УДК 681.3

Е.М. Лаврищева

ИНТЕРФЕЙС В ПРОГРАММИРОВАНИИ

Рассматриваются базовые понятия интерфейса, подходы к обеспечению интерфейса языков программирования (ЯП) и взаимодействия разноязыковых программ и данных. Определены общие проблемы неоднородности ЯП, платформ компьютеров и сред, влияющие на выполнение связей между разноязыковыми программами, сформулированы пути их решения. Изложены стандартные решения ISO/IEC 11404–1996 по обеспечению независимых от ЯП типов данных и стандарты преобразования форматов данных.

1. Определение интерфейса

Общее определение. *Интерфейс* – это связь двух отдельных сущностей. В компьютерной области интерфейс определяется на разных уровнях: от уровня видимых коммуникаций между людьми до аппаратных, программных, пользовательских, языковых и других интерфейсов. Аппаратный интерфейс – это разъемы, конекторы и другие устройства для объединения компонентов в компьютерную систему и обеспечения перемещения информации с одного компьютера в другой. На программном уровне интерфейс между программами и ОС, между ОС и аппаратурой обеспечивает передачу и преобразование входных/выходных данных взаимодействующих программ в ОС или во время объединения компьютера с периферийным оборудованием. Пользовательский интерфейс включает средства взаимодействия пользователя с некоторой программой через графический дизайн, панели выбора меню, подсказки и др. В ЯП интерфейс – типы данных и описание констант, переменных, параметров и сложных структур данных, которые образуют межъязыковой интерфейс ЯП как способ эквивалентных взаимосвязей.

В практическом программировании интерфейс представляется набором операций, обеспечивающих определение видов услуг и способов их получения от программного объекта/компонента, предоставляющего эти услуги. На начальном этапе практического программирования в роли *интерфейса* выступали операторы обращения программы к ее процедурам и функциям через формальные параметры, которые записывались в одном ЯП. Операторы обращения включают имена вызываемых объектов (процедур и функций), список фактических параметров, задающих значения формальным параметрам, и получаемых результатов. Выполнение функции или подпрограммы в рамках программы на одном ЯП не вызывало проблем, так как соответствие типов данных устанавливается автоматически.

Когда один из элементов – программа, процедура или функция записаны на разных ЯП и, кроме того, если они располагаются на разных компьютерах, то возникают проблемы неоднородности представления типов данных параметров в этих ЯП, структурах памяти платформ компьютеров и операционных сред выполнения программных объектов. *Понятие интерфейса как посредника* между двумя взаимодействующими программами, сформировалось в связи со сборкой или объединением разноязыковых программ и модулей в монолитную систему на ЕС ЭВМ, память которых позволяла получать большие размеры программ (до 100–200 тыс. команд).

Основу *модуля–посредника* составляет оператор связи между вызываемым и вызывающим разноязыковыми модулями в языках Алгол, Фортран, Кобол, Ассемблер в системе АПРОП (1976–985) [1, 2], первой в СССР фабрики программ, идею которой предложил В.М. Глушков. Интерфейсный модуль–посредник включал описание формальных и фактических параметров, операторы вызова и проверки соответствия передаваемых параметров (по количеству и порядку расположения), а также эквивалентности типов

передаваемых данных. Если типы данных параметров оказывались не релевантными (например, передается целое, а результат функции – вещественное или наоборот), то осуществлялось прямое и обратное преобразование передаваемых типов данных с учетом ЯП и структуры памяти компьютеров. На рис.1 показана общая схема интерфейса программы С, содержащая два вызова – Call A (…) и Call B (…) с параметрами, которые через интерфейсные модули–посредники А'и В' осуществляет преобразование данных и их передачу модулям А и В. После выполнения модулей А и В их результаты преобразуются обратно к виду программы С в тех же модулях А'и В' .

нию через уровень представления данных, устанавливая интерфейс с системой кодирования (перекодирования) данных к виду заданных компьютеров. Это направление остается важным, особенно когда появляются новые ЯП.

1.1. Интерфейс и объектно-ориентированное программирование (ООП)

Интерфейс получил новое развитие в ООП. В нем главным элементом является класс, включающий множество объектов с одинаковыми свойствами, операциями и отношениями. Класс имеет внутреннее (реализацию) и внешнее представление – интерфейс (рис. 2).

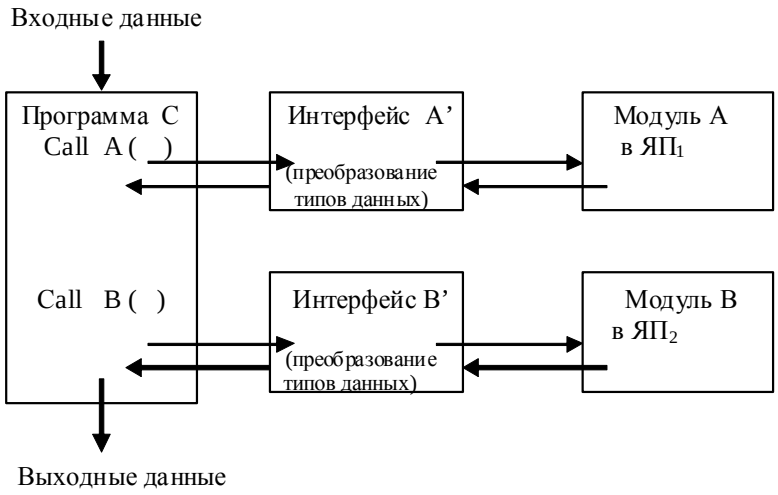

Рис.1. Схема вызова модулей А и В из программы С через модулей–посредников А'и В'

Проблема интерфейса обсуждалась на международной конференции «Интерфейс СЭВ» (1987), на которой было признано, что интерфейс является стратегическим направлением в решении задач связывания программ и систем, записанных в разных языках четвертого поколения на машинах ЕС.

