, конструирование, тестирование и сопровождение), процессы поддержки и организации разработки (экспертиза, верификация, тестирование), сбор данных об ошибках и отказах, которые используются для оценки разных показателей качества ПП, в частности надежности.

Построение формальной модели домена выполняется методами онтологии путем аккумуляции знаний о моделях, ресурсах, доменах, которые созданы в некоторой операционной среде.

Аспектом производства программ для домена рассматривается генерация, которая базируется на представлении знаний о специфике ПрО и накопленных знаний, а также на методах, средствах и инструментах SE, которые необходимы для линий автоматизированного изготовления отдельных ресурсов (компонентов) и ПС.

Процесс генерации рассматривается как последовательная трансформация промежуточных продуктов одного процесса ЖЦ в продукты следующего процесса путем использования систем программирования и других систем трансформации предметно ориентированных описаний членов ПС с применением КПИ. Используется следующая таблица процессов и подпроцессов (табл.6.1).

Таблица 6.1.

Процессы ЖЦ

№ п/п	Процесс (подпроцесс)
1. Категория «Основные процессы»	
1.1	Заказ (договор)
1.1.1	Подготовка заказа, выбор поставщика
1.1.2	Мониторинг деятельности поставщика, принятие потребителем
1.2	Поставка (приобретение)
1.3	Разработка
1.3.1	Выявление требований
1.3.2	Анализ требований к системе

№ п/п	Процесс (подпроцесс)
1.3.3	Проектирование архитектуры системы
1.3.4	Анализ требований к системе
1.3.5	Проектирование системы
1.3.6	Конструирование (кодировка) системы
1.3.7	Интеграция системы
1.3.8	Тестирование системы
1.3.9	Системная интеграция
1.3.10	Системное тестирование
1.3.11	Инсталляция системы
1.4	Эксплуатация
1.4.1	Функциональное применение
1.4.2	Поддержка потребителя
1.5	Сопровождение
	2. Категория «Процессы поддержки»
2.1	Документирование
2.2	Управление конфигурацией
2.3	Обеспечение гарантии качества
2.4	Верификация
2.5	Валидация
2.6	Общий обзор
2.7	Аудит
2.8	Решение проблем
2.9	Обеспечение применимости продукта
2.10	Оценивание продукта
	3. Категория «Организационные процессы»
3.1	Управление
3.1.1	Управление на уровне организации
3.1.2	Управление проектом
3.1.3	Управление качеством
3.1.4	Управление риском
3.1.5	Организационное обеспечение
3.1.6	Измерение
3.1.7	Управление знаниями
3.2	Усовершенствование
3.2.1	Внедрение процессов
3.2.2	Оценивание процессов
3.2.3	Усовершенствование процессов

Этот вариант стандарта ЖЦ 2007 (табл.6.2) включает в себя 17 процессов, 74 подпроцесса и 232 технологических операционных задачи (действий).

Таблица 6.2. Процессы, подпроцессы и задачи ЖЦ

Класс	Процесс	Действие	Задача
Основные процессы	5	35	135
Процессы поддержки	8	25	70
Организационные процессы	4	14	27
Всего	17	74	232

Процессы ЖЦ представлены в виде схем для задания по ним онтологического описания.

Как правило, в зависимости от целей конкретного проекта на ПП главный разработчик и менеджер выбирают процессы, действия и задачи, выстраивают определенную схему ЖЦ для применения в конкретном программном проекте.

Описание семантики процессов, парадигм и методов их выполнения (объектные, компонентные, сервисные и др.) приведены в ядре знаний SWEBOK (www.swebok.com). В каждой технологии программирования сложных ПС с использованием стандарта ЖЦ применяются теоретические, прикладные методы, стандарты качества, общие и фундаментальные ТД (ISO/IEC 15404, ISO/IEC 9126, ISO/IEC 11404 GDT и др.), а также методики этих стандартов.

Задача автоматизации стандартного ЖЦ возникла при выполнении фундаментального проекта по ТП (2007—2011) и разработке ИТК. Была постелена цель — автоматизировать ЖЦ и обеспечить генерацию разных его вариантов при изготовлении отдельных ПС из готовых ресурсов. Первый эксперимент по реализации ЖЦ проведен с участием студентов КНУ 4-го курса кафедр ИС, ТТП, МФТИ и двоих аспирантов. Участники разработки изучили современные онтологичные средства и средства визуального представления процессов ЖЦ — WWF (Windows Workflow Foundation), DSL Tools VS.Net, Protégé и др. На основе этих средств было реализовано описание онтологии ЖЦ в графическом и XML видах в рамках систем DSL Tools VS.Net и Protégé. С вариантом онтологии ЖЦ автор настоящего издания выступала на международных конференциях TAAPSD’12—15, ICTERY —2013 и «Science and Information —2015».

