Б А К А Л А В Р И А Т

Г.С. Иванова

ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

Допущено
УМО по университетскому политехническому образованию
в качестве **учебника** для студентов высших учебных заведений,
обучающихся по направлению «Информатика и вычислительная техника»

Третье издание, стереотипное

КНОРУС • МОСКВА • 2022

УДК 681.3.06(075.8)
ББК 32.973-018я73
И21

Рецензенты:
Е.В. Юркевич, заведующий лабораторией Института проблем управления им. В.А. Трапезникова РАН, д-р техн. наук, проф.,
В.М. Черненький, заведующий кафедрой «Системы обработки информации и управления» МГТУ им. Н.Э. Баумана, д-р техн. наук, проф.

Иванова, Галина Сергеевна.
И21 Технология программирования : учебник / Г.С. Иванова. — 3-е изд., стер. — Москва : КНОРУС, 2022. — 336 с. — (Бакалавриат).

ISBN 978-5-406-10176-6

Подробно рассмотрены основные методы и нотации, применяемые при разработке сложного программного обеспечения. Особое внимание уделено проектированию программных систем с использованием структурного и объектного подходов. Детально разобраны основные приемы обеспечения требуемых технологических свойств. Приведена классификация и проанализированы принципы проектирования пользовательских интерфейсов программного обеспечения. Материал учебника проиллюстрирован большим количеством примеров, поясняющих рисунков и проектной документации.

Соответствует ФГОС ВО последнего поколения.

Для студентов вузов, которые обучаются по направлениям, предполагающим изучение технологии программирования. Полезен при оформлении документации к курсовым и дипломным работам и проектам, связанным с разработкой программного обеспечения. Может быть интересен всем изучающим программирование самостоятельно.

УДК 681.3.06(075.8)
ББК 32.973-018я73

Иванова Галина Сергеевна

ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

Изд. № 656996. Формат 60×90/16. Гарнитура «Times New Roman».
Усл. печ. л. 27,3. Уч.-изд. л. 27,0..Тираж 500 экз.

ООО «Издательство «КноРус».
117218, г. Москва, ул. Кедрова, д. 14, корп. 2.
Тел.: +7 (495) 741-46-28.
E-mail: welcome@knorus.ru www.knorus.ru

Отпечатано в АО «Т8 Издательские Технологии».
109316, г. Москва, Волгоградский проспект, д. 42, корп. 5.
Тел.: +7 (495) 221-89-80.

ISBN 978-5-406-10176-6

Оглавление

ПРЕДИСЛОВИЕ

До последнего времени элементы технологии разработки программного обеспечения студенты изучали в таких курсах, как «Алгоритмические языки и программирование» и «Системное программирование», параллельно с основным материалом, что не позволяло сконцентрироваться на указанных вопросах. Однако сравнительно недавно в учебных планах специальностей, связанных с информатикой, появился курс «Технология программирования», полностью посвященный этой теме.

Такое внимание к этой теме вызвано тем, что современное программирование – сложный производственный процесс, в котором участвует целый коллектив разработчиков: математиков, программистов, тестеровщиков и других специалистов. Организация взаимодействия между всеми участниками проекта требует максимально точного планирования работы и наличия соответствующей документации. Таким образом, любой член коллектива разработчиков программного обеспечения должен владеть основными технологическими знаниями в этой области. И чем объемнее и сложнее разрабатываемое программное обеспечение, тем больше специалистов участвует в его разработке и тем большее значение имеет правильная организация технологического процесса его создания.

Существенную роль играет также знание основных технологических приемов, позволяющих существенно сократить время разработки, а также улучшить качество создаваемого программного обеспечения. Такие приемы существуют практически для всех этапов разработки: от составления технического задания до тестирования готового продукта, и знание их абсолютно необходимо будущим специалистам в области создания сложного программного обеспечения.

В предлагаемом учебнике сделана попытка обобщения и методического осмысления опыта, накопленного специалистами в области разработки программного обеспечения на протяжении всей истории существования. Имеющийся опыт позволяет превратить плохо управляемый «творческий» процесс создания программного обеспечения в хорошо организованное производство технологически качественных программных продуктов.

