Национальный Открытый Университет "ИНТУИТ": www.intuit.ru
Екатерина Лаврищева, Владимир Петрухин
Лекция 9. Интерфейсы, взаимодействие и изменение программ и данных

В данной лекции рассматриваются базовые понятия программной инженерии – интерфейсы, средства их представления, взаимодействие разноязыковых программ и *преобразование типов данных*. Представлены подходы к обеспечению интерфейсов программ, записанных в разных языках программирования (ЯП), методы преобразования неэквивалентных типов данных при взаимодействии модулей и программ.

Определены общие задачи неоднородности ЯП, платформ и сред, влияющих на установление связей между разноязыковыми программами, сформулированы пути их решения. Рассмотрены рекомендации стандарта ISO/IEC 11404-1996 *по* обеспечению независимых от современных ЯП типов данных.

Рассмотрены подходы к *преобразованию форматов данных* и данных в *БД*, а также методы изменения (эволюции) программ.

8.1. Задачи интерфейса при разработке программ

Общее определение. *Интерфейс* – это *связь* двух отдельных сущностей. Виды интерфейсов: языковые, программные, аппаратные, пользовательские, цифровые и т. п. Программный (*API*) и/или аппаратный *интерфейс* (port) – это способы преобразования входных/выходных данных во время объединения компьютера с периферийным оборудованием. В ЯП – это *программа* или часть программы, в которой определяются *константы*, переменные, параметры и структуры данных для передачи другим.

В программировании термин *интерфейс* олицетворяет собой набор операций, обеспечивающих *определение* видов услуг и способов их получения от программного объекта, предоставляющего эти услуги. На начальном этапе программирования в роли интерфейса выступают *операторы* обращения к ее процедурам и функциям программ через формальные параметры. Программы, процедуры и функции записывались в одном ЯП. *Операторы* обращения включали имена вызываемых объектов (процедур и функций) и *список* фактических параметров, задающих значения формальным параметрам и получаемым результатам. Последовательность и число формальных параметров соответствовало фактическим параметрам. Выполнение функции в среде программы на одном ЯП не вызывало проблем, так как типы данных параметров совпадали.

В случае, 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 типов данных в этих ЯП, структур памяти платформ компьютеров и операционных сред, где они выполняются. Понятие интерфейса, как самостоятельного объекта, сформировалось в связи со сборкой или объединением разноязыковых программ и модулей в монолитную систему на больших ЭВМ (*mainframes*) [8.1, 8.2].

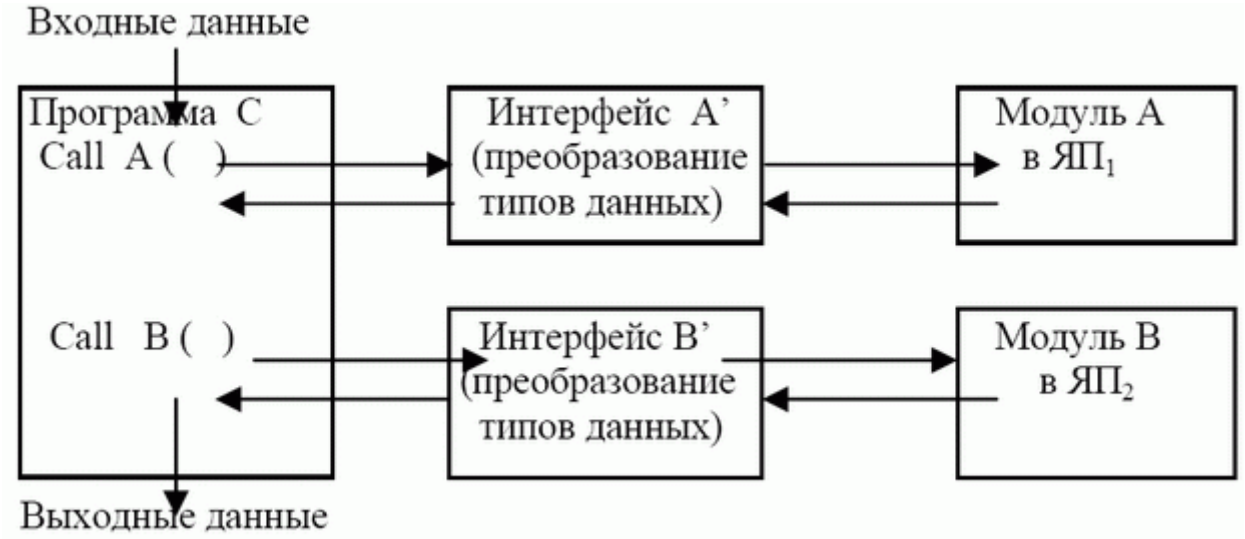


Рис. 8.1. Схема вызова модулей A и B из C через интерфейсы A'и B'

Интерфейс играл роль посредника между вызываемым и вызывающим модулями. В нем давалось описание формальных и фактических параметров, производилась проверка соответствия передаваемых параметров (количества и порядка расположения), а также их типов данных. Если типы данных параметров оказывались не релевантными (например, передается целое, а результат функции – вещественное или наоборот), то производилось прямое и обратное их преобразование с учетом структуры памяти компьютеров. На рис. 8.1 приведена схема программы C, в которой содержатся два вызова – $Call\ A()$ и $Call\ B()$ с

параметрами, которые через интерфейсные модули-посредники A' и B' производят *преобразование данных* и их передачу модулям A и B. После выполнения A и B результаты преобразуются обратно к виду программы C.

8.1.1. Интерфейс в ООП и в современных средах

Интерфейс в ООП. В ООП главным элементом является класс, включающий множество объектов с одинаковыми свойствами, операциями и отношениями. Класс имеет внутреннее (реализацию) и внешнее представление – интерфейс (табл. 8.1).

Таблица 8.1. Структура представления класса и интерфейса

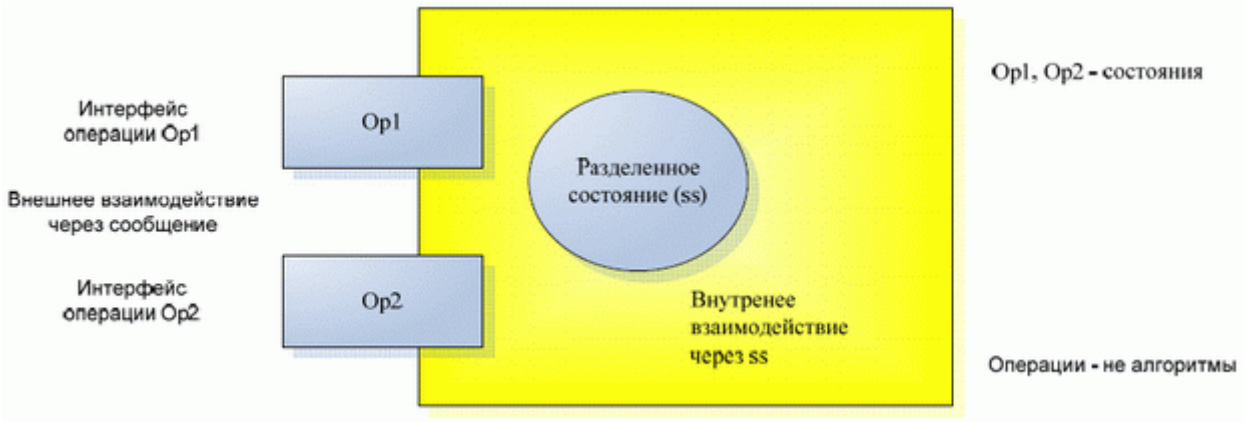
Класс	
Внешнее представление	**Внутреннее представление**
Интерфейсные операции: публичные, доступные всем клиентам: защищенные, доступные классу п подклассу: приватные, доступные классу.	Реализация операций класса, определение поведения.

Интерфейс содержит множество операций, описывающих его поведение. Класс может поддерживать несколько интерфейсов, каждый из которых содержит операции и сигналы, используются для задания услуг класса или программного компонента. Интерфейс именует множество операций или определяет их сигнатуру и 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 ту же операцию, соответствующую поведению системы. Класс может реализовывать другой класс через интерфейс.

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 реализацию можно рассматривать как наследование поведения.