6.1. Описание процессов ЖЦ средствами DSL и Protege

Для описания онтологии домена ЖЦ взят Eclipse DSL. В нем имеются средства разработки графических моделей ЖЦ. Описание основных процессов домена ЖЦ с помощью инструментария DSL Tools VS приведен на рис2. Типы отношений в данном графическом представлении задают основную логику процессов домена ЖЦ. В каждом классе заданы методы и поля, необходимые для функционирования.

Процессы поддержки заключают в себе все процессы, которые выполняются после построения системы и поддержки его работоспособности. Их онтологическая структура отражает структуру основных процессов ЖЦ. Затем осуществляется генерация имеющихся графических моделей в текстовое представление XML-языка. Графическое представление процессов ЖЦ дано на рис. 32—34.

Процессы поддержки включают в себе все процессы, которые выполняются после построения модели и проверки ее работоспособности. Онтологическая структура отражает структуру процессов ЖЦ и их представление в XML-языке средства DSL Tools VS.Net (для трех категорий процессов ЖЦ описание в языке XML занимает 15 страниц). «Подход к автоматизации ЖЦ» докладывался на Международной конференции «Science and Information-2015», www.conference.thesai.org, july 28-30. London UK, p.965-972. Комитет IEEE предложил сделать патент на описание средства автоматизации ЖЦ.

