Федеральное агентство по образованию

Санкт-Петербургский государственный электротехнический
университет "ЛЭТИ"

В. А. Гладцын  К. В. Кринкин  В. В. Яновский

# СЕРВИС-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА

## СТАНДАРТЫ, АЛГОРИТМЫ, ПРОТОКОЛЫ

Учебное пособие

Санкт-Петербург
Издательство СПбГЭТУ "ЛЭТИ"
2006

УДК 681.3
ББК 3973.23
Г 47

Гладцын В. А., Кринкин К. В., Яновский В. В. Сервис-ориентированная архитектура. Стандарты, алгоритмы, протоколы: Учеб. пособие. СПб.: Изд-во СПбГЭТУ "ЛЭТИ", 2004. 108 с.
Г 47
ISBN 5-7629-0714-7

Содержит теоретические сведения о принципах организации распределенных систем на основе сервис-ориентированной архитектуры. Рассматриваются тенденции развития распределенных систем, их основные характеристики и свойства. Приведена систематическая информация по моделям СОА включая модели, ориентированные на сервисы, ресурсы, сообщения, модели политик и управления. Уделено внимание стандартам и протоколам, используемым при построении распределенных СОА-систем.

Предназначено студентам специальности 220400 "Программное обеспечение вычислительной техники и автоматизированных систем" направлений 510200 "Прикладная математика и информатика" и 552800 "Информатика и вычислительная техника", изучающим дисциплины "Современные компьютерные технологии", "Сетевые технологии", "Сети ЭВМ и телекоммуникации".

УДК 681.3
ББК 3973.23

Рецензенты: кафедра цифровой вычислительной техники и информатики Санкт-Петербургского гос. ун-та телекоммуникаций; д-р техн. наук проф. Н. В. Егоров (СПбГУ).

Утверждено
редакционно-издательским советом университета
в качестве учебного пособия

ISBN 5-7629-0714-5

# ВВЕДЕНИЕ

Проблемы разработки эффективных систем обработки информации актуальны несколько десятилетий начиная с того момента, как появились первые действующие системы, решающие комплексные задачи. Спектр такого рода прикладных систем широк. Ключевыми направлениями являются как простые – автоматизация различных производственных процессов, выполнение распределенных вычислений в научных расчетах, моделирование различного рода систем и т. п., так и сложные – управление бизнесом или производством, глобальная синхронизация банковских операций и прочие. С самого начала создания подобных систем специалисты в области информационных технологий, пытаются разработать, с одной стороны, эффективные средства интеграции функциональных и структурных компонентов, а с другой – простые способы управления каждым из них в отдельности.

Прослеживая историческое развитие систем обработки информации, можно видеть, что они прошли путь от монолитных систем мейнфреймового типа, к двух- и трехуровневым архитектурам "клиент–сервер". Далее приобретают популярность многослойные вертикальные структуры; т. е. налицо тенденция все более узкой специализации и распределения отдельных компонентов систем. В последнее время активно обсуждаются вопросы организации распределенных систем с использованием глобальных сетей и web-технологий, но лишь с появлением ряда стандартов в области организации управления сервисами стало возможным говорить о новом направлении – сервис-ориентированной архитектуре (СОА).

Информация по сервис-ориентированной архитектуре в основном рассосредоточена по журнальным статьям и электронным публикациям (так так это довольно молодая область исследований), поэтому одной из главных задач авторов настоящего пособия являлась задача интеграции разрозненных сведений по рассматриваемому вопросу в рамках данного пособия.

Данное издание направлено на систематическое освещение вопросов построения распределенных систем с использованием сервис-ориентированной архитектуры. Особое внимание уделяется используемым моделям. Рассматриваются модели, ориентированные на сервисы, ресурсы, сообщения, модели политик и управления. Описываются основные стандарты и протоколы, а также приводятся примеры реализаций различных СОА-систем. Для многих рассматриваемых технологий проводится сравнительный анализ, позволяющий глубже понять рассматриваемый вопрос.

Структурно пособие включает в себя четыре части. В первой (разд. 1 и 2) рассмотрены тенденции развития систем обработки информации, приведены базовые сведения о структуре и функционировании распределенных систем.

Во второй части (разд. 3 и 4) приведены основные сведения о сервис-ориентированной архитектуре, рассмотрены используемые модели. Уделено внимание вопросам взаимодействия и организации совместной работы сервисов. Приведены формальные модели построения композитных приложений.

Третья часть пособия (разд. 5 и 6) посвящена стандартам в области СОА-систем. Приведены сведения об основополагающих стандартах: описания сервисов, организации брокеров и реестров. Рассмотрены стандартные языки и технологии описания бизнес-процессов, принципы разграничения доступа. В качестве примеров рассмотрены конкретные реализации систем на основе СОА.