ВВЕДЕНИЕ

Создание программной системы – весьма трудоемкая задача, особенно в наше время, когда обычный объем программного обеспечения превышает сотни тысяч операторов. Будущий специалист в области разработки программного обеспечения должен иметь представление о методах анализа, проектирования, реализации и тестирования программных систем, а также ориентироваться в существующих подходах и технологиях.

Изложение материала учебника строится в соответствии с основными этапами разработки программного обеспечения. Исключением являются первые главы, в которых рассмотрены общие вопросы технологии программирования.

В главе 1 проанализирована история развития технологии программирования, показано, что в основе разработки программного обеспечения лежит блочно-иерархический подход, рассмотрены особенности применения этого подхода к разработке программных продуктов.

Глава 2 содержит описание приемов обеспечения качества программного обеспечения: основных положений структурного, модульного и защитного программирования. В ней также приведены некоторые рекомендации, например по стилю оформления программ.

В главе 3 рассматриваются проблемы, связанные с постановкой задачи: от классификации программных продуктов до разработки технического задания и принятия основных решений начального этапа проектирования, например выбора подхода, среды и языка программирования.

Главы 4, 5 посвящены особенностям разработки программного обеспечения при структурном подходе; четвертая – анализу различных моделей разрабатываемого программного обеспечения, используемых на этапе уточнения спецификаций, а пятая – методикам проектирования.

Главы 6, 7 содержат аналогичный материал для объектного подхода. В качестве основного языка описания моделей анализа и проектирования при объектном подходе используется UML как мощное и практически стандартное средство описания объектных разработок.

В главе 8 подробно рассмотрены проблемы проектирования пользовательского интерфейса и предлагаются соответствующие модели.

В главе 9 описан пример проектирования небольшой программной системы Записная книжка. Этот пример может послужить основой для написания расчетно-пояснительной записки по курсовой или дипломной работам или проектам.

Глава 10 посвящена тестированию программных продуктов как по частям, так и в целом, одиннадцатая – методам, средствам и методикам отладки разрабатываемого программного обеспечения.

В главе 11 рассмотрены методы и приведена методика отладки программного обеспечения.

В главе 12 приведены сведения и рекомендации по разработке программной документации.

Материал сопровождается большим количеством сравнительно простых примеров, причем по возможности использованы три примера разработки, для которых рассмотрены различные аспекты проектирования.

Курс обучения целесообразно завершать курсовым проектом или курсовой работой, целью которых должно быть создание небольшого, но завершенного программного продукта (в отличие от небольших и, как правило, недокументированных программ, которые студенты пишут на лабораторных работах при изучении основ программирования и (или) конкретных языков). Проект должен начинаться с составления и утверждения технического задания и сопровождаться подготовкой необходимой программной документации.

ГЛАВА 1. ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ. ОСНОВНЫЕ ПОНЯТИЯ И ПОДХОДЫ

Программирование – сравнительно молодая и быстро развивающаяся отрасль науки и техники. Опыт ведения реальных разработок и совершенствования имеющихся программных и технических средств постоянно переосмысливается, в результате чего появляются новые методы, методологии и технологии, которые в свою очередь служат основой более современных средств разработки программного обеспечения. Исследовать процессы создания новых технологий и определять их основные тенденции целесообразно, сопоставляя эти технологии с уровнем развития программирования и особенностями имеющихся в распоряжении программистов программных и аппаратных средств.

1.1. Технология программирования и основные этапы ее развития

Технологией программирования называют совокупность методов и средств, используемых в процессе разработки программного обеспечения. Как любая другая технология, технология программирования представляет собой набор технологических инструкций, включающих:

- указание последовательности выполнения технологических операций;
- перечисление условий, при которых выполняется та или иная операция;
- описания самих операций, где для каждой операции определены исходные данные, результаты, а также инструкции, нормативы, стандарты, критерии и методы оценки и т. п. (рис. 1.1).

Кроме набора операций и их последовательности технология также определяет способ описания проектируемой системы, точнее модели, используемой на конкретном этапе разработки.