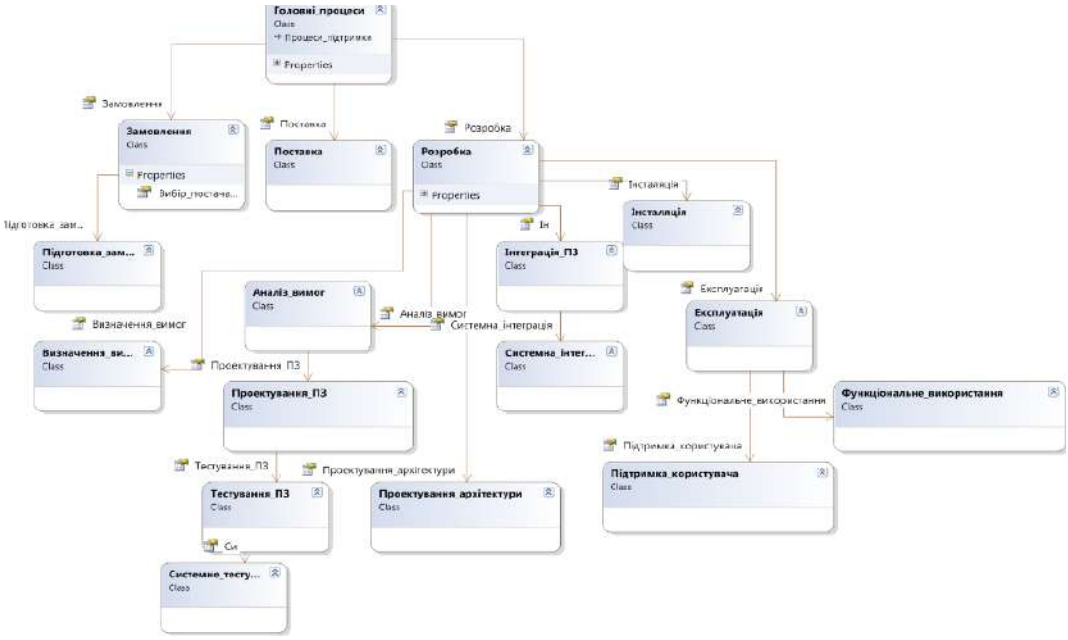


Рис. 32. Графическое представление основных процессов ЖЦ

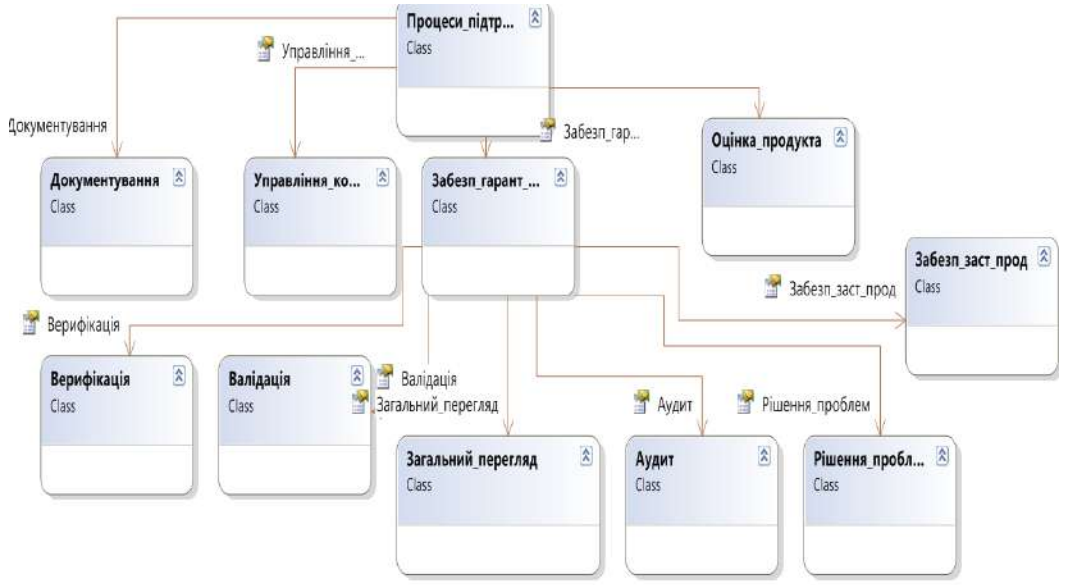


Рис. 33. Графическое представление процессов поддержки ЖЦ

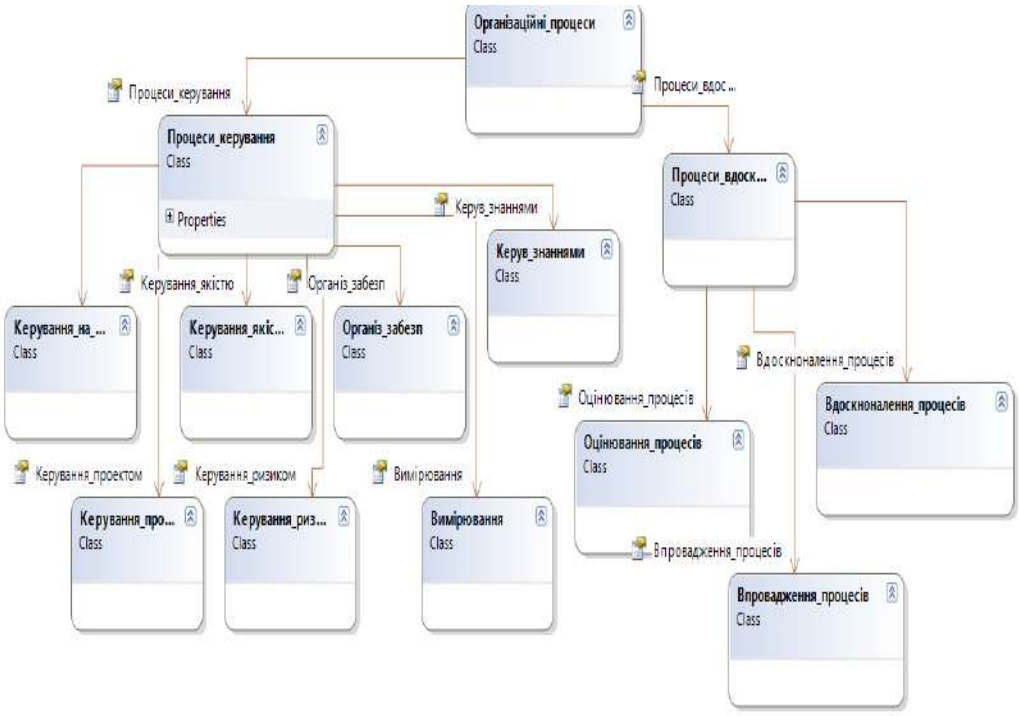


Рис. 34. Графическое представление организационных процессов ЖЦ

Литература

1. Гаврилова и др. Базы знаний интеллектуальных систем // Учебник для вузов. — СПб.: Питер.
2. Федоренко Д.Г., Астраханцев Н.А. Автоматическое извлечение новых концептов предметно специфичных терминов. Труды ИСП РАН, том 25, 2013, стр. 167-178.
3. 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.
4. Ekaterina M. Lavrischeva. Ontology of Domains. Ontological Description Software Engineering Domain—The Standard Life Cycle, Journal of Software Engineering and Applications, 2015, 8, p.1-15. Published Online July 2015 in SciRes.http://www.scirp.org/journal/jsea.