Общая идея интерфейса отображена в стандарте открытых систем (Open Systems Interconnection – OSI) [3]. В нем комбинируются принципы управления архитектурой и посредством программирования, обозначающие пути интеграции разных систем для их взаимодействия и общения. Согласно модели OSI приложение передает запросы другому приложе-

Интерфейс содержит множество операций, описывающих его поведение. Класс может поддерживать несколько интерфейсов, каждый из которых содержит операции и сигналы, которые используются для задания услуг класса или программного компонента. Интерфейс определяет сигнатуру множества операций и/или результирующие действия. Если интерфейс реализуется с помощью класса, то он наследует все его операции. Одни и те же операции могут появляться в различных интерфейсах. Если их сигнатуры совпадают, то они задают одну и туже операцию, соответствующую поведению системы. Класс может реализовывать другой класс через интерфейс [4].

.

Класс	
Внешнее представление	Внутреннее представление
Интерфейсные операции: – публичные, доступные всем клиентам, – защищенные, доступные классу и подклассу – приватные, доступные классу	**Реализация** операций класса и определение поведения

Рис. 2. Структура представления класса с интерфейсом

Операции и сигналы могут быть связаны отношениями обобщения. Интерфейс–потомок включает в себя все операции и сигналы своих предков и может добавлять собственные путем наследования всех операций прямого предка, т.е. его реализация рассматривается как наследование поведения.

Новое толкование интерфейса объектов дано в работе П. Вагнера [5], который сформулировал парадигму перехода от алгоритмов вычислений к *взаимодействию объектов*. Суть этой парадигмы заключалась в том, что вычисление и взаимодействие объектов рассматривались как две ортогональные концепции. Взаимодействие – это некоторое действие (action), но не вычисление, а сообщение – не алгоритм, а действие, ответ на которое зависит от последовательности операций (Op), влияющих на состоянии разделенной (shared state, ss) памяти локальной программы (рис. 3). Операции интерфейса (Op1 и Op2) относятся к классу неалгоритмических и обеспечивают взаимодействие объектов через сообщение.

Модель взаимодействия – это обобщение машины Тьюринга, как распределенной интерактивной модели взаимодействия объектов с входными (input) и выходными (output) действиями (actions) и возможностью продвижения в ней потенциально бесконечного входного потока (запросов, пакетов) в заданном интервале времени.

Дальнейшим развитием идеи взаимодействия, основанного на действиях, является язык AL (Action language), основанный на задании вызовов процедур (локальных или распределенных) и развертки каждого вызова в программу [6, 7], состоящую из операторов действий. Программа из вызовов процедур рассматривается в AL как ограниченное множество конечных программ, взаимодействующих со средой, в которую они погружаются.

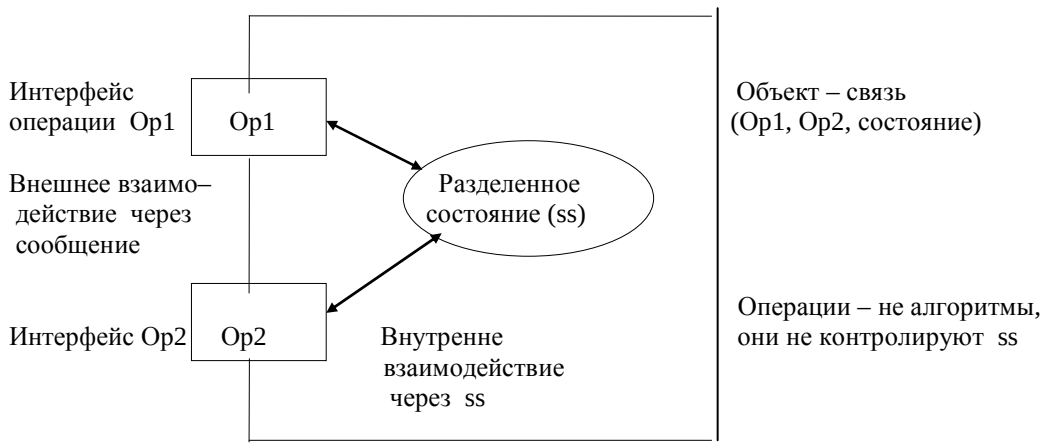

Рис. 3. Интерфейс взаимодействия через операции Op1, Op2

1.2. Интерфейс в современных средах и сетях

Появление разных ПК и их объединение в локальные и глобальные сети привело к *уточнению понятия интерфейса*, в виде удаленного вызова программ, расположенных в разных узлах сети или среды и получающих входные данные из сообщений.

Сети базируются на стандартной семиуровневой модели открытых систем OSI (Open Systems Interconnection) [3]. Объекты уровней этой модели связываются между собой по горизонтали и вертикали. Запросы от приложений поступают на уровень представления данных для их кодирования (перекодирования) к виду используемой в приложении платформы. Открытые системы предоставляют разным приложениям разного рода услуги: управление удаленными объектами, обслуживание очередей и запросов, обработка интерфейсов и т.п. Доступ к услугам осуществляется с помощью таких механизмов:

– вызова удаленных процедур RPC (Remote Procedure Call) в системах ONC SUN, OSF DSE [8];

– связывания распределенных объектов и документов в системе DCOM [9];

– языка описания интерфейса IDL (Interface Definition Language) и брокера объектных запросов – ORB (Object Request Broker) в системе CORBA [10, 11];

– вызова RMI (Remote Methods Invocation) в системе JAVA [12] и др.

Удаленный вызов RPC задает интерфейс к удаленным программам сети в языках высокого или низкого уровней. Язык высокого уровня определяет в RPC–вызове параметры удаленной процедуры, значения которых передаются ей сетевым сообщением. Язык низкого уровня позволяет указать более подробную информацию удаленной процедуре, например, тип протокола, размер буфера данных и т.п.