Новое толкование интерфейса объектов дано в работе П.Вегнера [8.3], который сформулировал парадигму перехода от алгоритмоввычислений 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. 8.3). Операции интерфейса ($Op1$ и $Op2$) относятся к классу неалгоритмических и обеспечивают взаимодействие объектов через сообщения.



увеличить изображение

Рис. 8.3. Интерфейс взаимодействия через операции интерфейса (по Вегнеру)

Вегнер рассматривает модель взаимодействия, как обобщение машины Тьюринга – распределенной интерактивной модели взаимодействия объектов с входными (input) и выходными (output) действиями 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), обеспечиывющий вызовов процедур (локальных или распределенных) с разверткой каждого вызова в программу [8.4], состоящую из операторов действий. Программа из вызовов процедур рассматривается в AL как ограниченное множество конечных программ, взаимодействующих со средой, в которую они погружаются.

Интерфейс в современных средах и сетях.Появление разных компьютеров и их объединение в локальные и глобальные сети привело к уточнению понятия интерфейса как удаленного вызова (сообщения) программ, расположенных в разных узлах сети или среды и получающих входные данные из сообщений.

Сети строятся на основе стандартной семиуровневой модели открытых систем OSI (Open Systems Interconnection) [8.5]. Объекты уровней в 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 любым приложениям разного рода услуги: управление удаленными объектами, обслуживание очередей и запросов, обработка интерфейсов и т. п.

Доступ к услугам осуществляется с помощью разных механизмов:

- вызова удаленных процедур RPC (Remote Procedure Call) в системах ONC SUN, *OSF DSE* [8.5, 8.6];
- связывания распределенных объектов и документов в системе DCOM [8.7];
- языка описания интерфейса IDL (*Interface Definition* Language) и *брокера объектных запросов – ORB* (*Object Request Broker*) в системе CORBA [8.8];
- вызова RMI (Remote Methods Invocation) в системе JAVA [8.9, 8.10] и др.

RPC– вызов задает интерфейс удаленным программам в языках высокого или низкого уровней. Язык высокого уровня служит для задания в RPC-вызове параметров удаленной процедуры, которые передаются ей через сетевое сообщение. *Язык низкого уровня* позволяет указывать более подробную информацию удаленной процедуре: тип протокола, размер буфера данных и т. п.

Взаимосвязь процесса с удаленно расположенным от него другим процессом (например, сервером) на другом компьютере выполняет протокол UDP или TCP/IP, который передает параметры в *stub*- интерфейсе клиента *stub*-серверу для выполнения удаленной процедуры.

Механизм посылки запроса в системе CORBA базируется на описании запроса в языке IDL для доступа к удаленному методу/функции через протокол *IIOP* или *GIOP*. Брокер *ORB* передает запрос генератору, затем посылает *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. [8.2], то он обеспечивает взаимодействие объектов в разных сетевых средах.

Вызов метода RMI в системе JAVA выполняет виртуальная машина (virtual machine), которая интерпретирует byte-коды вызванной программы, созданные разными системами программирования ЯП (JAVA, Pascal, C++) на разных компьютерах и средах. Функции RMI аналогичны брокеру *ORB*.

8.1.2. Интерфейс в среде клиента и сервера

В распределенной среде реализуется два способа связывания: на уровне ЯП через интерфейсы прикладного программирования и компиляторов IDL, генерирующих клиентские и серверные Stab. Интерфейсы определяются в языках IDL или *APL*, динамический интерфейс от объектаклиента к объектусервера и обратно выполняет брокер *ORB*. Интерфейсы имеют отдельную реализацию на ЯП и доступны разноязыковым программам. Компиляторы с IDL как часть промежуточного слоя сами реализуют связывание с ЯП через интерфейс клиента и сервера, заданного в том же ЯП [8.8, 8.11–8.13].

Интерфейс в IDL или в API включает описание формальных и фактических параметров программ, их типов и порядка задания операций передачи параметров и результатов при их взаимодействии. Это описание есть не что иное, как спецификация интерфейсного посредника двух разноязыковых программ (аналогично, как на рис. 8.1), которые взаимодействуют друг с другом через механизм вызова интерфейсных функций или посредников двух типов программ (клиент и сервер), выполняемых на разных процессах.

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 не зависит от ЯП взаимодействующих объектов и в целом одинаково для всех вызывающих и вызываемых объектов. Посредник описывается в языке спецификации интерфейса IDL.

Интерфейсные посредники задают связь между клиентом и сервером (*stub* для клиента и *skeleton* для сервера). Их описания отображаются в те ЯП, в которых представлены соответствующие им объекты или компоненты. Эти интерфейсы используются в системах CORBA, DCOM, LAVA и др. Они предоставляют всевозможные сервисы разработки и выполнения приложений в распределенной среде. Системные сервисы подключаются к приложению с помощью брокера. Брокер обеспечивает *интероперабельность* компонентов и объектов при переходе из одной среды другую.

Под *интероперабельностью* понимается способность совместного, согласованного взаимодействия разнородных компонентов системы для решения определенной задачи.

К средствам обеспечения интероперабельности и передачи данных между разными средами и платформами относится, например, стандартный механизм связи между JAVA и C/C++ компонентами, основанный на применении концепции Java Native Interface (*JNI*), реализованной как средство обращения к функциям из JAVA- классов и библиотек, разработанных на других языках.

Эти средства включает в себя анализ JAVA-классов в целях поиска прототипов обращений к функциям, реализованных на языках C/C++, и генерацию заголовочных файлов для использования их при компиляции C/C++ программ. В средстве JAVA классу известно, что в нем содержится обращение не к JAVA-методу (он называется $native$ и для загрузки необходимых C/C++ библиотек добавляется вызов функции), ориентируется именно на такую связь. Данная схема действует в одном направлении – от JAVA к C/C++ и только для такой комбинации ЯП.

Еще вариант реализации аналогичной задачи предлагает технология Bridge2Java,которая обеспечивает обращение из JAVA- классов к COM-компонентам. В этих целях генерируется оболочка для COM-компонента, который включает прокси-класс, обеспечивает необходимое преобразование данных средствами стандартной библиотеки преобразований типов. Данная схема не требует изменений в исходном Java-классе и COM-компоненты могут быть написаны в разных языках.

Механизм интероперабельности реализован также на платформе . Net с помощью языка CLR (Common Language Runtime). В этот язык транслируются коды, написанные в разных ЯП (C#, Visual Basic, C++, Jscript). CLR разрешает не только интегрировать компоненты, разработанные в разных ЯП, а и использовать библиотеку стандартных классов независимо от языка реализации.

Такой подход позволяет реализовать доступ к компонентам, которые были разработаны раньше без ориентации на платформу . Net, например к COM-компонентам. Для этого используются стандартные средства генерации оболочки для COM-компонента, с помощью которой он представляется как . Net-компонент. При такой схеме реализуются все виды связей и для любых ЯП данной среды.

8.2. Интерфейс ЯП

8.2.1. Интерфейс и взаимосвязь с ЯП

Основные ЯП, используемые для описания компонентов в современных средах, это C++, Паскаль, JAVA и др. [8.2, 8.12, 8.13].

Разноязыковые программы, записанные в этих языках, обращаются друг к другу через удаленный вызов, который предполагает взаимно однозначное соответствие между фактическими параметрами $V = \{v^1, v^2, \ldots, v^к\}$ вызывающей программы и формальными параметрами $F = \{f^1, f^2, \ldots, f^{к1}\}$ 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 в ЯП. Представим это преобразование такими этапами.

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

Этап 2.Построение отображения простых типов данных для каждой пары взаимодействующих компонентов в $l_{\alpha 1}$ и $l_{\alpha 2}$, а также применение операций селектора S и конструктора C для отображения сложных

структур данных в этих языках. Один из способов формализованного преобразования типов данных – создание алгебраических систем для каждого типа данных T_α^t:

$$G_\alpha^t =< X_\alpha^t, \Omega_\alpha^t >,$$

где t – тип данных, X_α^t – множество значений, которые могут принимать переменные этого типа данных, Ω_α^t – множество операций над этими типами данных.