Различают технологии, используемые на конкретных этапах разработки или для решения отдельных задач этих этапов, и технологии, охватывающие несколько этапов или весь процесс разработки. В основе первых, как правило, лежит ограниченно применимый *метод*, позволяющий решить конкретную задачу. В основе вторых обычно лежит базовый метод или *подход*, опре-

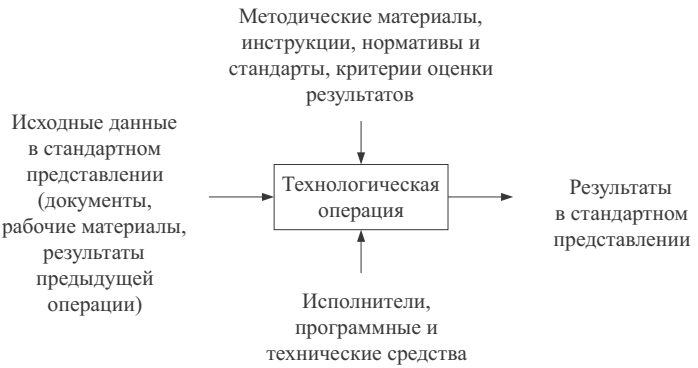


Рис. 1.1. Структура описания технологической операции

деляющий совокупность методов, используемых на разных этапах разработки, или *методологию*.

Чтобы разобраться в существующих технологиях программирования и определить основные тенденции их развития, целесообразно рассматривать эти технологии в историческом контексте, выделяя основные этапы развития программирования как науки.

Первый этап – «стихийное» программирование. Этот этап охватывает период от момента появления первых вычислительных машин до середины 60-х годов XX в. В этот период практически отсутствовали сформулированные технологии, и программирование фактически было искусством. Первые программы имели простейшую структуру. Они состояли из собственно программы на машинном языке и обрабатываемых ею данных (рис. 1.2). Сложность программ в машинных кодах ограничивалась способностью программиста одновременно мысленно отслеживать последовательность выполняемых операций и местонахождение данных при программировании.

Появление ассемблеров позволило вместо двоичных или 16-ричных кодов использовать символические имена данных и мнемоники кодов операций. В результате программы стали более «читаемыми».

Создание языков программирования высокого уровня, таких как FORTRAN и ALGOL, существенно упростило программирование вычислений, снизив уровень детализации операций. Это в свою очередь позволило увеличить сложность программ.

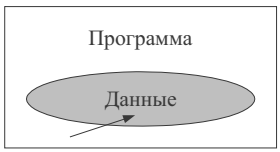


Рис. 1.2. Структура первых программ

Революционным было появление в языках средств, позволяющих оперировать подпрограммами. (Идея написания подпрограмм появилась

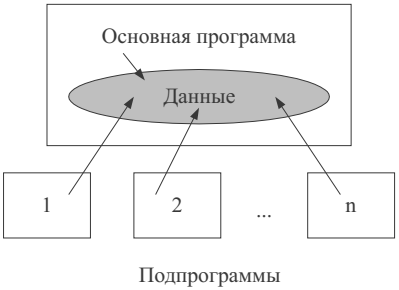


Рис. 1.3. Архитектура программы с глобальной областью данных

гораздо раньше, но отсутствие средств поддержки в первых языковых средствах существенно снижало эффективность их применения.) Подпрограммы можно было сохранять и использовать в других программах. В результате были созданы огромные библиотеки расчетных и служебных подпрограмм, которые по мере надобности вызывались из разрабатываемой программы.

Типичная программа того времени состояла из основной программы, области *глобальных данных* и набора подпрограмм (в основном библиотечных), выполняющих обработку всех данных или их части (рис. 1.3).

Слабым местом такой архитектуры было то, что при увеличении количества подпрограмм возрастала вероятность искажения части глобальных данных какой-либо подпрограммой. Например, подпрограмма поиска корней уравнения на заданном интервале по методу деления отрезка пополам меняет величину интервала. Если при выходе из подпрограммы не предусмотреть восстановления первоначального интервала, то в глобальной области окажется неверное значение интервала. Чтобы сократить количество таких ошибок, было предложено в подпрограммах размещать *локальные данные* (рис. 1.4).