Заключительная часть пособия является справочной. Она содержит список рекомендуемой литературы, которая может быть полезна при более глубоком изучении предмета, а также терминологический словарь, содержащий определения терминов и аббревиатур, используемых в области распределенных систем и технологий.

# 1. ТЕНДЕНЦИИ РАЗВИТИЯ РАСПРЕДЕЛЕННЫХ СИСТЕМ

Индустрия разработки программного обеспечения развивается уже в течение нескольких десятилетий. За это время поменялось множество технологий и принципов построения программных систем. Одни из них тесно связаны с компонентной базой, на которой строились вычислительные системы, другие – являются следствием борьбы с возрастающей сложностью самого программного обеспечения.

Для понимания современных принципов построения информационных систем необходимо знать, почему и каким образом одни архитектуры сменялись другими, а новые принципы построения программ приходили на смену старым подходам. Далее рассматривается генезис программных архитектур, в разное время имевших место в индустрии программного обеспечения корпоративных информационных систем, анализируются их достоинства и недостатки.

## 1.1. Монолитные системы

Архитектура первых программных систем была монолитной: весь программный код сосредоточивался в одном исполняемом модуле, и практически

не существовало его разделения по функциональности и условиям исполнения. По существу, такой подход к разработке программных систем не являлся архитектурой – это был исторически сложившийся метод разработки программ для ЭВМ.

В общем случае "структура" монолитной системы (рис. 1.1) представляет собой отсутствие структуры [1]. Система разрабатывается как набор процедур, каждая из которых может вызывать другие, когда ей это нужно. При использовании такой техники каждая процедура системы имеет хорошо определенный интерфейс в терминах параметров и результатов и каждая может вызвать любую другую для выполнения некоторой работы.

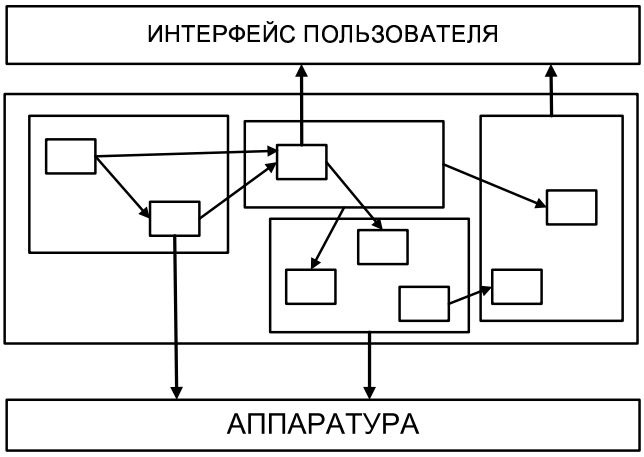


*Рис. 1.1*

Все работы по разработке первых монолитных систем ложились на плечи одного программиста (или группы программистов), а значит, приходилось каждый раз разрабатывать все от начала до конца. Причин этому несколько.

Во-первых, средства, предоставляемые операционной системой, были достаточно скудны и не обеспечивали прикладному программисту необходимый уровень сервисов.

Во-вторых, инструментальные средства для разработки программного обеспечения находились на примитивном уровне и не позволяли выполнять разработку и отладку отдельных компонентов программ.

В-третьих, все программы разрабатывались в рамках одной организации (или одного подразделения) и предназначались для решения одной очень узкой задачи. Возможность повторного использования разработанного программного обеспечения практически отсутствовала. Отчасти это было обусловлено плохой совместимостью аппаратуры и отсутствием стандартизации.

В описанных условиях разработка сложных программных систем была невозможной. Более того, принципиально невозможным являлось сопровождение и развитие уже созданных систем.

## 1.2. Архитектуры "клиент–сервер"

Пожалуй, первой хорошо зарекомендовавшей себя программной архитектурой для построения информационных систем стала архитектура "клиент–сервер". Главное ее отличие от монолитной архитектуры заключается в разделении системы на два четко определенных стандартных уровня:

- уровень организации и хранения данных;
- уровень бизнес-логики.

С появлением архитектуры "клиент–сервер" стали появляться распределенные (в современном понимании) системы обработки информации. Хранилища данных и код доступа к ним стало удобно выделять в виде отдельных серверов. Были также заимствованы многие достоинства мэйнфреймовой модели вычислений.

### *1.2.1. Двузвенная архитектура*

Появление двузвенной (классической) архитектуры "клиент–сервер" с выделением серверов данных и клиентстких приложений (рис. 1.2) стало возможным прежде всего с появлением стандартного интерфейса организации доступа к структурированным данным – SQL, который появился в середине 70-х гг. XX в. и был разработан в рамках проекта экспериментальной реляционной СУБД System R [2].