Глава 7

Средства автоматизации программ в Интернет среде (2002-2018)

Технологическая подготовка разработки (ТПР) технологических линий является концепцией усовершенствования процессов для управления проектами СПС; унификация процесса тестирования; Концепция управления качеством; Унификация процесса на Входе и Выходе. Любая фабрика программ работает по ТП или ТЛ, которая создается на этапе технологической подготовки работ) по производству некоторого технического продукта (Лаврищева Е.М. «Основы ТПР прикладных программ СОД». – 1987. - 29с.).

ТЛ комплектуется из процессов, стандартных инструментов и технологических модулей (ТМ) отображения специфики ПрО в заданной среде ОС. Для ТЛ подбираются готовые ресурсы, КПИ, средства и инструменты порождения и реализации функций программ, а также маршрут управления процессами ТЛ с заполнением шаблонов (патернов) фиксации проектных решений, изменения состояний элементов на процессах и оценки продукта на качество.

В мировой практике сформировалась концепции создания фабрик сборки программ (2002-2018)

1. система конвейерной сборки модулей в АПРОП;
2. система сборки Sun Microsystems (IBMShpere - appfab);
3. OMA-архитектура ORB link (Stub, Skeleton);
4. фабрика "ручной» сборки разноязычных программ Инга Бейя;
5. фабрики мультисборки по диаграммам UML Дж. Гринфильда;
6. среда потоковой сборки (appfab) в MS.VSTS;
7. конфигурационная сборки (building, config) в системе Grid;
8. фабрика потоковой сборки К. Ленца на основе use case;
9. каркас фабрики программ Авдошина (ВШЭ).
10. продуктовая линия (Product Lines/Product family) Эзенакера сборки КПИ по моделям систем и вариабельности.
11. Интеграция систем М. Фаулера.
12. ЕПАМ сборка по продуктовой линии из готовых ресурсов.
13. INTEL (make), Grid (config) и др.

В словаре ISO/IEC FDIS 24765:2009 (E) – Systems and Software Engineering Vocabulary дано определение системы (фабрики) выпуска программной продукции на продуктовой линии: «группа продуктов или услуг, которые имеют общее управляемое множество свойств, удовлетворяющих потребностям определенного сегмента рынка продукта или вид деятельности.

Средства Интернет-браузера для проведения сборки относятся: клиент, сервер, Базы данных, СУБД, СП с ЯП (Python, C++, JAVA, и др.), Semantic Beb, Big Data, Cloud Continious и др. Средства клиента: Chrome, Firefox, MS Internet Explorer, Safari, Opera, SOAP, SOA, SCA, SCM ....

Модель клиента Мк ={объекты, данные, функции, отображение данных }. Эти элементы задаются в одном из скриптовых ЯП (JavaScript, VB Script, Java, HTML, XML) браузера и задают:

– работу с массивами, таблицами, списками;

– верификацию данных, которые вводит конечный пользователь;

– обработку нестандартных окон браузера при отображении объектов данных и др.

Средства сервера: Internet Information Server, Apache, JBoss, WebSphere, WebLogic, Cloudscape, Java Enterprise Edition, Crowdsourcing Data Fusion, Massively Parallel Processing... Интерфейс между клиентом и сервером задается в виде:

WebAppInterface={Request}, где Request – запрос.

Сервер приложений JEEE обеспечивает:

- генерацию серверных страниц (Java Server Pages);
- определение КПИ в Enterprise Java Beans;
- запуск серверного приложения в ЯП (Python, Ruby, Java, PHP и т.д.) для проведения вычислений и обработки аварийных событий в современных средах Интернет;
- создание КПИ на Enterprise Java Beans;
- транзакционное преобразование КПИ;
- услуги по обмену сообщениями (Java Message Queue);
- сетевые службы (веб-сервер, сервлеты и др.);
- сервис обмена сообщениями с компонентами JMS (Java Message Service);

Использование сервисов SOA, SCA для передачи сервисных объектов SDO и интерфейсов для компоновки и конфигурирования.

Сервер БД выполняет:

- обслуживание БД с обеспечением целостности, сохранности данных и доступ клиента к информации;
- веб-приложение, работающего с Big Data, выполняет управление, анализ данных большого объема и их манипулирование;
- интерфейс клиент-приложения к программно-аппаратной части для соединения с большими приложениями Интернет.