Сложность разрабатываемого программного обеспечения при использовании подпрограмм с локальными данными по-прежнему ограничивалась возможностью программиста отслеживать процессы обработки данных, но уже на новом уровне. Однако появление средств поддержки подпрограмм позволило осуществлять разработку программного обеспечения нескольким программистам параллельно.

В начале 60-х годов XX в. разразился «кризис программирования». Он выражался в том, что фирмы, взявшиеся за разработку сложного программного обеспечения, такого как операционные системы, срывали все сроки завершения проектов [8]. Проект устаревал

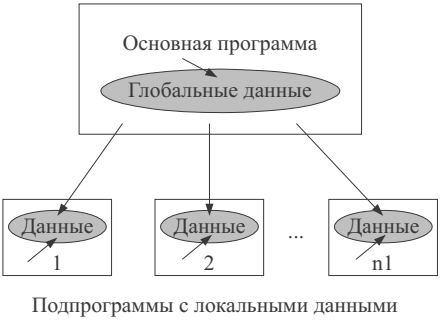


Рис. 1.4. Архитектура программы, использующей подпрограммы с локальными данными

раньше, чем был готов к внедрению, увеличивалась его стоимость, и в результате многие проекты так никогда и не были завершены.

Объективно все это было вызвано несовершенством технологии программирования. Прежде всего стихийно использовалась разработка «снизу-вверх» – подход, при котором вначале проектировали и реализовывали сравнительно простые подпрограммы, из которых затем пытались построить сложную программу. В отсутствие четких моделей описания подпрограмм и методов их проектирования создание каждой подпрограммы превращалось в непростую задачу, интерфейсы подпрограмм получались сложными, и при сборке программного продукта выявлялось большое количество ошибок согласования. Исправление таких ошибок, как правило, требовало серьезного изменения уже разработанных подпрограмм, что еще более осложняло ситуацию, так как при этом в программу часто вносились новые ошибки, которые также необходимо было исправлять... В конечном итоге процесс тестирования и отладки программ занимал более 80% времени разработки, если вообще когда-нибудь заканчивался. На повестке дня самым серьезным образом стоял вопрос разработки технологии создания сложных программных продуктов, снижающей вероятность ошибок проектирования.

Анализ причин возникновения большинства ошибок позволил сформулировать новый подход к программированию, который был назван «структурным» [18, 22].

Второй этап – структурный подход к программированию (60–70-е годы XX в.). *Структурный подход к программированию* представляет собой совокупность рекомендуемых технологических приемов, охватывающих выполнение всех этапов разработки программного обеспечения. В основе структурного подхода лежит *декомпозиция* (разбиение на части) сложных систем с целью последующей реализации в виде отдельных небольших (до 40–50 операторов) подпрограмм. С появлением других принципов декомпозиции (объектного, логического и т. д.) данный способ получил название *процедурной* декомпозиции.

В отличие от используемого ранее процедурного подхода к декомпозиции, структурный подход требовал представления задачи в виде иерархии подзадач простейшей структуры. Проектирование, таким образом, осуществлялось «сверху-вниз» и подразумевало реализацию общей идеи, обеспечивая проработку интерфейсов подпрограмм. Одновременно вводились ограничения на конструкции алгоритмов, рекомендовались формальные модели их описания, а также специальный метод проектирования алгоритмов – метод пошаговой детализации.

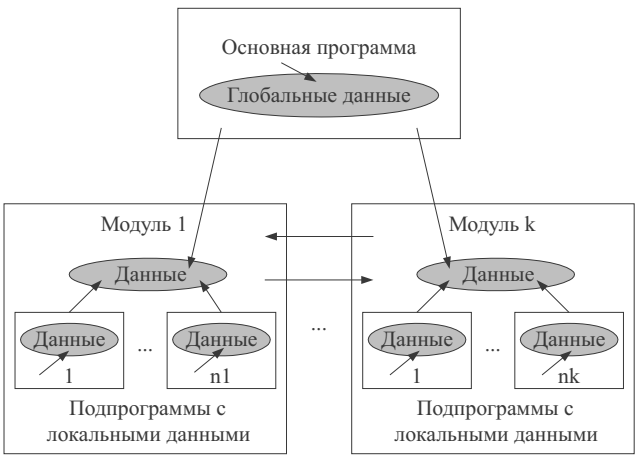
Поддержка принципов структурного программирования была заложена в основу так называемых *процедурных* языков программирования. Как правило, они включали основные «структурные» операторы передачи управления, поддерживали вложение подпрограмм, локализацию и ограничение об-

ласти «видимости» данных. Среди наиболее известных языков этой группы стоит назвать PL/1, ALGOL-68, Pascal, C.