В качестве простых типов данных современных ЯП могут быть $t = b(bool), c(char), i(int), r(real)$. *Сложные типы данных* $t = a(array), z(record), u(union), e(enum)$ – комбинация простых типов данных. Этим типам данных соответствуют следующие классы алгебраических систем:

$$\begin{matrix} \sum_1 = \{G_\alpha^b, G_\alpha^c, G_\alpha^i, G_\alpha^r\} \\ \sum_2 = \{G_\alpha^a, G_\alpha^z, G_\alpha^u, G_\alpha^e\} \end{matrix} \quad (8.1)$$

Каждый элемент класса простых и *сложных типов данных* определяется на множестве значений этих типов данных и операций над ними:

$G_\alpha^t =< X_\alpha^t, \Omega_\alpha^t >$, где $t = b, c, i, r, a, z, u, e$.

Операциям преобразования каждого t типа данных соответствует изоморфное отображение двух алгебраических систем с совместимыми типами данных двух разных языков. В классе систем (8.1) преобразование типов данных $t \to q$ для пары языков l_t и l_q обладает такими свойствами отображений:

1. системы G_α^t и G_β^q для языков l_t и l_q – изоморфны, если их типы данных q, t определены на одном том же множестве простых или *сложных типов данных*;
2. между значениями X_α^t и X_β^q типов данных t и q существует изоморфизм, если множества операций Ω_α^t и Ω_β^q, применяемых для этих типов данных, различны. Если это множество пусто, то имеем изоморфизм двух систем $G_\alpha^{t'} =< X_\alpha^t, \Omega >$ и $G_\beta^{q'} =< X_\beta^q, \Omega >$. Если тип данных t есть строка, а тип q – вещественное, то между множествами X_α^t и X_β^q не существует изоморфного соответствия;
3. *алгебраические системы* $|G_\alpha^t| = |G_\beta^q|$ по мощности должны быть равны, так как они представлены на множестве типов данных языков l_t и l_q.

Отображения 1, 2 сохраняют линейный порядок элементов, поскольку *алгебраические системы* являются линейно упорядоченными. **Общая схема связи ЯП в распределенной среде.**Характерная особенность ЯП, используемых в распределенных средах, – их неоднородность как в смысле представления типов данных в них, так и платформ компьютеров, где реализованы соответствующие системы программирования. Причина неоднородности – это различные способы передачи параметров между объектами в разных средах, наличие разных типов объектных моделей и форматов данных для задания параметров, разные виды операторов удаленного вызова и получения результатов выполнения запросов и др.

Системы программирования с ЯП имеют следующие особенности компилирования программ:

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

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

параметры вызовов объектов отображаются в операции методов;

связь с разными ЯП реализуется ссылками на указатели в компиляторах;

связь ЯП осуществляется через интерфейсы каждой пары из множества языков $(L_1, \ldots, L_n)$ промежуточной среды.

Связь между различными языками $L_1, \ldots, L_n$ осуществляется через интерфейс пары языков L_i, L_n, взаимодействующих между собой в среде, генерирующей соответствующие конструкции L_i в операции описания интерфейса и наоборот.

Взаимодействие ЯП в среде CORBA. Принцип взаимодействия объектов в среде CORBA состоит в том, что любой объект выполняет метод (функцию, сервис, операцию) при условии, если другой объект, выступающий в роли клиента для него, посылает ему запрос для выполнения этого метода. Объект выполняет метод через интерфейс.

Взаимодействие ЯП в системе CORBA состоит в отображении типов объектов в типы клиентских и серверных стабов путем

отображения описания запроса клиента в ЯП в операции IDL;

преобразования операций IDL в конструкции ЯП и передачу их серверу средствами брокера *ORB*,

реализующего $stub$ в типы данных 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 [8.2, 8.7, 8.8]. Для всех ЯПсистемы 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. 8.4).

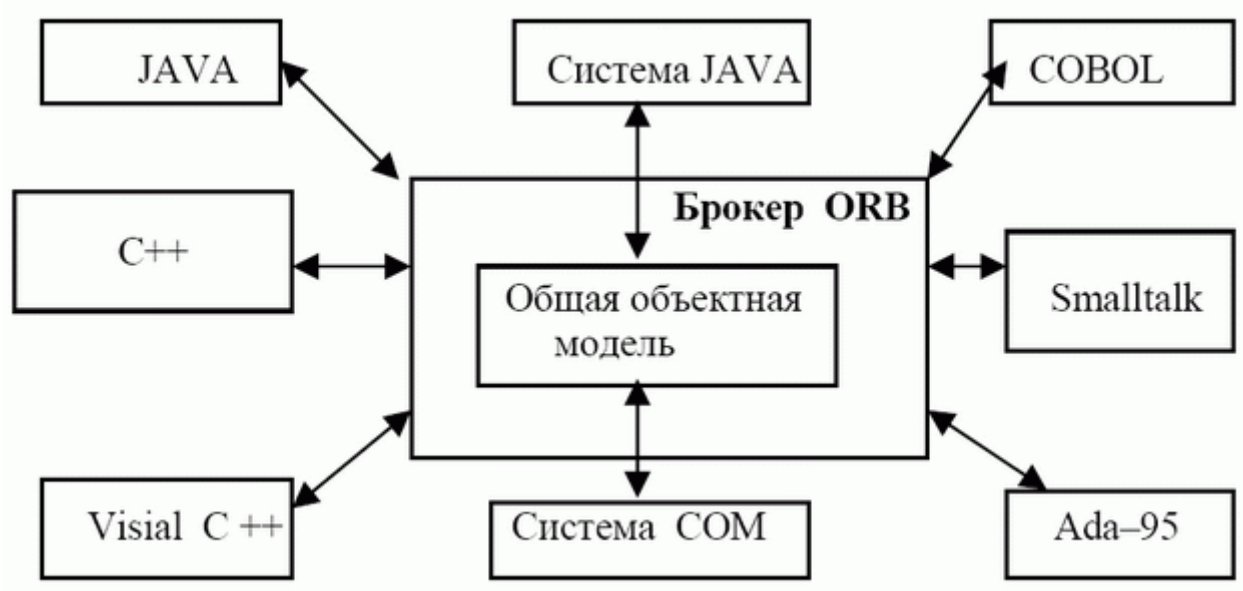


Рис. 8.4. Интегрированная среда системы CORBA

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 коде и допускается двоичная совместимость 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 ссылками на объекты, задаваемые указателями на адреса памяти.

Интерфейс как объектный тип реализуется классами и предоставляет удаленный доступ к нему сервера. Компилятор JAVA создает байткод, который интерпретируется виртуальной машиной, обеспечивающей переносимость байткодов и однородность представления данных на всех платформах среды CORBA.

8.2.2. Взаимодействие разноязыковых программ

Проблеме взаимодействия разноязыковых программ на множестве современных языков (C/C++, Visual C++, Visual Basic, Matlab,Smalltalk, Lava, LabView, Perl) посвящена работа [8.14]. В ней представлены различные варианты и конкретные примеры связей каждой пары ЯП из этого множества с помощью практически реализованных и приведенных функций преобразования, методов обращения к ним из программ на одном языке к программе на другом языке. В таблице 8.1. приведены варианты взаимосвязи разных ЯП.

В ней отражены особенности их взаимодействия через разные виды интерфейсов, приведены более 25 видов пар современных ЯП и соответственно прямого и обратного взаимодействия разноязыковых программ.

Для этих пар ЯП изложены принципы запуска разных программ и все технические вопросы передачи данных и преобразования параметров.

Материал учебного пособия 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. 8.1), здесь рассмотрены высокотехничные средства обеспечения процесса преобразования: панели, сценарии, иконки и образцы интерфейсных программ для каждого конкретного случая взаимодействия программ. Далее дается краткое описание шести схем средств описания разноязыковых программ, взаимодействующих с языками, приведенными во второй колонке данной 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 функциями Windows API, API DLL и операциями преобразования типов данных. В качестве примера приведена схема обработки Интернет-приложений, задаваемых HTML-страницами Basic Visual, размещаемых в Web-браузере и базах данных.

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. Приведена схема независимого приложения в среде Matlab, которая включает интерфейс между *VC* и Matlab, создаваемый MatlabCompiler путем преобразования программы в формате Matlab (M-файлы или M-функции) в формат C.

Таблица 8.2. Интерфейс современных языков и средств программирования

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

JAVA	C, C++ Visual C++ Matlab	Платформенно-ориентированные функции. Библиотеки функций в C++, C. Функции в *JNI*.
Perl	C, C++ API Visual C++	Платформенно-ориентированные функции. Программный интерфейс. Интерфейсные функции в C++.

Сформированный файл вызывается из программы в C++ и преобразовывается к виду архитектуры компьютера, куда отсылается 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. 8.5). Модель приложения содержит функции DLL из класса внешнего интерфейса, взаимодействующие с функциями библиотеки C++.