Взаимосвязь процесса с удаленно расположенным другим процессом (например, сервером) на другом компьютере выполняет сетевой протокол UDP или TCP/IP для передачи параметров в stub–интерфейсе клиента stub–интерфейсе сервера для выполнения удаленной процедуры.

Механизм посылки запроса в системе CORBA базируется на описании запроса в языке IDL для доступа к удаленному методу/функции через сетевой протокол IIOP или GIOP. Брокер ORB передает запрос генератору, посылает результат stub / skeleton серверу, выполняющему интерфейс средствами объектного сервиса (Common Object Services) или общими средствами (Common Facilities). Так как брокер реализован в разных распределенных системах: CORBA, COM, SOM, Nextstep и др., то он обеспечивает интерфейс (взаимодействие) объектов в разных сетевых средах.

Вызов метода RMI в системе JAVA выполняет виртуальная машина (virtual machine), которая интерпретирует byte–коды программ, созданные разными системами программирования ЯП (JAVA, Pascal, C++) на компьютерах. Функции RMI аналогичны брокеру ORB.

1.3. Интерфейс IDL как способ взаимосвязи клиента и сервера

В распределенной среде *интерфейс* реализуется двумя способами: на уровне ЯП через вызовы удаленных программ и интерфейсы в IDL, которые генерируются компиляторами для клиентских и серверных stab's. Интерфейс – это описание клиентских и серверных stab в языке IDL, выполняющих связь объекта–клиента с объектом-сервером и обратно. Интерфейсы имеют отдельную реализацию и доступны разноязыковым программам. Компиляторы с IDL как часть промежуточного слоя реализуют связывание с ЯП через интерфейсы программ клиента и сервера, заданные в этом ЯП [10].

Интерфейсные программы в IDL или в API включают в себя описание формальных и фактических параметров программ, их типов и порядка задания операций передачи параметров и результатов для взаимодействующих программ. Это описание есть ничто иное, как *интерфейсный посредник* (*stub* для клиента и *skeleton*

для сервера) двух разноязыковых программ (аналогично, как на рис.1), которые взаимодействуют друг с другом через механизм вызова двух типов программ (клиента и сервера), выполняемых на разных процессах. Их описания отображаются в те ЯП, в которых представлены соответствующие им объекты или компоненты.

В функции интерфейсного посредника клиента входят:

– подготовка внешних параметров клиента для обращения к сервису сервера,

– посылка параметров серверу и его запуск в целях получения результата или сведений об ошибке.

Общие функции интерфейсного посредника сервера состоят в следующем:

– получение сообщения от клиента, запуск удаленной процедуры, вычисление результата и подготовка (кодирование или перекодирование) данных в формате клиента;

– возврат результата клиенту через параметры сообщения и уничтожение удаленной процедуры и др.

Описание интерфейсного посредника в языке IDL не зависит от ЯП взаимодействующих программ и в целом одинаково для всех вызывающих и вызываемых программ или объектов. Типы параметров в программе клиента в ЯП (C++, Pascal и т.п.) передаются взаимодействующим процессам, ими могут быть: in – входной параметр, out – выходной параметр, inout – совместный параметр.

Описание интерфейсного посредника в IDL начинается со слова interface**,** за которым следует описание типов данных (integer, boolean, string, float, char и др.). Операции интерфейса включают в себя: наименование операции, список параметров, типы аргументов и результатов, а также управляющий параметр для случая возникновения исключительной ситуации и др. Это описание может наследоваться другим объектом класса, становясь базовым, и выполняется в среде CORBA, которая генерирует *stub* и *skeleton* для взаимодействия программ клиента и сервера.

2. Интерфейс и взаимосвязь ЯП

2.1. Формализация интерфейса ЯП. Основные ЯП, используемые для описания компонентов в современных средах, – это C++, Паскаль, JAVA и др. [1, 2, 10, 13].

Разноязыковые программы в ЯП обращаются друг к другу через удаленный вызов, являющийся *интерфейсом* этих программ. Он устанавливает взаимно однозначное соответствие между заданными фактическими параметрами $V=\{v^1,v^2,..,v^\kappa\}$ вызывающей программы и формальными параметрами $F=\{f^1, f^2, ..., f^{\kappa 1}\}$ вызываемой программы. При неоднородности одного из параметров из множества формальных или фактических параметров разноязыковых программ необходимо провести отображение (mapping) неэквивалентного типа данных параметра в одном ЯП в соответствующий тип данных в другом ЯП.

Аналогично решается задача преобразования неэквивалентных типов данных ЯП. Представим ее так.

Этап 1. Построение операций преобразования типов данных $T_\alpha = \{T_\alpha^t\}$ во множестве языков $L = \{l_\alpha\}_{\alpha=1,n}$.

Этап 2. Построение отображения неэквивалентных простых типов данных каждой пары $l_{\alpha 1}$ и $l_{\alpha 2}$. В случае отображения неэквивалентных сложных структур данных в этих ЯП применяются операции селектора S или конструктора C.

Для преобразования типов данных ЯП для каждого типа строится алгебраическая система такого вида: $G_\alpha^t = <X_\alpha^t, \Omega_\alpha^t>$, где $t \in T_\alpha^t$ – множество типов данных, X_α^t – множество значений, которые могут принимать переменные этого t типа данных, Ω_α^t – множество операций над этим типом данных.

Современные ЯП имеют следующие простые типы данных $t = b$ (*bool*), c (*char*), i (*int*), r (*real*), а также *с*ложные типы данных $t =$ a (*array*), z (*record*), u (*union*), e (*enum*), как комбинации простых типов данных. Простые и сложные типы данных представим следующими двумя классами алгебраических систем:

$\Sigma_1 = \{ G_\alpha^b , G_\alpha^c , G_\alpha^i , G_\alpha^r \},$

$\Sigma_2 = \{ G_\alpha^a , G_\alpha^z G_\alpha^u , G_\alpha^e ... \}.$

Каждый элемент класса – это алгебраическая система $G_\alpha^t = <X_\alpha^t , \Omega_\alpha^t >$, заданная на множестве значений типов данных (простых или сложных) и операций над ними. Преобразование t типа данных будем проводить путем изоморфного отображения двух алгебраических систем с совместимыми типами данных $l_{\alpha 1}$ и $l_{\alpha 2}$ языков, которые обладают следующими свойствами:

1) системы G_α^t и G_β^q для языков l_α и l_b являются изоморфными, если типы данных q, t определены на множестве простых или сложных типов данных.