Одновременно со структурным программированием появилось огромное количество языков, базирующихся на других концепциях, но большинство из них не выдержало конкуренции. Какие-то языки были просто забыты, идеи других были в дальнейшем использованы в следующих версиях развиваемых языков.

Дальнейший рост сложности и размеров разрабатываемого программного обеспечения потребовал развития *структурирования данных*. Как следствие этого в языках появляется возможность определения пользовательских типов данных. Одновременно усилилось стремление разграничить доступ к глобальным данным программы, чтобы уменьшить количество ошибок, возникающих при работе с глобальными данными. В результате появилась и начала развиваться технология модульного программирования.

Модульное программирование предполагает выделение групп подпрограмм, использующих одни и те же глобальные данные в отдельно компилируемые *модули* (библиотеки подпрограмм), например модуль графических ресурсов, модуль подпрограмм вывода на принтер (рис. 1.5). Связи между модулями при использовании данной технологии осуществляются через специальный интерфейс, в то время как доступ к реализации модуля (телам подпрограмм и некоторым «внутренним» переменным) запрещен. Эту технологию поддерживают современные версии языков Pascal и C (C++), языки Ада и Modula.



Модули с локальными данными и попрограммами

Рис. 1.5. Архитектура программы, состоящей из модулей

Использование модульного программирования существенно упростило разработку программного обеспечения несколькими программистами. Теперь каждый из них мог разрабатывать свои модули независимо, обеспечивая взаимодействие модулей через специально оговоренные межмодульные интерфейсы. Кроме того, модули в дальнейшем без изменений можно было использовать в других разработках, что повысило производительность труда программистов.

Практика показала, что структурный подход в сочетании с модульным программированием позволяет получать достаточно надежные программы, размер которых *не превышает 100 000 операторов* [10]. Узким местом модульного программирования является то, что ошибка в интерфейсе при вызове подпрограммы выявляется только при выполнении программы (из-за раздельной компиляции модулей обнаружить эти ошибки раньше невозможно). При увеличении размера программы обычно возрастает сложность межмодульных интерфейсов, и с некоторого момента предусмотреть взаимовлияние отдельных частей программы становится практически невозможно. Для разработки программного обеспечения большого объема было предложено использовать *объектный подход*.

Третий этап – объектный подход к программированию (с середины 80-х до конца 90-х годов XX в.). *Объектно-ориентированное программирование* определяется как технология создания сложного программного обеспечения, основанная на представлении программы в виде совокупности *объектов*, каждый из которых является экземпляром определенного типа (*класса*), а классы образуют иерархию с *наследованием* свойств [10, 23, 28]. Взаимодействие программных объектов в такой системе осуществляется путем передачи *сообщений* (рис. 1.6).

Объектная структура программы впервые была использована в языке имитационного моделирования сложных систем Simula, появившемся еще в 60-х годах XX в. Естественный для языков моделирования способ представления программы получил развитие в другом специализированном языке моделирования – языке Smalltalk (70-е годы XX в.), а затем был использован в новых версиях универсальных языков программирования, таких как Pascal, C++, Modula, Java.

Основным достоинством объектно-ориентированного программирования по сравнению с модульным программированием является «более естественная» декомпозиция программного обеспечения, которая существенно облегчает его разработку. Это приводит к более полной локализации данных и интегрированию их с подпрограммами обработки, что позволяет вести практически независимую разработку отдельных частей (объектов) программы. Кроме того, объектный подход предлагает новые способы организации программ, основанные на механизмах наследования, полиморфизма, композиции, наполнения. Эти механизмы позволяют конструировать сложные объекты из сравнительно простых. В результате существенно увеличивается пока-