7.1. Сборка сервисов Интернет в Веб- системы

Для сборки сервисов используется инструменты – Jopera for Eclipse (http://www.jopera.ethz.ch/), который обеспечивает:

- композицию сервисов и визуальный мониторинг отладки композиций сервисов;
- управление изменением интерфейсов сервиса с помощью сообщений об изменениях в сервисе;
- масштабируемость и автономное исполнение процесса запуска конфигурируемых систем с помощью сообщений.
- поиск сервисов по их семантическому описанию (Feta Client и Feta Engine, где Feta Client – это GUI-плагин системы Интернет Taverna для описания сервиса, а Feta Engine для задания Web-сервиса).
- конструирование запросов к сервисам через Feta Engine и отображение информации о результатах выполнения запроса;
- интегрирование сервисов в WorkFlow в соответствии с моделями OSI, SOA, SCA, которые используются для представления и обработки интеллектуальных и информационных ресурсов в сети Интернет.

Технологические процессы сборки КПИ на WEB- среде

1. Разработка графа системы из готовых компонентов, КПИ.
2. Поиск готового КПИ в библиотеках Интернет и хранение в Библиотеки СПБ.
3. Разработка нового КПИ в ЯП (Python, Ruby, Java, Ada, Prolog и др.) иинтерфейсной части, заданной в языке WSDL.
4. Статический анализ КПИ, исправление ошибок, получение кода.
5. Проверка правильности (верификация) КПИ на контрольных примерах и занесение в БСПБ.
6. Проверка интерфейса связываемых КПИ и передачи данных на эквивалентный обмен параметров КПИ в разных ЯП.
7. Сборка по графу КПИ отдельных программ (Р1, Р2) системы и проверка правильности взаимосвязи КПИ с БСПБ.
8. Конфигурирование Р = (Р1, Р2, ..., Рм).
9. Тестирование Р на разных наборах данных.

10. Генерация документов для сопровождения и эксплуатации веб-системы. AVS – ресурсы для автомобильной промышленности.

https://www.osp.ru/os/1996/02/178854
http:// www.osp.ru/IRIS EXPLOER
Вариант № 2
Работа в среде сайта
http://7draguns.ru/ru

Сущность подхода к сборке Веб-систем

Процесс проектирования Веб-систем проводится по методу компонентного программирования ОКМ. Он содержит совокупность программных компонентов нескольких уровней [8]. Каждый из уровней имеет собственную структуру и более детализированные элементы следующих уровней. Компоненты первого уровня играют роль контейнеров. Все процессы ЖЦ web-систем начинаются с формирования и определения архитектуры web-системы в среде Интернет:

- модель web-системы и модель характеристик (Model Feature) для управления вариантами систем;
- клиент-серверную архитектуру с Web-сервером и web-клиентом для выполнения запросов от клиента к серверу из модели web-системы;
- взаимодействие между программными элементами выполняет интерфейс;
- программные элементы – это компонент и сервис Интернет, имеющие уникальное имя, интерфейс и реализацию;
- схема запроса к имени функции элемента системы соответствуют операции вызова удаленных процедур с набором входных и выходных параметров.

Операция вызова включает URL –данные: имя домена Интернет, порт web-сервера, маршрут и имя серверного компонента для обработки запросов от клиента. Список входных параметров определяется множеством CGI-параметров, каждый из которых является парой: имя параметра и его значение. Исходным параметром является HTML-страница, которая отображает результат запроса.

Модель создаваемой системы

Программная система формируется из отдельных программных элементов, которые реализуют объекты ПрО и их интерфейсы связи элементов между собой в заданной среде. Программный элемент, исходя из общей точки зрения – это самостоятельный продукт (код) метода или функции, которая входит в объектную модель, реализующую часть или всю отдельную предметную область и умеет взаимодействовать с другими элементами системы через интерфейсы [9, 10].