2) между значениями X_α^t и X_β^q типов данных t, q существует изоморфизм, если множества операций Ω_α^t и Ω_β^q для этих типов данных различны. Изоморфизм двух систем $G_\alpha^{t'} = < X_\alpha^t , \Omega >$ и $G_\beta^{q'} = < X_\beta^q , \Omega >$ имеет место, если множество операций $\Omega = \Omega_\alpha^t \cap \Omega_\beta^q$ не пусто. Между множествами X_α^t и X_β^q не существует изоморфного соответствия в случае, если тип данных t и q неэквивалентны. Например, t является строкой цифр, а тип q – вещественное, тогда строится функция прямого и обратного преобразования.

3) алгебраические системы $|G_\alpha^t| = |G_\beta^q|$ равны по мощности, если они представлены на множестве типов данных языков l_α и l_b.

Отображение со свойствами 1), 2) сохраняет линейный порядок элементов, поскольку элементы множества алгебраических систем линейно упорядочены.

2.2. Общая схема связи ЯП через интерфейс. В виду неоднородности ЯП, как в смысле представления в них типов данных, так и их реализации системами программирования на разных платформах компьютеров, интерфейс имеет следующие особенности:

– разные двоичные представления результатов работы компиляторов для одного и того же ЯП, реализованных на разных компьютерах;

– двунаправленность связей между ЯП и их зависимость от среды и платформы;

– отображение параметров вызовов в операции методов;

– реализация ссылками на указатели связей ЯП в компиляторах;

– связь между типами данных ЯП осуществляется через интерфейсы для каждой пары из множества языков (L_1, …, L_n) (рис. 3).

Как видно из рис. 3, связь между различными языками L_1,…, L_n, осуществляется через интерфейс каждой пары языков L_i, L_n, который генерирует соответствующие конструкции языка L_i в язык L_n и наоборот. Она устанавливается с учетом механизмов класса алгебраических систем и функций преобразования неэквивалентных типов данных ЯП этих систем.

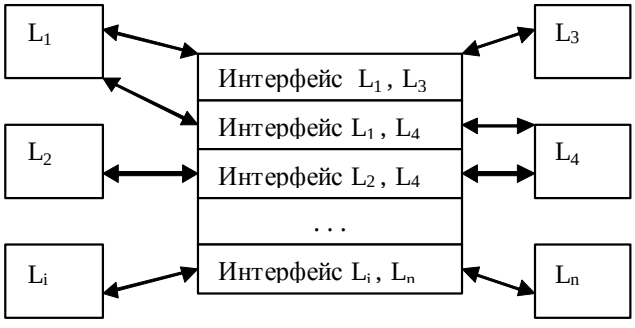

Рис.3. Связь между L_1, L_2, …, L_n через интерфейсы для каждой пары языков

2.3. Независимые от ЯП типы данных стандарта ISO/IEC 11404–1996

Одним из способов решения проблемы интерфейса ЯП является стандарт [14], который рекомендует определение *интерфейса для независимых от ЯП типов данных* средствами стандартного языка LI (Language Independent). Он объединяет существующие типы данных ЯП, расширяет их новыми типами и структурами, предоставляет механизмы генерации новых типов данных и преобразования типов данных ЯП в LI-язык и наоборот. Стандарт задает правила и операции генерации примитивных типов данных и объединений LI-языка в более простые структуры данных ЯП, а также подходы к определению интерфейса ЯП с помощью языков IDL, RPC и API. Типы данных LI-языка разделены на примитивные, агрегатные, сгенерированные (рис. 4). Кроме того, LI-язык включает семейство данных и механизмы генерации типов данных.

Типы данных в стандарте описываются в LI-языке, который претендует на статус более общего языка описания типов данных ЯП, содержит все типы данных существующих ЯП и собственные типы данных, с помощью которых генерируются новые отсутствующие типы данных. Он позволяет описывать параметры вызова в интерфейсе для обращения к стандартным сервисам и готовым программам. В стандарте содержатся средства объявления разных типов данных, которые включены в LI-язык. Кроме того, в нем представлены 33 проблемы, которые решались при создании данного языка.

В [21, 22] LI-языке могут описываться типы данных и выполняться следующие виды преобразований:

– внешнее преобразование – типы данных ЯП в LI-тип данных;

– внутреннее преобразование – из LI-типа данных в тип данных ЯП;

– обратное преобразование.

Внешнее преобразование типов данных и генераторов типов данных состоит в следующем:

а) связывание сгенерированного внешнего типа данных с LI-примитивным типом данных;

в) установление связи между допустимым значением типа данных ЯП и эквивалентным LI-типом данных;