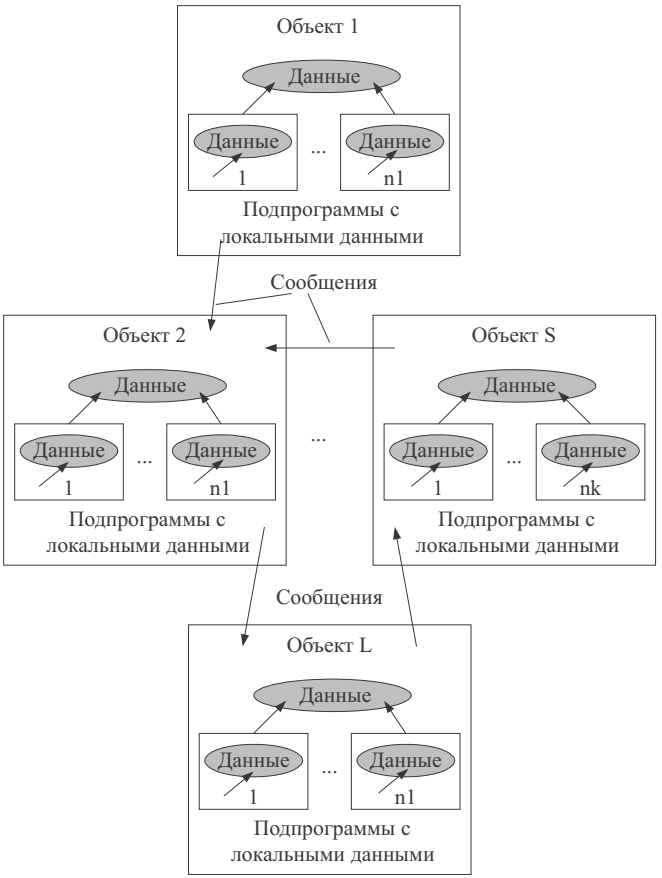


Рис. 1.6. Архитектура программы
при объектно-ориентированном программировании

затель повторного использования кодов и появляется возможность создания библиотек классов для различных применений.

Бурное развитие технологий программирования, основанных на объектном подходе, позволило решить многие проблемы. Так были созданы среды, поддерживающие *визуальное программирование*, например Delphi, C++ Builder, Visual C++ и т. д. При использовании визуальной среды у программиста появляется возможность проектировать некоторую часть, например, интерфейсы будущего продукта, с применением визуальных средств добавления и настройки специальных библиотечных компонентов. Результатом

визуального проектирования является заготовка будущей программы, в которую уже внесены соответствующие коды.

Использование объектного подхода имеет много преимуществ, однако его конкретная реализация в объектно-ориентированных языках программирования, таких как Pascal и C++, имеет существенные недостатки:

• фактически отсутствуют стандарты компоновки двоичных результатов компиляции объектов в единое целое даже в пределах одного языка программирования: компоновка объектов, полученных разными компиляторами C++ в лучшем случае проблематична, что приводит к необходимости разработки программного обеспечения с использованием средств и возможностей одного языка программирования высокого уровня и одного компилятора, а значит, требует наличия исходных кодов используемых библиотек классов;

• изменение реализации одного из программных объектов, как минимум, связано с перекомпиляцией соответствующего модуля и перекомпоновкой всего программного обеспечения, использующего данный объект.

Таким образом, при использовании этих языков программирования сохраняется зависимость модулей программного обеспечения от адресов экспортируемых полей и методов, а также структур и форматов данных. Эта зависимость объективна, так как модули должны взаимодействовать между собой, обращаясь к ресурсам друг друга. Связи модулей нельзя разорвать, но можно попробовать стандартизировать их взаимодействие, на чем и основан компонентный подход к программированию.

Четвертый этап – компонентный подход и CASE-технологии (с середины 90-х годов XX в. до нашего времени). *Компонентный подход* предполагает построение программного обеспечения из отдельных компонентов – физически отдельно существующих частей программного обеспечения, которые взаимодействуют между собой через *стандартизованные двоичные интерфейсы.* В отличие от обычных объектов объекты-компоненты можно собрать в динамически вызываемые библиотеки или исполняемые файлы, распространять в двоичном виде (без исходных текстов) и использовать в любом языке программирования, поддерживающем соответствующую технологию. На сегодня рынок объектов стал реальностью, так в Интернете существуют узлы, предоставляющие большое количество компонентов, рекламой компонентов забиты журналы. Это позволяет программистам создавать продукты, хотя бы частично состоящие из повторно использованных частей, т. е. использовать технологию, хорошо зарекомендовавшую себя в области проектирования аппаратуры.