Модель системы имеет вид: $M_{ПС} = <L, M_f, M_s, M_i, M_d>$,

$L = (L_1, L_2, \ldots, L_k)$ – язык описания функций домена и каждый язык включает в себя набор операций алгебры действий (actions) по реализации и выполнению соответствующих элементов;

M_f – множество функций модели домена, которая является моделью объектов

$M_f = (O_1, O_2., O_r,)$, каждый объект которой трансформируется к программному коду;

$M_s = (M_{sin}, M_{sout}, M_{sinout})$ – множество сервисов, которое базируется на модели связи и включает в себя модели передачи входных данных M_{sin}, выходных данных M_{sout} и серверных данных M_{sinout} системы на сервере Application, базированных на множестве системных сервисах (Common Facility Services) операционной среды, которая реализуется специальным брокером или монитором сервисов и др.;

M_i – множество интерфейсов в языке IDL с описанием входных *in* и выходных параметров *out* и *inout* КПИ для пары связанных элементов в структуре системе. С

практической точки зрения все общие ТД специфицируются как внешние данные в паспорте модели ПС из объектов домена;

M_d – множество данных и метаданных предметной области системы, которые специфицированы в КПИ, как примитивные или сложные фундаментальные ТД языков программирования, а также в модулях-посредниках (stub, skeleton) в языке IDL.

Множества M_{inc}, M_s и M_i связаны с интерфейсными и общим данными и имеют пересечение по данным, которые отнесены к *in, out, inout* и входят в состав внешних типов данных, которыми обмениваются по сети одни элементы с другими сервера Application. По переданным данным выполняются компоненты и отправляют результаты к серверному компоненту.

Модель сайта системы

Сайт системы – это набор готовых функциональных, системных и сервисных элементов, определяющих сущность и назначение задач сайта. Вход на сайт осуществляет специальная программа Администратор, предоставляя пользователю, заданные на сайте проблемные и системные функции, а также разные материалы и данные программ сайта.

Модель сайта имеет вид: $Mst = <F_f, S_s, S_i, I_{d,} Msys, Md,$ где

$M_f = (f_{O1}, f_{O2}., f_{Or},)$ - – множество функций системы, каждая из которых реализует некоторый метод или задачу предметной области;

$M_{ser} = (S_{si}, S_{sout}, S_{snout})$ – множество сервисов для обработки входных, выходных и промежуточных данных, которые базируются на множестве системных сервисах (Common Facility Services);

Msys - множество системных функций клиента и сервера Веб-сайта;

M_d - множество данных и метаданных предметной области сайта, которые специфицированы в параметрах готовых ресурсов, сложные типы данных (ТД) и используются менеджером информационных объектов клиента и сервера.

Системные элементы включают задание оборудования, Операционной система и Веб-сервера, а также дополнительное программное обеспечение. В состав сайта входят клиент – серверная архитектура или Интернет браузер. На веб-сервере происходит обработка запросов или/и генерация запросов.

7.2. Подход к созданию экспериментального ядра OS Linux

В настоящее время задача создания некоторого варианта ОС для класса прикладных систем (медицины, биологии и др.). 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. Зарубежные специалисты в рамках VAMOS при моделировании изменяемых систем создали модель MF с базовыми характеристическими элементов ОС и модели системы *Msys,* включая множество функциональных M_f,, интерфейсных M_{io} и M_d работы с данными базового ядра ОС. Эти множества компонентов тестируются на правильность их идентификации и выполнения, на наборах тестов и операций связи 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 путем задания следующих параметров ядра [4–8]:

- 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 OS Linux Национального агентства по безопасности NSA, ориентированных на защиту и безопасность функционирования варианта ядра ОС.

Перед проведением конфигурационной сборки требуемого варианта ОС Linux 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, в котором задаются две строки для каждого сетевого адаптера. Первая строка сетевой карты задает описание и комментарии с заданием идентификаторов оборудования и устройств согласно карте PCI и драйверов. При задании имени интерфейса идентификатор аппаратного обеспечения не используются.
Вторая строка — это описание правил Udev, которое соответствует сетевой карте с фактически присвоенным именем.

Некоторые программы ПО ОС 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, входящего в ОС.

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.

С общей точки зрения модель OS Linux включает множество разных операционных артефактов (функций) и интерфейсов между ними [17–19]: $M_{oc} = (C_k, M_f, M_s, M_o, M_i)$, где

C_k — множество отдельных фрагментов, артефактов ОС;

M_s — множество характеристик ядра Linux (более 10 000 для версии 4.11);

M_o — множество ограничений характеристик функций на глубине дерева зависимостей (глубина 8 для 22 ограничений [20]);

M_i — множество интерфейсных характеристик (menuconfig) в ядре Linux.

Модель вариабельности ОС согласно проведенным исследованиям и обсуждениям 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 характеристики. При генерации некоторого варианта ОС требуется извлечь из ядра системы необходимые готовые артефакты или функциональные элементы с их интерфейсами [16-20 37-42]. Вариант ядра OS Linux разработан с участием магистрантов и представлен на конференции OS DAY-2021

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-практическая конференция OS DAY "Российские аппаратные платформы и операционные системы" - Москва 14 октября 2021. Опубликован доклад-статья - Наука и технологии - Системный администратор. Ноябрь, 2021, с.80-89.