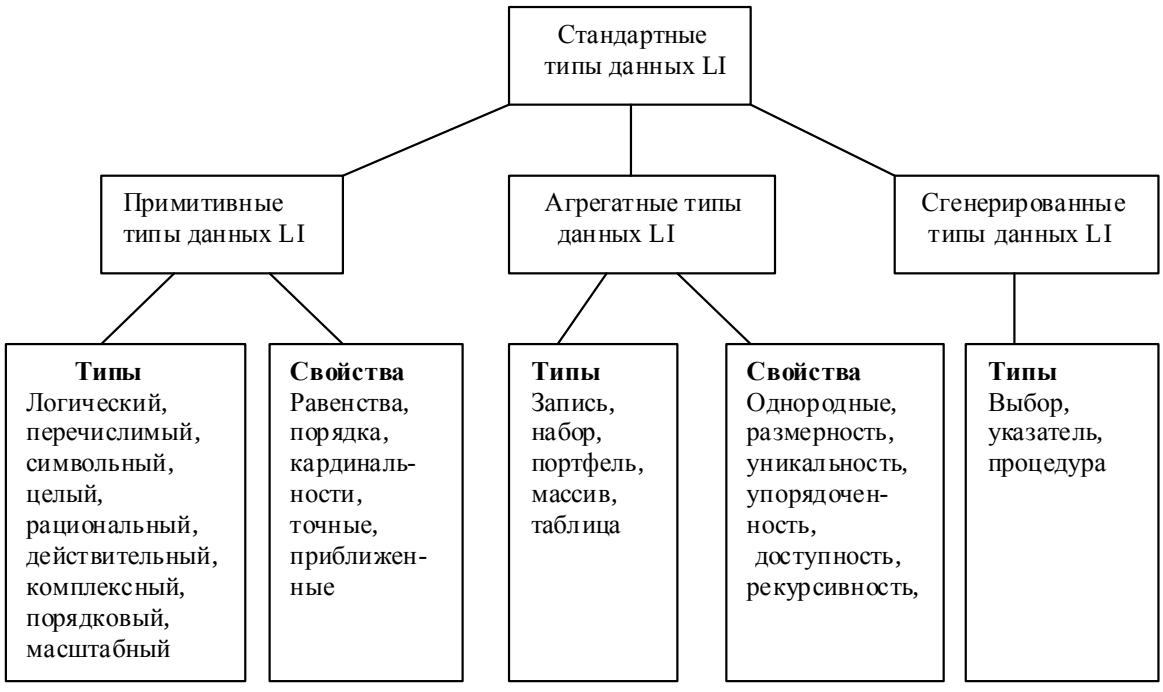

Рис. 4. Независимые от ЯП типы данных стандарта ISO/IEC 11404–1996

с) определение значения внутреннего типа данных ЯП, участвующего в преобразовании, к LI-типу данных.

Внутреннее преобразование обеспечивает связь примитивного типа данных или сгенерированного в LI-тип данных с внутренним типом данных ЯП. Представители LI-типов данных могут преобразовываться в различные внутренние типы данных ЯП при условии, если для каждого LI-типа данных (примитивного или сгенерированного) имеется соответствующий тип данных в ЯП или установлено отношение между допустимым значением его типа и эквивалентным значением соответствующего внутреннего типа ЯП. Для внутреннего типа данных ЯП стандарт определяет способ его преобразования в соответствующий LI-тип данных.

Обратное внутреннее преобразование – это преобразование значений внутреннего LI-типа данных в тип данных ЯП и соответственно обратное преобразование типа данных ЯП в LI-тип данных.

3. Взаимодействие разноязыковых программ в современных средах

3.1. Интерфейс как средство взаимодействия объектов. *Интерфейс – основа взаимодействия* объектов в среде CORBA. Он включает удаленный запрос клиента на выполнение метода (функции, сервиса, операции) или программы [11, 15–17]. Для обеспечения взаимодействия ЯП осуществляется отображение типов данных в типы данных клиентских и серверных стабов путем:

– отображения запроса клиента в ЯП в операции IDL;

– преобразования операций IDL в конструкции ЯП и передачу их серверу брокером ORB для реализации stub клиента.

Так как ЯП системы CORBA могут быть реализованы на разных платформах и в разных средах, то их двоичное представление зависит от конкретной аппаратной платформы. Для всех ЯП системы CORBA (C++, JAVA, Smalltalk, Visual C++, COBOL, Ada–95) предусмотрен общий механизм связи и расположения параметров методов объектов в промежуточном слое. Связь между объектными моделями каждого ЯП системы COM и JAVA выполняет брокер ORB (рис. 5).

Если в общую объектную модель CORBA входит объектная модель COM, то в ней типы данных определяются статически, а конструирование сложных типов данных осуществляется только для массивов и записей. Методы объектов используются в двоичном представлении, допускается соответствие машинного кода объекта, созданного в одной среде разработки, коду другой среды, а также совместимость разных ЯП за счет свойства отделения интерфейсов объектов от реализаций.

В случае вхождения в состав модели CORBA объектной модели JAVA/ RMI, вызов удаленного метода объекта осуществляется посредством ссылок на

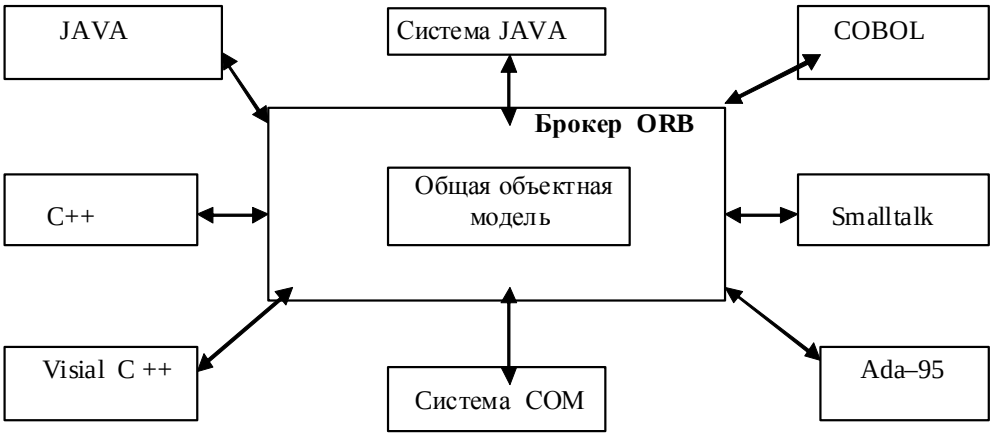

Рис. 5. Интегрированная среда систем CORBA, JAVA и COM

объекты, задаваемые указателями на соответствующие адреса памяти.