Компонентный подход лежит в основе технологий, разработанных на базе COM (Component Object Model – компонентная модель объектов), и технологии создания распределенных приложений CORBA (Common Object Request Broker Architecture – общая архитектура с посредником обработки запросов объектов). Эти технологии используют сходные принципы и различаются лишь особенностями их реализации.

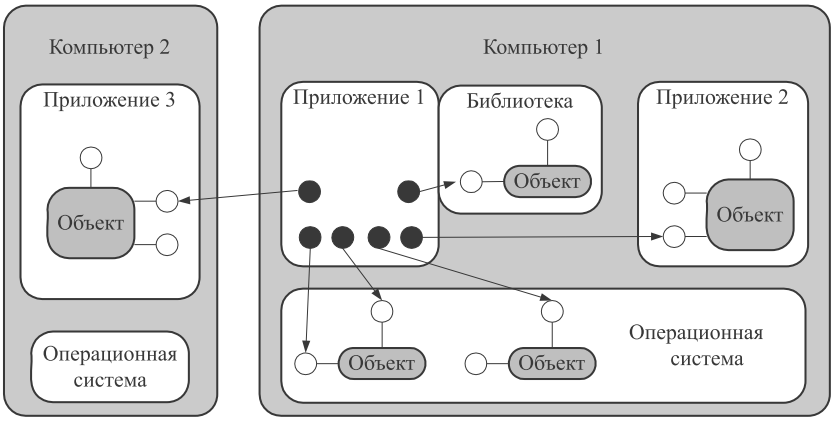


Рис. 1.7. Взаимодействие программных компонентов различных типов

Технология COM фирмы Microsoft является развитием технологии OLE 1 (Object Linking and Embedding – связывание и внедрение объектов), которая использовалась в ранних версиях Windows для создания составных документов. Технология COM определяет *общую парадигму взаимодействия программ любых типов*: библиотек, приложений, операционной системы, т. е. позволяет одной части программного обеспечения использовать функции (*службы*), предоставляемые другой, независимо от того, функционируют ли эти части в пределах одного процесса, в разных процессах на одном компьютере или на разных компьютерах (рис. 1.7). Модификация COM, обеспечивающая передачу вызовов между компьютерами, называется DCOM (Distributed COM – распределенная COM).

По технологии COM приложение предоставляет свои службы, используя специальные объекты – *объекты COM*, которые являются экземплярами *классов COM*. Объект COM так же, как обычный объект, включает поля и методы, но в отличие от обычных объектов каждый объект COM может реализовывать несколько интерфейсов, обеспечивающих доступ к его полям и функциям. Это достигается за счет организации отдельной таблицы адресов методов для каждого интерфейса (по типу таблиц виртуальных методов). При этом интерфейс обычно объединяет несколько однотипных функций. Кроме того, классы COM поддерживают *наследование интерфейсов*, но не поддерживают *наследования реализации*, т. е. не наследуют код методов, хотя при необходимости объект класса-потомка может вызвать метод родителя.

Каждый интерфейс имеет имя, начинающееся с символа «I» и глобальный уникальный идентификатор IID (Interface IDentifier). Любой объект COM обязательно реализует интерфейс IUnknown (на схемах этот интерфейс

всегда располагают сверху). Использование этого интерфейса позволяет получить доступ к остальным интерфейсам объекта.

Объект всегда функционирует в составе *сервера* – динамической библиотеки или исполняемого файла, которые обеспечивают функционирование объекта. Различают три типа серверов:

- внутренний сервер – реализуется динамическими библиотеками, которые подключаются к приложению-клиенту и работают в одном с ними адресном пространстве – наиболее эффективный сервер, кроме того, он не требует специальных средств;
- локальный сервер – создается отдельным процессом (модулем. exe), который работает на одном компьютере с клиентом;
- удаленный сервер – создается процессом, который работает на другом компьютере.