Заключение

Конвейерная сборка, сформировавшаяся при реализации задач ВПК СССР (70-80г.XX столетия), начиная со сборки разноязыковых модулей, записанных в ЯП (Algol-60, Fortran, Cobol, PL/1 и др.) и библиотека интерфейса получила дальнейшее развитие. Толком к развитию является появление теории общих типов данных GDT стандарт ISO/IEC 11404 – 2007, 2012 и конкретных огромных новых не структурированных данных в связи с исследованиями небесных, земных и подводных недр. Представлена схема генерации таких типов данных типов данных DT к более простым, отработанных на практике. Приведены Веб- сервисы, клиент-серверная архитектура, системные и сервисные ресурсы Семантик Веб и новые операции сборки make, assembling, config в компиляции и сборки этих ресурсов в более совершенные системы решения сложных задач предметных областей знаний. Приведен общий сборшик ресурсов в Интернет, позволяющий на перспективу производить сложные системы на производственной основе с высокой производительностью и качеством реализации алгоритмов задач для разных предметных областей знаний из ресурсов в современных ЯП (C, C++, Python, Ruby, Java и др.) и математической логики. Дана характеристика новых языков API, ABI, WSDL, SOAP, RDR, XML, OWL и др. для описания и сборки систем в Интернет. Представлен модель сборки Веб-систем и сайтов из готовых ресурсов Интернет и разработан экспериментальный вариант OS Linux для применения в современных предметных областях знаний (медицина, физика, биология, математика).

Литература

1. Лебедев В. Н., Соколов А П. Введение в систему программирования ОС ЕС. М.: Статистика. 1978. с. 187.
2. Дал У., Дигора.9. Хор К. Структурное программирование. М.: Мир. 1975, с. 246.
3. Редько В. Н. Композиции программ и композиционное программирование. — Программирование. 1978, № 5. с. 3—24.
4. Загадки Б. А. Автоматизация реакторных расчетов. М, Атом злат, 1974, с 101.
5. Глушков В. М., Велькицкий И. В. Технология программирования и проблемы ее автоматизации — Уснём. 1976. М. 6. С. 75—93.
6. Modular and structured programming techniques. Data Processing 17(1975), 5. p. 310—316
7. Глушков В.М., Канишиова Ю.В., Легнчевскнй А.А О применении метода формализованных технических заданий к проектированию программ обработки структур данных. — Программирование, 1978, № 6, с. 31—43.
8. Система автоматизации производства программ (АПРОП), Киев, РФАП, 1976. 135с.

9. Гришенко В. Н,, Лаврншсва Е. М. О создании межъязыкового интерфейса для ОС ЕС ЭВМ - УСнМ. 1978. № I. с 34-41.
10. Лавришева Е. М. Вопросы объединения разноязыковых модулей в ОС ЕС ЭВМ. - Программирование, 1978. № I. с. 22-27.
11. Алферова И. А, Лихачева Г. В., Шураков В. В. Математическое обеспечение НИС ЭВМ. - М.: Статистика, 1974.
12. Гальченко О. Н. О сопряжении программ, написанных на языках ПЛ и Фортран. УСнМ, 1979. № 0. с. 69-71.
13. Орлов Б. II Пономарев А. С. Система генерации модульных программ. Кибернетика, 1980, *St* 2. с. 82—84.
14. Браун П. Обзор макропроцессоров. Пер. с англ. М., Статистика. 1975, с 77.
15. Филина Л. Н. Вопросы связи модулей, транслированных с языков Фортран, ПЛ/1. Ассемблер, в ОС ЕС. — Программирование. 1980. М 3. с.39—43.
16. Жоголев Е. А. Принципы построения многоязыковой системы модульного программирования. — Кибернетика, 1974, Л* 4, с. 1—5.
17. Жоголев Е. А. Технологические основы модульного программирования. Программирование. 1980. № 2, с. 44—49.
18. Кахро М. И., Мяннисалу М. А., Саан Ю. П., Тыугу Э. Х. Система программирования ПРИЗ. Программирование, 1976, № I, с 38—46.
19. Ершов А. П. Проектные характеристики многоязыковой системы программирования. Кибернетика. 1975, № 5.
20. Лавришева Е. М. Об автоматизированном изготовлении программных агрегатов из разноязыковых модулей. — УСнМ, 1979, № 5, с 54—60.
21. Лавришева Е. М. Методика модульного изготовления программных агрегатов— Кибернетика, 1980, № 1.
22. 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.
23. 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.
24. 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
25. Е.М.Лаврищева. Компонентная теория и коллекция технологий для разработки приложений из готовых ресурсов//4-Научно-практическая конференция «Актуальные проблемы системной и программной инженерии.-20-24 мая 2015. Сборник научных трудов АПСПИ-2015. с.101-119.
26. E. M. Lavrischeva. Component theory and collection of technologies for application development from ready-made resources/ / 4-Scientific and practical conference "Actual problems of system and software engineering. 20-24 may 2015. Collection of scientific works APSSE-2015, Moscow. pp. 101-119.
27. 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.
28. 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