Интерфейс как объектный тип реализуется классами и предоставляет удаленный доступ к нему сервера.

Компилятор JAVA создает байт–коды, которые интерпретируются виртуальной машиной, обеспечивающей переносимость байт–кодов и однородность представления данных на всех платформах среды CORBA.

3.2. Взаимодействие разноязыковых программ. Проблема взаимодействия программ между собой постоянно возникает в программировании, как только появляются новые ЯП и системы их поддержки. Механизм связи разноязыковых программ при этом решается каждый раз индивидуально с учетом новых возможностей реализаций ЯП. Так в руководстве программиста [18] автором проделана огромная работа по исследованию проблемы взаимодействия разноязыковых программ в классе современных ЯП (C/C++, Visual C++, Visual Basic, Matlab, Smalltalk, Lava, LabView, Perl), широко используемых в практике программирования. В книге объемом 868 страниц представлено множество различных вариантов конкретных примеров связей каждой пары ЯП из класса ЯП. Эти варианты практически реализованы и проверены на практике. Они включают функции преобразования, методы обращения программы на одном языке к программе на другом языке и средства поддержки интерфейсов (IDL, API и др.).

В таблице сведены проанализированные варианты взаимосвязи в классе приведенных ЯП и дана характеристика особенностей их взаимодействия через разные виды интерфейсов.

В приведенной таблице отображено более 25 видов пар современных ЯП и соответственно прямого и обратного взаимо-

Таблица. Интерфейс в классе современных языков программирования

Средства описания Программ	Языки для взаимодействия	Виды интерфейсов
Visual Basic	– ANCI C – C, C++ – Windows API – DLL – Visual Basic 6.0 – Win 32 –API Viewer	Платформенно–ориентированные функции Программный интерфейс Динамическая библиотека функций Интерфейс между Visual Basic Функции обработки событий Интерфейс в API
Matlab	– C, C++ – Matlab Engine – Mat lab в JNI – Visual Basic 6.0 – Java	Вызов приложения из среды Встраивание функций в VC++ Использование интерфейса JNI Функции из Matlab Функции в Java
Smalltalk	– C++ – Matlab – Start V1	Модель приложения в VisualWorks Функции графической библиотеки Библиотеки C, C++ и процедуры VisualWorks
Lab View	– ANCI C – Visual C++ – Visual Basic 6.0 – C, C++	Интерфейс VI и API Связь Visual C, DLL, Obj Lib c, C++ Интерфейсные функции драйвера
JAVA	– C, C++ – Visual C++ – Matlab	Платформенно–ориентированные функции Библиотеки функций в C++, C Функции в JNI
Perl	– C, C++ – API – Visual C++	Платформенно–ориентированная функции Программный интерфейс Интерфейсные функции в C++

действия разноязыковых программ. Для этих пар ЯП изложены принципы запуска разных программ и все технические вопросы передачи данных и преобразования параметров. Материал книги содержит многочисленные примеры интерфейсных программ, которые разработаны для преобразования разнотипных параметров с учетом особенностей их реализации системами программирования.

В отличие от рассмотренной общей схемы взаимодействия программ с двумя модулями (рис.1), здесь предложены высокотехничные средства обеспечения процесса преобразования: панели, сценарии, иконки и образцы интерфейсных программ для каждого конкретного случая взаимодействия программ.

Далее дается краткое описание основных схем средств описания разноязыковых программ (наименование средств – первая колонка), взаимодействующих с языками описания разноязыковых программ второй колонки таблицы.

Интерфейс между Visual Basic и другими ЯП осуществляется с помощью оператора обращения, параметрами которого могут быть строки, значения, массивы и другие типы данных. Их обработка проводится с помощью функций Windows API, API DLL и операций преобразования типов данных. Проверка интерфейса проведена для схемы обработки Интернет-приложений, задаваемых HTML-страницами BasicVisual и размещаемых в Web-браузере и БД.

Система Matlab содержит средства для решения задач линейной и нелинейной алгебры, действий над матрицами и др. и обеспечивает математические вычисления с помощью MatlabCompiler, Matlab C++, MatlabLibriary, Matlab Graphic Library. Схема независимого приложения в среде Matlab включает интерфейс между VC и Matlab, создаваемый MatlabCompiler путем преобразования программы в формате Matlab (M- файлы или M-функции) в формат C. Сформированный файл вызывается из программы на C++ и преобразуется к виду архитектуры компьютера, куда посылается результат.

Базовые средства Smalltalk обеспечивают создание приложений в среде VisualWorks и включают: модель приложений, методы объектов, сообщения для передачи значений внешним объектам и пользовательский интерфейс (рис. 6). Модель приложения содержит функции DLL из класса внешнего интерфейса, элементы которого применяются для взаимодействия с функциями преобразования библиотеки C++. В эту библиотеку помещаются результаты обработки атрибутов параметров объектов модели домена.

Система LabView предназначена для автоматизации производственных процессов, сбора данных, проведения измерений и управление созданием программ, взаимодействующих с аппаратурой.

В ее состав входит прикладные средства, тестирования программ и драйверы взаимодействия с аппаратурой, запускаемых с пульта. Система взаимодействует с ANS C, Visual Basic, Visual C++ Lab Windows/CV. Эти средства расширяют возможности создания систем

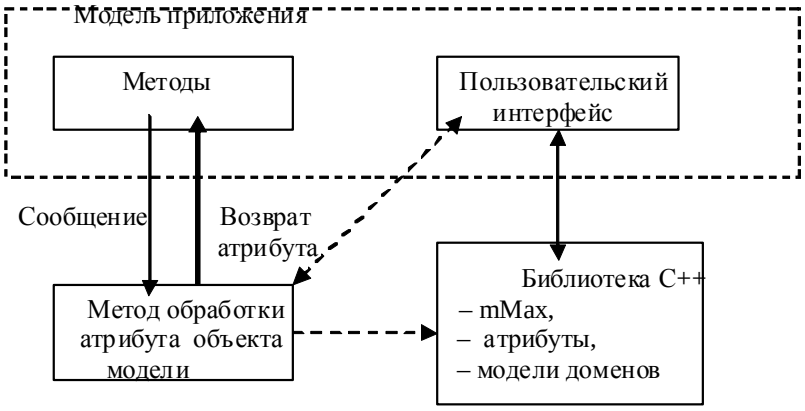

Рис. 6. Схема взаимодействия модели приложения с библиотекой

реального времени, которые позволяют производить измерение аппаратуры типа: регуляторы, термометры, переключатели и др. Результаты измерений могут передаваться по сети.