Например, Microsoft Word является локальным сервером. Он включает множество объектов, которые могут использоваться другими приложениями.

Для обращения к службам клиент должен получить указатель на соответствующий интерфейс. Перед первым обращением к объекту клиент посылает запрос к библиотеке COM, хранящей информацию обо всех, зарегистрированных в системе классах COM объектов, и передает ей имя класса, идентификатор интерфейса и тип сервера. Библиотека запускает необходимый сервер, создает требуемые объекты и возвращает указатели на объекты и интерфейсы. Получив указатели, клиент может вызывать необходимые функции объекта.

Взаимодействие клиента и сервера обеспечивается базовыми механизмами COM или DCOM, поэтому клиенту безразлично местонахождение объекта. При использовании локальных и удаленных серверов в адресном пространстве клиента создается *proxy-объект* – заместитель объекта COM, а в адресном пространстве сервера COM – *заглушка*, соответствующая клиенту. Получив задание от клиента, заместитель упаковывает его параметры и, используя службы операционной системы, передает вызов заглушке. Заглушка распаковывает задание и передает его объекту COM. Результат возвращается клиенту в обратном порядке.

На базе технологии COM и ее распределенной версии DCOM были разработаны компонентные технологии, решающие различные задачи разработки программного обеспечения.

OLE-automation или просто Automation (автоматизация) – технология создания программируемых приложений, обеспечивающая программируемый доступ к внутренним службам этих приложений. Вводит понятие *диспинтерфейса* (dispinterface) – специального интерфейса, облегчающего вызов функций объекта. Эту технологию поддерживает, например, Microsoft Excel, предоставляя другим приложениям свои службы.

ActiveX – технология, построенная на базе OLE-automation, предназначена для создания программного обеспечения как сосредоточенного на од-

ном компьютере, так и распределенного в сети. Предполагает использование визуального программирования для создания компонентов – элементов управления ActiveX. Полученные таким образом элементы управления можно устанавливать на компьютер дистанционно с удаленного сервера, причем устанавливаемый код не зависит от используемой операционной системы. Это позволяет применять элементы управления ActiveX в клиентских частях приложений Интернет.

Основными преимуществами технологии ActiveX, обеспечивающими ей широкое распространение, являются:

- быстрое написание программного кода – поскольку все действия, связанные с организацией взаимодействия сервера и клиента берет на программное обеспечение COM, программирование сетевых приложений становится похожим на программирование для отдельного компьютера;
- открытость и мобильность – спецификации технологии недавно были переданы в Open Group как основа открытого стандарта;
- возможность написания приложений с использованием знакомых средств разработки, например Visual Basic, Visual C++, Borland Delphi, Borland C++ и любых средств разработки на Java;
- большое количество уже существующих бесплатных программных элементов ActiveX (к тому же, практически любой программный компонент OLE совместим с технологиями ActiveX и может применяться без модификаций в сетевых приложениях);
- стандартность – технология ActiveX основана на широко используемых стандартах Internet (TCP/IP, HTML, Java), с одной стороны, и стандартах, введенных в свое время Microsoft и необходимых для сохранения совместимости (COM, OLE).

MTS (Microsoft Transaction Server – сервер управления транзакциями) – технология, обеспечивающая безопасность и стабильную работу распределенных приложений при больших объемах передаваемых данных.

MIDAS (Multitier Distributed Application Server – сервер многозвенных распределенных приложений) – технология, организующая доступ к данным разных компьютеров с учетом балансировки нагрузки сети.

Все указанные технологии реализуют компонентный подход, заложенный в COM. Так, с точки зрения COM элемент управления ActiveX – внутренний сервер, поддерживающий технологию OLE-automation. Для программиста же элемент ActiveX – «черный ящик», обладающий свойствами, методами и событиями, который можно использовать как строительный блок при создании приложений.

Технология CORBA, разработанная группой компаний OMG (Object Management Group – группа внедрения объектной технологии программирования), реализует подход, аналогичный COM, на базе объектов и интерфейсов CORBA. Программное ядро CORBA реализовано для всех основных аппаратных и программных платформ и потому эту технологию