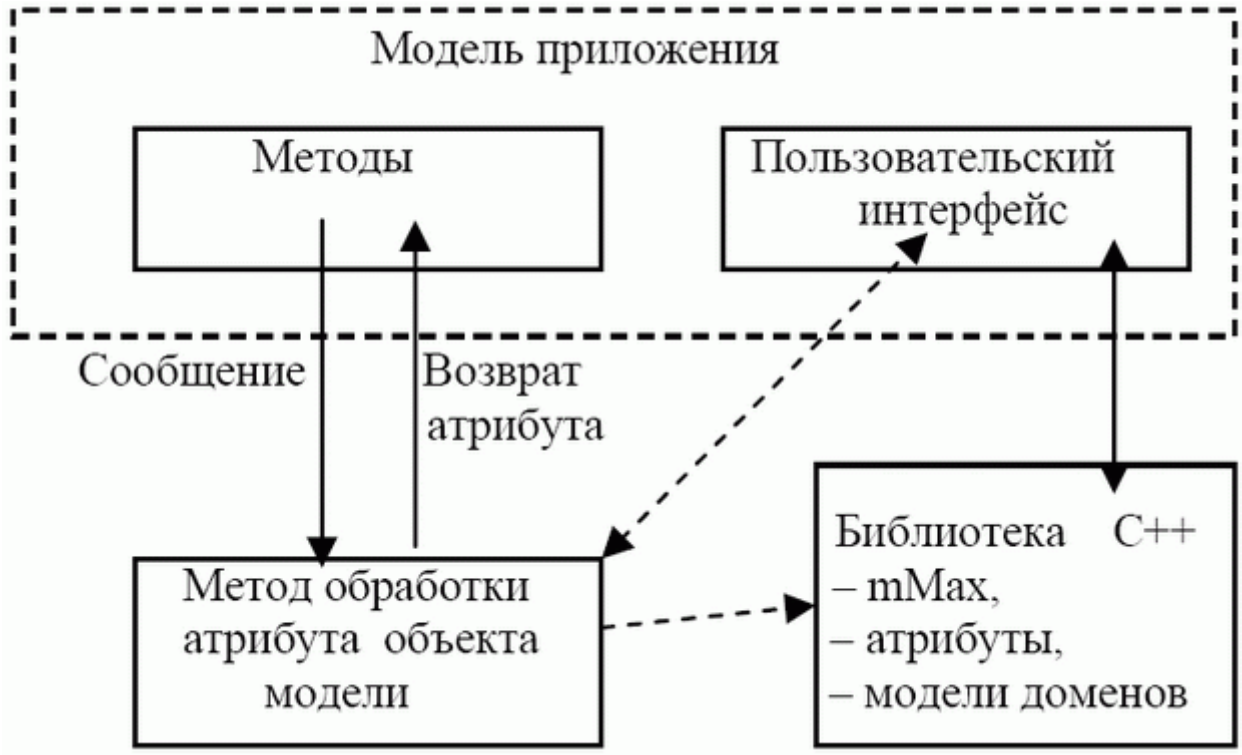


Рис. 8.5. Схема взаимодействия модели приложения с библиотекой

Система LabView предназначена для автоматизации производственных процессов, сбора данных, проведения измерений и управление созданием программ, взаимодействующих с аппаратурой. В ее состав входят 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 реального времени, которые позволяют производить с помощью функций связи измерение аппаратуры типа: регуляторы, термометры, переключатели и др. Результаты измерений могут передаваться в сеть.

Среда Java содержит инструменты взаимодействия со всеми языками, приведенными во второй колонке таблицы. Общая схема связи языков JAVA, C и C++ программ приведена на рис. 8.6. *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++, 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.

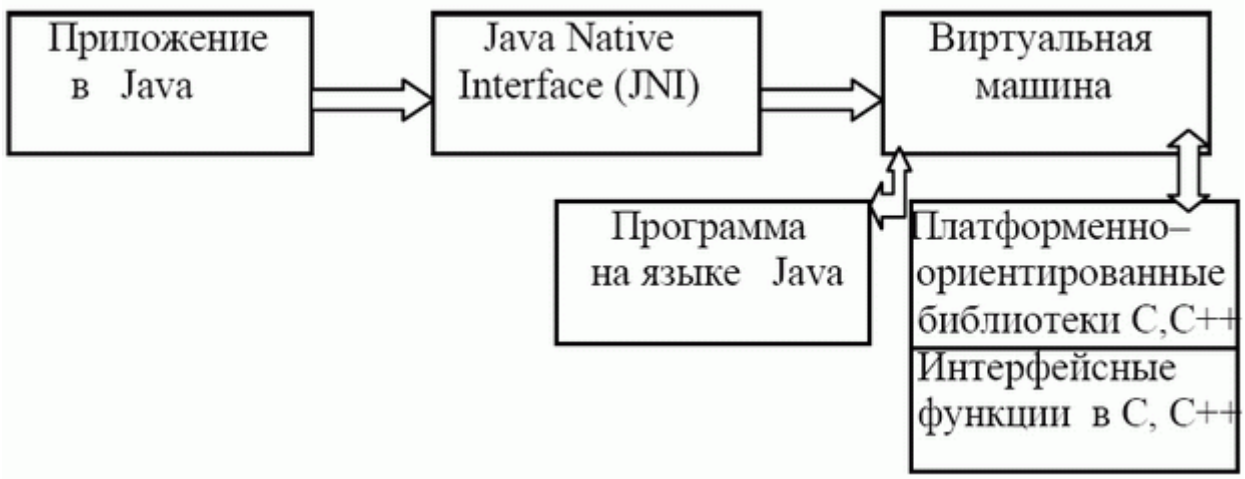


Рис. 8.6. Схема взаимодействия приложенияи программ Java, C, C++

Оператор вызова программы в C или C++ обеспечивает преобразование ее в специальный код, который размещается в библиотеке интерпретатора Perl. Сам интерпретатор может быть включен в Win32 или в программу на C/C++.

Таким образом, в работе [8.14] тщательно исследованы самые современные средства и инструменты представления разноязыковых программ и принципы их взаимодействия с широко используемыми ЯП. Даны рекомендации по конкретному применению каждого средства с учетом условий среды и правил прямой и обратной передачи параметров программе в ЯП из класса рассмотренных ЯП. Приведены многочисленные примеры, которые проверены экспериментально, ими можно пользоваться на практике либо использовать в качестве образца.

8.2.3. Стандарт ISO/IEC 11404-1996 преобразования данных

Цель данного стандарта [8.15] состоит в том, чтобы обеспечить не только описание типов данных в стандартном языке LI (*Language Independent*) и их генерацию, но и преобразование типов данных ЯП в LI-язык и наоборот. Стандарт предлагает специальные правила ихарактеристические операции генерации *примитивных типов данных* и объединений LI-языка в более простые структуры данных ЯП, а также определение параметров интерфейса средствами языков IDL, RPC и API.

Независимые отЯП типы данных стандарта разделены на примитивные, агрегатные и сгенерированные (рис. 8.7).

увеличить изображение

Рис. 8.7. Независимые от ЯП типы данных стандарта ISO/IEC 11404-1996

В этот язык также включено семейство и генератор типов данных. Типы данных в стандарте описываются в LI-языке, который является более общим языком, в отличие от конкретных средств описания типов данных ЯП. Он содержит все существующие типы ЯП и общие типы данных, ориентированные на генерацию других типов данных.

Стандарт имеет раздел объявления типов данных, переименования существующих; объявление новых генераторов, значений и результатов. Каждый тип данных имеет шаблон, включающий описание и спецификатор типа данных, значение в пространстве значений, синтаксическое описание и операции над типами данных.

Стандарт состоит из разделов (рис. 8.8): объявление типов данных, объявленные типы данных; объявление новых генераторов (стеков, деревьев). Для объявляемого типа данных задается шаблон, включающий синтаксическое описание, спецификатор типа данных, значение в пространстве значений и операции над типами данных.

Средствами LI описываются параметры вызова, как элементы интерфейса, необходимые при обращении к стандартным сервисам и готовым программным компонентам.

LI-язык стандарта рекомендует следующие виды преобразования данных:

- внешнее преобразование типов данных ЯП в LI-тип данных;
- внутреннее преобразование из LI-типа данных в тип данных ЯП;
- обратное преобразование.