29. 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.
30. 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.
31. 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.
32. Е.М.Лаврищева, И.Б.Петров. Теория моделирования технических и математических задач предметных областей знаний. Евроазиатское Научное Объединение. 2021.№ 1- (1). с.35-4
33. 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.
34. Лаврищева Е.М. Теория графового моделирования сложных систем для прикладных областей Science»1 часть №28/2019. р.12-30. http://austria-Science.
35. 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
36. Лаврищева Е.М. Теория графового моделирования сложных систем из модульных элементов для прикладных областей Austria-science»1 часть №28/2019. р.12-30. http://austria-science.info
37. E.M. Lavrischeva, A.K.Petrenko, B.A.Pozin. Technology of Assembly of Intellectual and Information Resources Internet.-CEUR-WS.org/vol-2543 ipaper-22.pdf. 27 p.
38. 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», №311 (81), Ноябрь, 2021. с.94-105. DOI: 10.5281/zenodo.5796926.
39. Лаврищева Е.М., Рыжов А.Г. Подход к моделированию систем и сайтов из готовых ресурсов. Научный сервис в сети Интернет: труды XX Всероссий-ской 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.
40. 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, Ноябрь, 2021. с.80-89.
41. Е.М. Лаврищева, И. Б. Петров, А. К. Петренко. Парадигмы моделирования и программирования задач предметных областей знаний. Москва; Берлин: Директ-Медиа, 2021. 496 с.
42. Лаврищева Е.М..Технология разработки и моделирование вариантов программных систем. Москва-Юрайт-2017, ISBN 978-5-534—901-574».
43. Лаврищева Е.М.Программная инженерия и технология программирования сложных систем», 2018. 431с. Юрайт-2018. www.urait.ru, www.biblio- online.ru.
44. Лаврищева Е.М. Программная инженерия. Парадигмы, технология, средства программирования, Юрайт, 2017. 431с. www.urait.ru, www.biblio-online.ru. (копия книги Software Engineering Компьютерных систем. Парадигмы, технологии, CASE-средства программирования, 2013.Киев. Наук.Думка. 431с.).
45. Лаврищева Е.М., Грищенко В.Н. Связь разноязыковых модулей в ОС ЕС. Москва: Финансы и статистика, 1982. 127 с.
46. Доклад–статья Е.М. Лаврищева, Л. Е. Карпов, А. Н. Томилин. Семантические ресурсы для разработки онтологии научной и инженерной предметных областей, Всероссийская

конференция "Научный сервис в сети Интернет" 21-26 сентября 2015, Абрау-Дерсу. – с.193-218. Наука и сервис в Интернет 2015 .PPT-к.

47. Доклад-статья Лаврищева Е.М., Слабоспитская О.А. Технология моделирования изменяемых программных продуктов и систем//XII Межд. Научно-практической конференции «Теоретические и прикладные аспекты построения программных систем».-TAAPSD'2015, 23-26 ноября, 2015.-с.118-128. ЛАВР-Слаб TAAPSD-2015 (8).ppt

ПРИЛОЖЕНИЕ. Используемые сокращения

ОС ЕС	–	операционная система Единой серии
АПРОП	–	Автоматизация производства Программ
ЯП	–	языки программирования
ВЦ	–	вычислительный центр
МИ	–	межъязыковый интерфейс
БМИ	–	библиотека межъязыкового интерфейса
ИВ	–	информационный вектор
СВС	–	словарь внешних символов
ЯУЗ	–	язык управления заданиями
М-С	–	модуль-связка
КМ	–	корневой модуль
БМО	–	библиотека макроопределений
БЗМ	–	библиотека загрузочных модулей
МО	–	макроопределение
ТПЗ	–	таблица подзадач
ТВС	–	таблица внешних символов
ЛБ	–	личная библиотека

Оглавление