Среда Java содержит инструменты взаимодействия со всеми языками, приведенными во второй колонке таблицы. Общая схема связи языков JAVA и C, C++ при взаимодействии соответствующих программ показана на рис. 7.

Язык Perl появился в 80–х годах прошлого столетия как язык задания сценариев для взаимодействия с Интернет, управления задачами и создания CGI – сценариев на сервере в системе Unix. Данный язык имеет интрфейс с C.C++б, Visual Basic и Java. Интерпретатор с языка Perl написан в языке C и каждый интерфейс с другим языком рассматривается как расширение, представляемое процедурами динамической библиотеки. Оператор вызова программы в C или C++ преобразуется в специальный код, который также размещается в библиотеке интерпретатора Perl. Сам интерпретатор может быть включен в Win 32 или в программу на C/C++.

Рассмотренные схемы взаимодействия современных средств и класса ЯП для представления разноязыковых программ рекомендуются для практического применения, так как они экспериментально проверены на многочисленных примерах.

4. Интерфейс платформенного преобразования данных

Программы, расположенные на разных типах компьютеров сети, передают друг другу данные через *интерфейсные протоколы*. Переданные данные в формате одного компьютера преобразуется к формату данных (так называемая процедура маршалинга данных) принимающей серверной платформы с учетом порядка и стратегии выравнивания, принятой на ней. Демаршалинг данных обеспечивает обратное преобразование полученного на сервере результата к виду клиентской программы. Если среди передаваемых параметров в операторе вызова оказались нерелевантные типы данных, то осуществляется прямое и обратное их преобразование средствами ЯП или стандарта [14].

К средствам преобразования данных и их форматов относятся:

– стандарты кодировки данных (XDR – eXternal Data Representation [8], CDR – Common Representation Data [10, 11], NDR – Net Data Representation [19], XML [20]) и методы их преобразования;

– механизмы обращения компонентов друг к другу;

– языки описания интерфейсов компонентов – RPC, IDL и RMI для передачи данных между разными компонентами [21, 22].

На каждой платформе компьютера используются соглашения о кодировке символов, о форматах целых чисел и чисел с плавающей точкой (например, IEEE, VAX и др.). При представлении целых типов, как правило, используется дополнительный код, а для типов float и double – стандарт ANSI / IEEE и др. [21].

Порядок расположения байтов зависит от структуры платформы (Big Endian или Little Endian) и устанавлива-

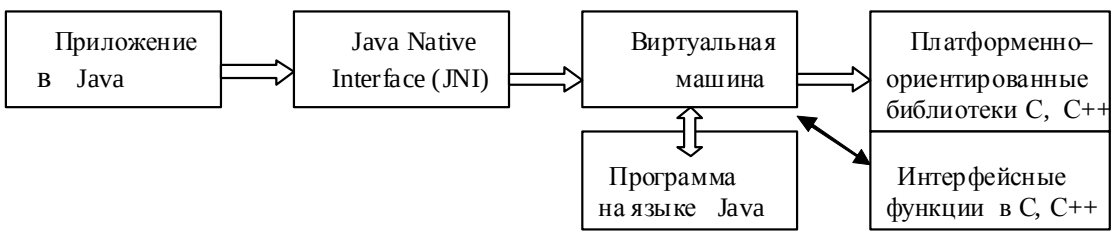

Рис. 7. Схема взаимодействия приложения и программы Java, C и C++

ется от старшего к младшему байту или от младшего байта к старшему. Процессоры UltraSPARC и PowerPC поддерживают обе возможности. При передаче данных с одной платформы на другую учитывается возможное несовпадение порядка байтов. Маршаллинг данных поддерживается несколькими стандартами, которые рассмотрены далее.

XDR–стандарт содержит язык описания структур данных произвольной сложности и средства преобразования данных, передаваемых на платформы (Sun, VAX, IBM и др.). Программы в любом ЯП могут использовать данные в XDR–формате с учетом того, что компиляторы выравнивают их в памяти машины по–разному.

В XDR–стандарте целые числа с порядком "от младшего" приводятся к порядку байтов «от старшего» и обратно. Преобразование данных – кодирование (code) или декодирование (decode) – данных выполняется с помощью XDR–процедур. Кодирование – это преобразование из локального представления в XDR–представление и запись в XDR–блок. Декодирование – это чтение данных из XDR–блока и преобразование в локальное представление заданной платформы.

Выравнивание данных состоит в размещении значений базовых типов с адреса, кратного (2, 4, 8, 16) по границе с наибольшей длиною (например, 16). Системные процедуры оптимизируют расположение полей памяти под сложные структуры данных и преобразуют их к формату принимающей платформы. Обработанные данные декодируются обратно к виду формата платформы, отправившей эти данные.

CDR–стандарт обеспечивает преобразование данных в форматы передающей и принимающей платформ в среде CORBA. Маршаллинг данных выполняется интерпретатором TypeCode и брокером ORB. Процедуры преобразования сложных типов включают в себя:

– дополнительные коды для представления целых чисел и чисел с плавающей точкой (стандарт ANSI / IEEE);

– схему выравнивания значений базовых типов в среде компилятора;

– базовые типы (signed и unsigned) языка IDL, а также плавающий тип двойной точности и др.