Рис. 8.8. Объявление типов данных в стандарте ISO/IEC 11404-1996

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

- для каждого примитивного типа для сгенерированного внешнего типа данных преобразование связывается с одним LI-типом данных;
- для каждого внутреннего типа данных преобразование определяет связь между допустимым значением внутреннего типа данных и эквивалентным значением соответствующего LI-типа данных;
- для каждого значения LI-типа данных, участвующего в преобразовании, определяется существование значения любого внутреннего типа данных, преобразуемого в LI-тип данных с взятием этого значения.

Внешнее преобразование документирует аномалии при идентификации внутренних типов и дает гарантию того, что интерфейс между программными компонентами адекватно задается сервисным средством и игнорирует среду ЯП.

Внутреннее преобразование связывает примитивный тип данных или сгенерированный в LI-тип данных с конкретным внутренним типом данных ЯП. Представители отдельного семейства LI-типа данных могут

преобразовываться в различные внутренние типы данных ЯП. Данное преобразование обладает следующими свойствами:

- для каждого LI-типа данных (примитивного или сгенерированного) преобразование определяет наличие этого типа данных в ЯП;
- для каждого LI-типа данных преобразование определяет отношение между допустимым значением этого типа и эквивалентным значением соответствующего внутреннего типа ЯП;
- для каждого значения внутреннего типа данных преобразование определяет, является ли это значение образом (после преобразования) какогото значения LI-типа данных и его способ преобразования.

Обратное внутреннее преобразование для LI-типа данных состоит в преобразовании значений внутреннего типа данных в соответствующее значение LI-типа при наличии соответствия и отсутствия двусмысленности. Это преобразование для ЯП является коллекцией обратных внутренних преобразований LI-типа данных.

В стандарте имеется набор приложений.

В "Малая энциклопедия инструментов ООП" приведен перечень действующих стандартов (около 40), определяющих наборы символов. Для обеспечения совместимости используемых и реализуемых типов данных в "Словарь терминов программной инженерии" содержатся рекомендации по идентификации типов данных и описанию аннотаций для атрибутов, параметров и др.

В "Перечень стандартов программной инженерии" приведены рекомендации по соответствующим внутренним типам данных, которые должны преобразовываться LI-типы данных. В "Малая энциклопедия инструментов ООП" показано, что синтаксис LI-языка является подмножеством стандарта *IDM* (*Interface Definition* Notation), предназначенного для описания интерфейса в LI-языке. Приведен вариант внутреннего преобразования LI-типов данных в типы данных ЯП Паскаль (ISO/IEC 7185-90). В нем рассмотрены примеры преобразования *примитивных типов данных* LI-языка (логический, перечислимый, символьный, целый рациональный и др.) в типы данных языка Паскаль.

Предложенные в стандарте рекомендации, а также средства описания типов данных и методов их преобразования – универсальны. Программной поддержки стандарта пока отсутствует.

8.3. Интерфейс платформ и преобразование данных

8.3.1. Преобразование форматов данных

Программы, расположенные на разных типах компьютеров, передают друг другу данные через протоколы, их форматы преобразуется к формату данных принимающей серверной платформы (так называемый маршалинг данных) с учетом порядка и стратегии выравнивания, принятой на этой платформе. Демаршалинг данных – это обратное преобразование данных (т. е. полученного результата) к виду передавшей клиентской программы. Если среди передаваемых параметров оператора вызова содержатся нерелевантные типы или структуры данных, которые не соответствуют параметрам вызванного объекта, то производится прямое и обратное их преобразование средствами ЯП или стандарта [8.2, 8.11].

К средствам преобразования данных и их форматов относятся:

- стандарты кодировки данных (*XDR* – eXternal *Data Representation*, CDR – Common Representation Data [8.8]), *NDR* – Net *Data Representation*) и методы их преобразования;
- ЯП и механизмы обращения компонентов друг к другу;
- языки описания интерфейсов компонентов – RPC, IDL и RMI для передачи данных между разными компонентами.

На каждой платформе компьютера используются соглашения о кодировке символов (например, ASCII), о форматах целых чисел и чисел с плавающей точкой (например, IEEE, VAX и др.). Для представления целых типов, как правило, используется дополнительный код, а для типов float и double-стандарт ANSI/IEEE и др. Порядок расположения байтов зависит от структуры платформы (Big *Endian* или Little *Endian*) и от старшего к младшему байту и 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, некоторые из них рассмотрим ниже.

XDR-стандарт содержит язык описания структур данных произвольной сложности и средства преобразования данных, передаваемых на платформы (Sun, VAX, IBM и др.). Программы, написанные в ЯП, могут использовать данные в *XDR*-формате, несмотря на то, что компиляторы выравнивают их в памяти машины поразному.

В *XDR*-стандарте целые числа с порядком "от младшего" приводятся к порядку байтов "от старшего" и обратно. Преобразование данных – это кодирование (code) или декодирование (decode) *XDR*-процедурами форматирования простых и *сложных типов данных*. 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 – это размещение значений базовых типов с адреса, кратного действительному размеру в байтах (2, 4, 8, 16). Границы данных выравниваются по наибольшей длине (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 среды CORBA обеспечивает преобразование данных в форматы передающей и принимающей платформ. Маршаллинг данных выполняет интерпретатор TypeCode и брокер *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`) в IDL, а также плавающий тип двойной точности и др.

Преобразование данных выполняются процедурами `encoder()` и `decoder()` интерпретатора TypeCode, который используют базовые примитивы при выравнивании информации и помещения ее в буфер. Для сложного типа вычисляется размер и границы выравнивания, а также их размещение в таблице с индексами значений TCKind, используемых при инициализации брокера *ORB*.

XML-стандарт обеспечивает устранение неоднородности во взаимосвязях компонентов в разных ЯП с помощью XML-формата данных, который учитываются разные платформ и среды. Промежуточные среды (CORBA, DCOM, JAVA и др.) имеют в своем составе специальные функции, аналогичные XML – альтернатива сервисам CORBA в плане обеспечения взаимосвязей разноязыковых программ.

XML 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-стандарте описываются в формате ASCII, что дает возможность более эффективно применять их при обмене данными. XML используется для кодирования типов данных с помощью файловых форматов. При необходимости перехода программной системы к XML-стандарту проводится переформатирование данных системы в формат XML и наоборот.

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 модели взаимодействия компонентов прикладного уровня. XML упрощает обработку документов, работу с БД с помощью стандартных методов и средств (XML-парсеры, DOM-интерфейсы, XSL-отображение XML в HTML и др.).

8.3.2. Преобразование данных БД

Преобразование данных БД связано с различием логических структур данных, а также со следующими проблемами:

1. Многомодельность представления данных (иерархические, сетевые, реляционные) в различных БД и СУБД;
2. Различия в логических структурах данных, в справочниках, классификаторах и в системах кодирования информации;
3. Использование различных языков для представления текстовой информации;
4. Разные типы СУБД и постоянное развитие данных БД в процессе эксплуатации.

Проблема 1 решается путем перехода к реляционной модели данных и СУБД, которая является мощным математическим аппаратом, основанным на теории множеств и математической логике. Эта модель состоит из структурной, манипуляционной и целостной частей. В этих частях соответственно фиксируется структура данных, описание программ в SQL-языке и требования к целостности. Иерархические или *сетевые модели данных* в общем не поддерживают целостность, поэтому при переходе от них к реляционным БД возникает нарушение целостности данных.

Проблема 2 вызвана тем, что логическая структура данных или концептуальная схема БД предполагают проектирование новой структуры БД при изменении предметной области или при переходе на новый тип СУБД. При этом сопоставляются данные старой и новой БД и изменяется справочная информация и классификаторы.

Проблема 3 определяется разноязычными текстовыми представлениями информации в БД. В старых БД использовался, как правило, один язык, а в новых может быть их несколько, поэтому для хранения данных с простым доступом к текстовым данным устанавливается соответствие текстовых данных, записанных в разных языках.

Проблему 4 можно сформулировать как метод хранения и обработки разных данных, вызванных спецификой СУБД иерархического, сетевого и реляционного типов. Наличие явной несовместимости типов и структур

этих моделей данных, различные языки манипулирования данными приводят к тому, что нельзя сгенерировать на языке старой СУБД скрипты для переноса данных и последующего запуска БД в среде другой СУБД. Каждая СУБД обеспечивает внесение изменений в БД, которые в некоторой степени меняют и *концептуальную модель данных*, если в нее вносятся новые объекты. Внесенные изменения должны отображаться в справочниках и классификаторах, что обеспечивает перенос данных из старой БД в новую с учетом текущих изменений [8.18, 8.19].

Преобразование данных в БД.Учитывая приведенные проблемы, рассмотрим пути их решения. При длительной промышленной эксплуатация систем, работающих с БД, могут изменяться прикладные программы и данные, если в систему введена новая БД, а часть ранее определенных данных перенесена в новую БД. Это влечет за собой доработку прикладных программ доступа, чтобы приспособить их к измененной структуре новой БД или к старой БД. Для переноса данных из старой БД в новую создаются скрипты или DBF-файлы, которые размещаются в транзитной БД для переноса в новую БД. Если окажется, что процесс приведения структуры транзитной БД к новой окажется нецелесообразным, то разработка новой БД проводится "нуля". При этом справочники и классификаторы дополняются появившимися новыми данными.

Проблемы преобразования данных при использовании разных СУБД возникают также из-за того, что данные имеют различные способы хранения данных, среди которых могут оказаться несовместимые типы данных или доступ к данным осуществляется разными языками манипулирования.

Преобразование данных может проводиться несколько раз путем создания специальных скриптов и файлов с учетом ранее введенных данных, без их дублирования и корректного приведения несовместимых типов данных. Могут возникнуть ошибки, связанные с изменением форматов данных, дополнением старых справочников новыми данными и т. п.

Этапы преобразования данных основаны на использовании:

- *метода 1*, выполняющего перенос данных из старой БД в транзитные файлы, а затем занесение этих файлов в транзитную БД;
- *метода 2* для обработки данных в транзитной базе при изменении кодировки данных, приведении соответствия между структурами старой и новой БД, а также кодов справочников и классификаторов;
- *метода 3*, предназначенного для системного переноса данных из транзитной базы в основную БД с проверкой преобразованных данных.

Первый метод – наиболее безболезненный для пользователей и разработчиков. Второй метод представляет собой создание нового проекта системы с заданной моделью данных. При третьем методе – система создается заново, в новую БД могут заноситься унаследованные данные из старой БД. Поскольку структуры БД могут оказаться различными, то, как правило, создаются временные приложения, в которых осуществляются необходимые преобразования данных при переносе в новую БД.

При применении первого и второго методов структура старой БД сохраняется и никакого преобразования данных, соответствия справочников и классификаторов не требуется, так как они используют единый формат хранения данных. **Файлы передачи данных между разными БД.**Проблема преобразования и переноса данных между различными СУБД решается на основе использования:

1. специального драйвера (две СУБД соединяются друг с другом и напрямую передают данные, используя интерфейс);
2. транзитных файлов, в которые копируются данные из старой БД для переноса в новую БД.

Процесс преобразования и переноса данных из разных БД в новую БД приведен на рис. 8.9.

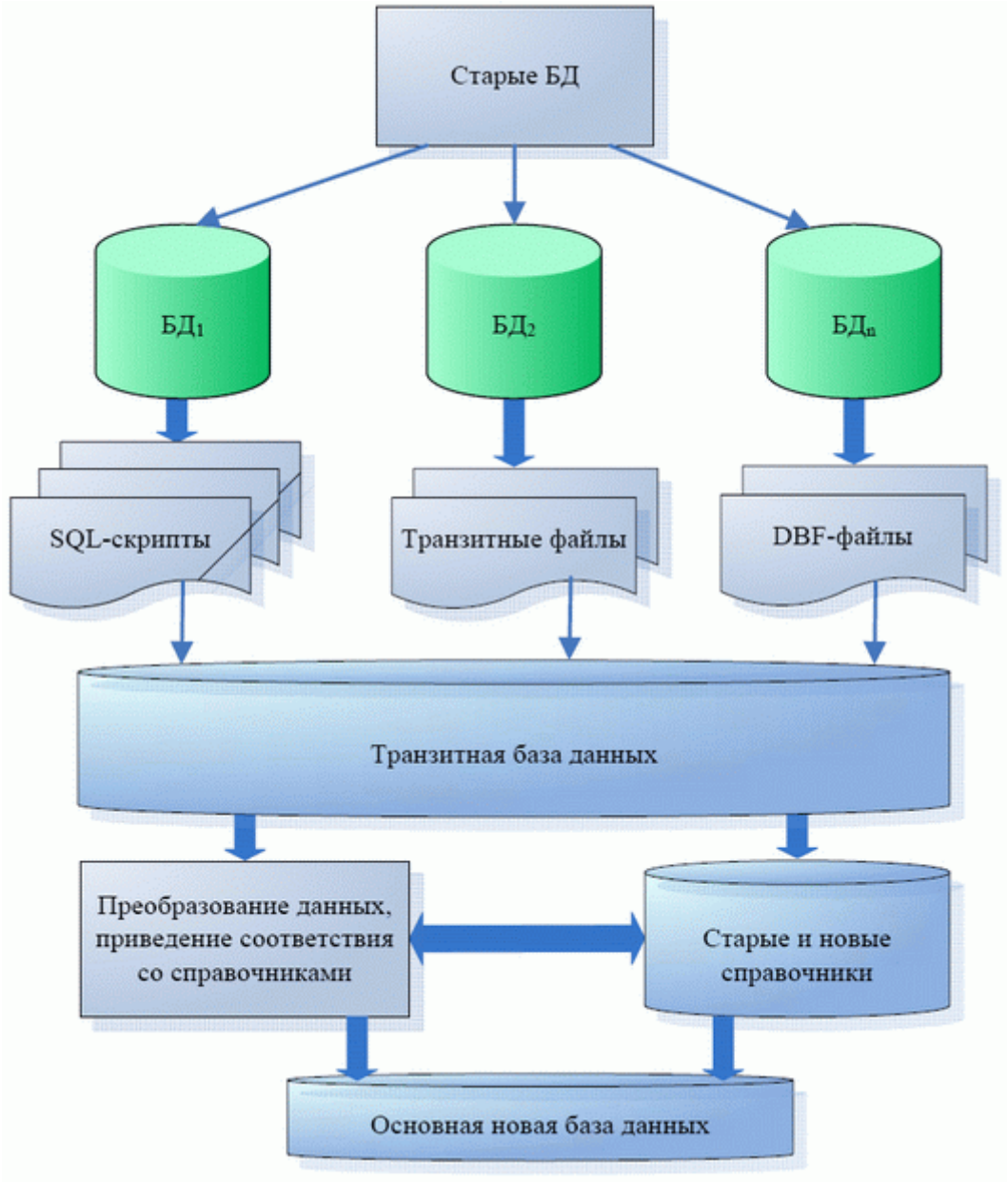


Рис. 8.9. Процесс преобразования и формирования новой БД из старых БД

В случае использования драйвера две СУБД соединены напрямую и передают данные, используя определенный интерфейс и специальныепрограммы взаимодействия двух СУБД, при которых вторая СУБД понимает результаты выполнения запросов на языке манипулирования данными первой СУБД, и наоборот. Данные на выходе первой СУБД являются данными на входе второй СУБД в языке манипулирования данными второй СУБД, такие данные могут быть внесены в транзитную БД.

Данный метод сложный в реализации и требует поставки программ переноса данных из других СУБД, которые привязаны к старой и новой СУБД. Поэтому второй метод переноса данных между различными СУБД – более предпочтительный.

Во втором случае данные из старой БД переносятся в транзитные файлы, SGL-скрипты, DBF-файлы с заранее заданными форматами данных, которые пересылаются в новую транзитную БД через сеть с помощью специальных утилит или средств новой СУБД.

Если вторая СУБД реляционного типа, то данные в транзитных файлах преобразуются к табличному виду. Если первая СУБД не реляционная, то данные должны быть приведены к табличному виду и первой нормальной форме.

Дальнейшая нормализация данных и приведение их к структуре новой БД осуществляется в транзитной БД с использованием 3-я или 4-я нормальной формы для задания структур данных. Каждая более высокая форма нормализации содержит в качестве подмножества более низкую форму, например, первую нормальную форму в виде скалярных значений.

Иными словами, отношения находятся в первой нормальной форме, если они хранятся в табличном виде (все ячейки в строке таблицы расположены в строго определенной последовательности) и каждая ячейка таблицы содержит только атомарные значения (элемент не является множеством).