Преобразование данных выполняются процедурами encoder (…) и decoder (…) интерпретатора TypeCode с помощью базовых примитивов выравнивания информации и помещения ее в буфер. Для сложного типа вычисляется размер и границы выравнивания, выполняется их размещение в таблице TCKind с индексами значений, которая используется при инициализации брокера ORB.

XML–стандарт устраняет неоднородность во взаимосвязях компонентов в разных ЯП с помощью XML–формата данных. Он учитывает разные платформ и среды (CORBA, DCOM, JAVA и др.), которые имеют в своем составе функции, аналогичные XML. Данный стандарт имеет различные системные поддержки: браузер – Internet Explorer для визуализации XML–документов, объектная модель DOM (Document Object Model) для отображения XML–документов и интерфейс IDL в системе CORBA.

Тексты в XML описываются в формате ASCII, что дает возможность более эффективно применять их при обмене данными и кодировании типов данных с помощью файловых форматов. Переход программной системы к XML–стандарту предполагает переформатирование данных в формат XML и обратно.

Таким образом, XML–язык позволяет представлять объекты для разных объектных моделей на единой концептуальной, синтаксической и семантической основе. Он не зависит от платформы и среды взаимодействия, имеет стандартные методы и средства (XML–парсери, DOM–интерфейсы, XSL–отображение XML в HTML и др.).

5. Перспективы развития проблематики интерфейса

Дальнейшее развитие *методов и средств интерфейса* обусловлено созданием:

– новых интерфейсных языков (Domain Specific Langiage) для взаимодействия доменов и предметных областей, объекты которых можно будет динамически изменять и синхронизировать в распределенных средах;

– языково–ориентированного программирования предметных областей типа Language Workbench, включающего подмножество языков описания специальных задач предметной области и их генерации к языкам типа XML;

– интегрированной среды поддержки генерирующего программирования на основе IDE Eclipse и систем UML, Aspect, Junit, Corba, Java, СУБД, GreenStone и др.

1. *Лаврищева Е.М., Грищенко В.Н.* Связь разноязыковых модулей в ОС ЕС.– М: Финансы и статистика, 1982.–127 с.
2. *Лаврищева Е.М., Грищенко В.М.* Сборочное программирование.– Киев.– Наук. думка, 1991. –213 с.
3. *OSI 7498 –1989*. Open System Interconnection.– Basic Reference Model. – Technical Report, Iinternational Standartization Organization.
4. *Open* Software Foundation. Inroduce to Disributed Computed Environments.– Englewood Cliffs: Prentice Hall, 1993.– 437 p.
5. *Буч Г.* Объектно-ориентированный анализ .– М.: Бином, 1998. – 560 с.
6. *Wegner P.* Interaction Foundation of Object–oriented Programming ECOOP–97 th Europian Conference on OOP Finland, June 9–12, 1997.– С. 123–139.
7. *Letichevsky A.A., Gilbert D.R* . A General Theory of Action Language // Кибернетика и системный анализ. – 1998. – № 1.– С. 16 – 36.
8. *Летичевский А.А., Капитонова Ю.В. и др.* Инсерционное программирование // Там же.– 2003.– № 1.– С. 12–32.
9. *Corbin J.* The art of Distributed Application. Programming Techn. For Remote Procedure Calls.– Berlin: Springer Verlag, 1992.– 305 p.
10. *Роджерсон Д.* Основы COM. Русск..пер.– Microsoft Press,1996.– 361 с.
11. *Эммерих В.* Конструирование распределенных объектов // Методы и средства программирования интероперабельных объектов в архитектурах OMG/CORBA, Microsoft COM и Java RMI. – М.: Мир, 2002. – 510 с.
12. *Сигел Дж.* CORBA 3. – М.: Малип, 2002. – 412 с.
13. *Барлет Н., Лесли А., Симкин С.* Программирование на JAVA // Путеводитель.– Киев.–1996.– 736 с.
14. *Лаврищева Е.М.* Методы программирования // Теория, инженерия, практика.– К .: Наук. думка, 2006. – 451 с.
15. *Гост 30664* (ИСО/МЭК 11404:1996). Информационные технологии. Языки программирования, их среда и системный интерфейс // Независимые от языков типы данных.– 2000. – 109 с.
16. *Бакалов Ю.В., Смелянский Р. Л.* Язык спецификации распределенных программ // Программирование.– 1996.– № 5.– С. 41-52.
17. *Цимбал А., Аншина М.* Технология создания распределенных систем. – Питер*Санкт–Петербург* Москва* Харьков* Минск, 2003.– 575 с.
18. *Бабенко Л.П., Лаврищева Е.М.* Основы программной инженерии.– Киев: Знание, 2001.– 269 с.
19. *Бей И.* Взаимодействие разноязыковых программ // Руководство программиста.– М.: Издательский дом «Вильямс», Москва–Санкт–Петербург–Киев, 2005. – 868 с.
20. *Просиз Дж.* Программирование для Microsoft .NET. – М.: Издательско–торговый дом “Русская Редакция”, 2003. – 704 с.
21. *Питц–Моултис Н., Кирк Ч.* XML: Пер. с англ. – СПб.: BHV–Санкт–Петербург, 2000.– 736 с.
22. *Иванников В.П., Дышлевый К.В., Мажелей С.Г* и др. Распределенные объектно–ориентированные среды. – Москва // РАН.ИСП. Труды ИСП, 2000.– С. 84–100.
23. *Иванова Е. Б., Вершинин М.М.* Java 2, Enterprise Edition. Технологии проектирования и разработки. – СПб.: БХВ–Петербург, 2003. – 1088 с.

Получено 23.04.2007

Об авторе:

Лаврищева Екатерина Михайловна,
заведующая отделом, доктор физико-математических наук, профессор.

Место работы автора:

Институт программных систем
НАН Украины, 03680,
Киек–187, Украина,
проспект Академика Глушкова, 40.
тел. 526 3470