Отношение находится в *третьей нормальной форме* тогда и только тогда, когда каждый кортеж состоит из значения первичного ключа, идентифицирующего некоторую сущность, и набора пустых значений или значений независимых атрибутов этой сущности. Т. е. отношение находится в *третьей нормальной форме*, когда неключевые атрибуты – *взаимно независимы*, но зависят от первичного ключа.

Два или несколько атрибутов – *взаимно независимы*, если ни один из них не зависит функционально от какойлибо комбинации остальных атрибутов. Подобная независимость подразумевает, что каждый атрибут может быть обновлен независимо от остальных. *Процесс нормализации* отношений позволяет избавиться от проблем, которые могут возникнуть при обновлении, внесении или удалении данных, а также при обеспечении целостности данных.

Структуры старых БД не всегда можно привести к *третьей нормальной форме*, поэтому требуется, чтобы данные, находящиеся в транзитных файлах, существовали хотя бы в первой нормальной форме и относились к реляционной модели.

В качестве унифицированного формата транзитных файлов используется формат DBF-файлов, поскольку многие СУБД, такие как DB2, FoxPro и некоторые другие хранят данные в таких файлах, тем самым не требуется начальный перенос данных из старой СУБД в транзитные файлы. Большинство СУБД, формат хранения данных которых отличается от формата DBF-файлов, снабжены утилитами или драйверами, которые позволяют перенести данные в такой формат.

8.4. Методы изменения (эволюции) компонентов и ПС

Активное использование готовых ПС проводится при создании и сопровождении системы. При этом возникают разного рода ошибки, которые требуют внесения изменений в систему после того, как ошибка обнаружена или возникла необходимость в изменении или улучшении некоторых характеристик системы [8.15, 16-22] .

В отличие от технического обеспечения, которое с течением времени требует ремонта, *программное обеспечение* не "снашивается", и поэтому процесс сопровождения нацелен более всего на эволюцию системы, то есть не только на исправление ошибок, а и на замену ее отдельных функций и возможностей.

Типичные причины внесения изменений это:

- выявление дефектов в системе во время эксплуатации, которые не были обнаружены на этапе тестирования;
- выяснение несоответствия или невыполнения некоторых требований заказчика, благодаря чему система не выполняет отдельные функции;
- изменение условий заказчиком, которые связаны с корректировкой ранее поставленных им требований.

Как утверждают эксперты, процесс внесения изменений в эксплуатируемую систему достаточно дорогой, оценки его стоимости достигают от 60 до 80 % от общей стоимости разработки системы.

К видам сопровождения относятся:

- корректировка – внесение изменений в ПС для устранения ошибок, которые были найдены после передачи системы в эксплуатацию;
- адаптация продукта к измененным условиям (аппаратуре, ОС) использования системы после ее передачи в эксплуатацию;
- *предупредительное* сопровождение – деятельность, ориентированная на обеспечение адаптации системы к новым техническим возможностям.

Одна из проблем, влияющая на процесс внесения изменений, – это степень подготовки персонала, способного вносить необходимые изменения при возникновении определенных нерегулярных условий.

В связи с тем, что почти каждые 8-10 лет происходит смена архитектур компьютеров, ЯП и операционных сред, возникают проблемы сопровождения готовых ПС и их компонентов в новой среде или архитектуре, решение которых приводит к изменению либо обновлению отдельных элементов системы, или системы полностью.

В общем, процесс изменения (эволюции) ПС проводятся путем:

- анализа исходного кода для внесения в него изменений;
- настройки компонентов и системы на новые платформы;
- кодирования и декодирование данных при переходе с одной платформы на другую;
- изменения функций системы или добавления новых;
- расширения возможностей (сервиса, мобильности и др.) компонентов;
- преобразования структуры системы или отдельных ее компонентов.

Цель внесения изменений в один *компонент* или в их совокупности – придание старой ПС нового назначения в новых условий применения. Методы изменения ПС служат способом продления жизни наследуемых и

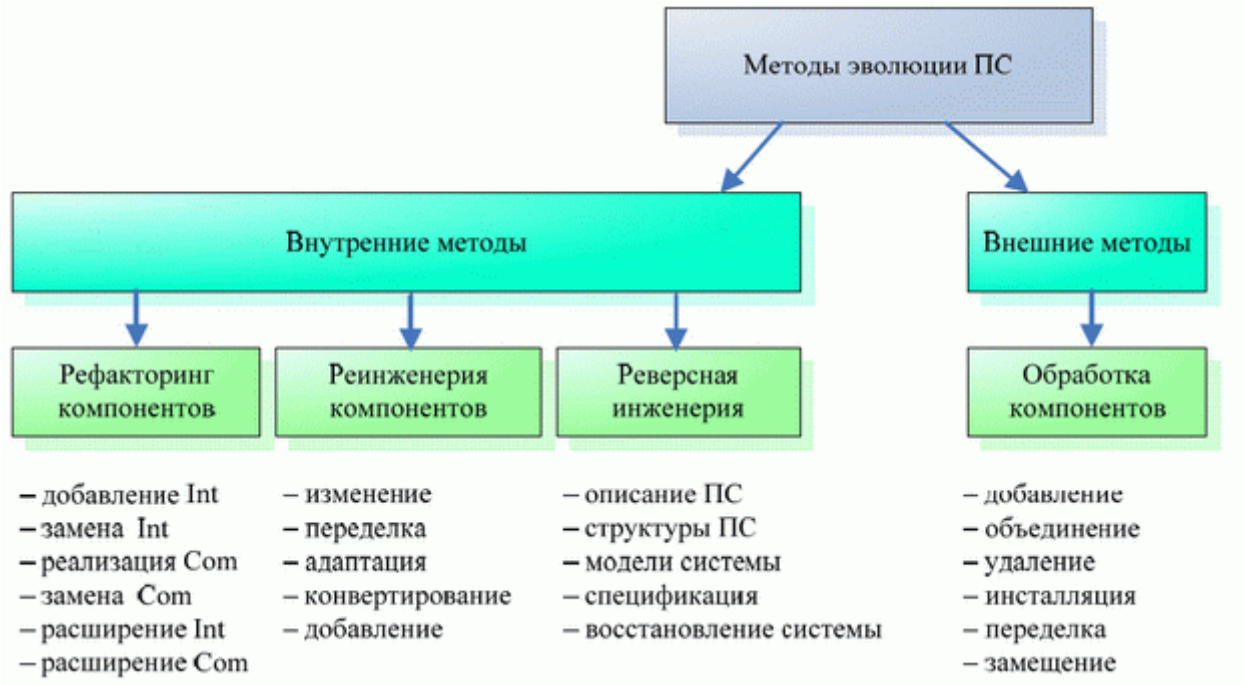
стареющих программ. С теоретической точки зрения эти методы изучены недостаточно, а с практической точки зрения многие программисты решают задачи внесения изменений в ПС постоянно.

Например, широкий круг специалистов затронула проблема изменения формата даты в 2000 году. Для систематической переделки функционирующих программ к новым возможностям ОС, языков и платформ современных компьютеров и т. п. используются современный *аутсорсинг* (Индия, Россия, Украина и др.).

Внесение изменений в *ПО* можно рассматривать как эволюционный *путь* его развития. *Эволюция ПО* осуществляетсявнешними методами обработки компонентов в распределенной среде и внутренними методами, как изменение компонентов (COM), интерфейсов (*Int*) и/или систем. К внутренним методам эволюции отнесены методы реинженерии, рефакторинга и реверсной инженерии (рис. 8.10).

Эти методы обеспечивают разноплановое изменение программ или систем.

К ним относятся корректировка спецификаций, документации и программного кода в соответствии с требованиями на изменения [8.15–8.20].



увеличить изображение

Рис. 8.10. Схема методов эволюции компонентов ПС

Суть этих методов состоит в следующем:

реинженерия обеспечивает перепрограммирование отдельных компонентов в новые ЯП, платформы и среды, а также расширение возможностей ПС;

рефакторинг обеспечивает внесение изменений в компоненты или интерфейсы (добавление, расширение и т. д.), добавление экземпляров компонентов, новых функций или системных сервисов; – реверсная инженерия означает полную переделку компонентов, а иногда и перепрограммирование всей системы.

8.4.1. Реинженерия программных систем

Реинженерия (reengineering) – это эволюция программы (системы) путем ее изменения в целях повышения удобства ее эксплуатации, сопровождения или изменения ее функций. Она включает в себя процессы реорганизации и реструктуризации системы, перевода отдельных компонентов системы в другой, более современный ЯП, а также процессы модификации или модернизации структуры и системы данных. При этом архитектура системы может оставаться неизменной.

Метод реинженерии – целевое средство получения нового компонента путем выполнения последовательности операций внесения изменений, модернизации или модификации, а также перепрограммирования отдельных компонентов ПС. Реализуется совокупностью моделей, методов и процессов, изменяющих структуру и возможности компонентов с целью получения компонента с новыми возможностями. Новые компоненты идентифицируются именами, которые используются при создании компонентных конфигураций и каркасов системы.

С технической точки зрения реинженерия – это решение проблемы эволюции системы путем изменения ее архитектуры в измененной среде, в которой компоненты размещаются на разных компьютерах. Причиной эволюции может быть изменение ЯП системы, например, Fortran, Cobol и др. с переходом на современные объектно-ориентированные языки, такие, как Java или C++.

Однако с коммерческой точки зрения реинженерию принимают часто за единственный способ сохранения наследуемых систем в эксплуатации. Полная эволюция системы – дорогостоящая либо рискованная процедура продления времени существования системы.

По сравнению с более радикальными подходами к совершенствованию систем реинженерия имеет следующие преимущества.

1. Снижение риска при повторной разработке ПС. В то же время существует риск получения неудовлетворительного результата при внесении ошибок в спецификации или при изменении функциональности некоторых программ. Снизить возникающие риски можно за счет удаления ошибок и улучшения качества работы измененных программ.
2. Снижение затрат за счет использования компонентов повторного использования при разработке новой ПС. Согласно данным различных коммерческих структур повторное использование в четыре раза дешевле, чем новая разработка системы.

Реинженерия применяется для изменения деловых процессов, снижения количества излишних видов деятельности в них и повышения эффективности отдельных деловых процессов за счет внедрения новых программ или модификации существующих программ. Если бизнеспроцесс зависит от наследуемой системы, то изменения в нее должны планироваться.

Основное различие между реинженерией и новой разработкой системы состоит в том, что написание системной спецификации начинается не с "нуля", а с рассмотрения возможностей старой наследуемой системы.

К основным этапам процесса реинженерии относятся:

перевод исходного кода в старом ЯП на современную версию этого языка либо в другой ЯП;

анализ программ по документированной структуре и функциональных возможностей системы;

модификация структуры программ для наращивания новых свойств и возможностей;

разбиение системы на модули для их группирования и устранения избыточности;

изменение данных, с которыми работает программа, с учетом проведенных изменений в программе.

Причинами, требующими преобразование исходного кода программ в другой язык, могут быть:

обновление платформы аппаратных средств, на которой может не выполняться компилятор ЯП;

недостаток квалифицированного персонала для программ, написанных в ЯП, вышедших из употребления;

изменение структуры программы в связи с переходом на новый стандартный язык программирования.

К операциям реинженерии относятся:

именование компонентов и их идентификация;

расширение функций существующей реализации компонентов;

перевод языка компонента в новый современный ЯП;

реструктуризация структуры компонента;

модификация описания компонента и его данных.

8.4.2. Рефакторинг компонентов

Рефакторинг получил развитие в объектно-ориентированном программировании в связи с широким применением интерфейсов, шаблонов проектирования и методов улучшения кода [8.5]. Разработаны библиотеки типовых трансформаций искомых объектов (классов), которые улучшают те или иные характеристики ПС.

Метод рефакторинга компонента – это целевой способ получения нового компонента на базе существующего, который включает операции модификации (изменение, замещение, расширение) компонентов и интерфейсов. Цель метода – преобразование состава компонентов ПС или изменение отдельного компонента системы для придания ему новых функциональных и структурных характеристик, удовлетворяющих требованиям конфигурации. Метод включает совокупность моделей, методов и процессов, применяемых к определенным классам объектов и компонентам для получения новых или измененных объектовкомпонентов с целью повышения качественных характеристик ПС или добавление новых возможностей.

Процесс рефакторинга может быть ориентирован на получение новых компонентов, которые включают следующие операции по организации проведения изменений:

добавление новой реализации для существующего и нового интерфейса;

замена существующей реализации новой с эквивалентной функциональностью;

добавление нового интерфейса (при наличии соответствующей реализации);

расширение существующего интерфейса для новых системных сервисов в компонентной среде.

Каждая операция рефакторинга – базовая, атомарная функция преобразования, сохраняющая целостность компонента, т. е. правила, ограничения и зависимости между составными элементами компонента, позволяющие рассматривать компонент как единую и цельную структуру со своими свойствами и характеристиками.

После выполнения операций рефакторинга компоненты должны быть идентичны функциям исходного компонента. В случае коренного изменения группы компонентов системы путем внесения новых функций система приобретает новую функциональность.

Операции над компонентами удовлетворяют условиям:

- объект, полученный в результате рефакторинга, – это компонент с соответствующими свойствами, характеристиками и типичной структурой;
- операция не изменяет функциональность компонента и новый компонент может применяться в ранее построенных компонентных системах;
- перестройка компонентов, а иногда и перепрограммирования проводится в процессе реверсной инженерии [8.15, 8.18].

8.4.3. Реверсная инженерия

Методы реверсной инженерии, которые разработаны в среде объектно-ориентированного программирования, базируются на выполнении базовых операций визуализации (visual) и измерения метрик (metric) ПС в рамках модели, которая предлагает следующие цели:

- обеспечение высокого качества системы и переосвидетельствование ее размера, сложности и структуры;
- поиск иерархии классов и атрибутов программных объектов с целью наследования их в ядре системы;
- идентификация классов объектов с определением размера и/или сложности всех классов системы;
- поиск паттернов, их идентификация, а также фиксация их места и роли в структуре системы.

Этот подход ориентирован на индустриальные системы в миллион строк кода с использованием метрических оценок характеристик системы. Он разрешает генерацию тестов для проверки кодов, а также проведение метрического анализа системы для получения фактических значений внутренних и внешних характеристик системы [8.20].

В результате анализа системы строится модель, которая содержит список классов и паттернов системы, которые могут модифицироваться и перепроектироваться и тем самым составлять процесс эволюции системы. Если некоторый класс плохо спроектирован (например, много методов, пустые коды) или система не выполняет требуемую работу, то проводится сбор информации для изменения модели системы. В данном подходе действия по визуализации системы отражаются на экране в виде иерархического дерева, узлы которого отображают объекты и их свойства, а отношения задаются контурами команд фрагментов программ. При этом применяется таблица метрик, в которой находятся сведения о метриках классов объектов (число классов, атрибутов, подклассов и строк кода), метрик методов объектов (количество параметров, вызовов, сообщений и т. п.), метрик атрибутов объектов (время доступа, количество доступов в классе и т. п.).

В процессе визуализации ведется сбор метрических данных о системе. Если реально определены все данные в разных фактических метриках ПС, выполняются оценка качества и разрабатывается план перестройки устаревшей системы на новую систему с получением тех же возможностей или еще и дополнительных.

Таким образом, рассмотрены базовые понятия интерфейса, подходы к обеспечению интерфейса языков программирования 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, стандарты преобразования форматов данных и эволюция программных систем.

Контрольные вопросы и задания

1. Определите цели и задачи интерфейса в программной инженерии.
2. Назовите системы, которые основываются на интерфейсах и обеспечивают преобразование данных.
3. Охарактеризуйте кратко современные распределенные системы (например, CORBA).
4. Назовите методы вызова компонентов в распределенных средах.
5. Определите формальную схему взаимодействия программ.
6. Определите основные задачи интерфейса ЯП.
7. Назовите современные подходы к взаимодействию разноязыковых программ.
8. Определите проблемы преобразования форматов данных.

9. Какие методы преобразования данных БД существуют?Определите цели и задачи изменения ПС при сопровождении.
10. Дайте краткую характеристику проблем, возникающих при сопровождении системы.
11. Определите основные задачи реинженерии ПС.
12. Чем отличается рефакторинг компонентов от реинженерии?
13. Определите основные операции реверсной инженерии ПС.

Внимание! Если Вы увидите ошибку на нашем сайте, выделите её и нажмите Ctrl+Enter.