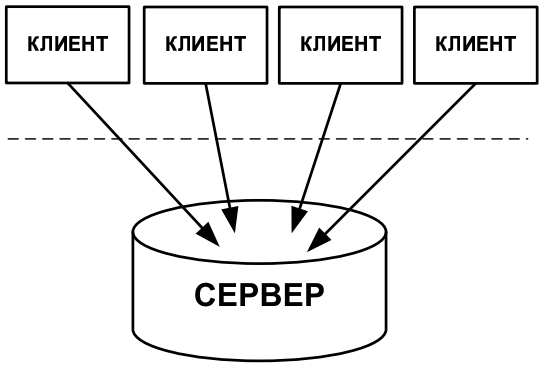


*Рис. 1.2*

Исходное название языка – SEQUEL (Structured English Query Language) – только частично отражает его суть. Язык был ориентирован главным образом на удобную и понятную пользователям формулировку запросов к реляционной БД и уже являлся полным языком БД, содержащим помимо операторов формулирования запросов и манипулирования БД средства определения и манипулирования схемой БД; средства определения ограничений целостности; возможности определения структур физического уровня, поддерживающих эффективное выполнение запросов; средства авторизации доступа к отношениям и их полям; возможности описания точек сохранения транзакций и откатов.

Сервер баз данных – основной компонент системы "клиент–сервер", осуществляющий целый комплекс действий по управлению данными, обязанностями которого являются:

- выполнение пользовательских запросов на выбор и на модификацию данных и метаданных, получаемых от клиентских приложений, функционирующих на ПЭВМ локальной сети;
- хранение и резервное копирование данных;
- поддержка ссылочной целостности данных согласно определенным в базе данных правилам;
- обеспечение авторизованного доступа к данным на основе проверки прав и привилегий пользователей;
- протоколирование операций и ведение журнала транзакций.

Одно из важнейших преимуществ архитектуры "клиент–сервер" – общее снижение затрат на взаимодействия компонетов системы. Другим преимуществом данной архитектуры является возможность хранения бизнес-правил (например, правил ссылочной целостности или ограничений на значения данных) на сервере, что позволяет избежать дублирования кода в различных клиентских приложениях, использующих общую базу данных. Кроме того, в данном случае любое редактирование данных, в том числе и редактирование средствами, не предусмотренными разработчиками информационной системы (например, различными утилитами администрирования сервера), может быть произведено только с учетом этих правил.

### *1.2.2. Многозвенные архитектуры*

Описанная двузвенная архитектура имеет ряд существенных недостатков:

- высокие требования к серверу при большом количестве клиентов и данных;
- сложность администрирования и сопровождения;
- невозможность организации эффективной работы удаленных пользователей.

Одной из первых практических задач, обусловившей дальнейшее развитие двузвенных архитектур, стала задача повышения производительности и надежности серверов данных. Первые попытки ее решения привели к возникновению следующих технологий организации данных [3]: распределенных БД; тиражирования; согласования транзакций.

Тенденция разделения информационных систем по специализированным уровням наблюдается и в настоящее время. Таким образом, сложность проектирования и разработки данных систем перемещается от чисто функциональных задач к задачам управления и взаимодействия.

**Технология распределенной БД.** Распределенная БД включает фрагменты данных, расположенных на различных узлах сети. С точки зрения пользователей, она выглядит так, как будто все данные хранятся в одном месте. Естественно, подобная схема предъявляет жесткие требования к производительности и надежности каналов связи.

**Технология тиражирования.** В этом случае в каждом узле сети дублируются данные всех компьютеров. При этом выполняются следующие требования:

- передаются только операции изменения данных, а не сами данные;
- передача может быть асинхронной (неодновременной для разных узлов);
- данные располагаются там, где обрабатываются.

Такие решения позволяет снизить требования к пиковой пропускной способности каналов связи, более того, при выходе из строя линии связи какого-либо компьютера пользователи других узлов могут продолжать работу. Однако при этом допускается неодинаковое состояние базы данных для различных пользователей в один и тот же момент времени, а значит, невозможно исключить конфликты между двумя копиями одной и той же записи.

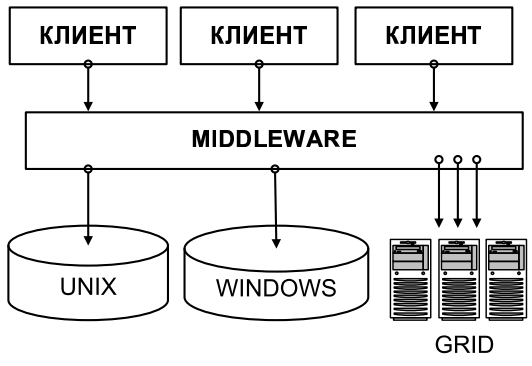


*Рис. 1.3*

Дальнейшее развитие методов распределенной обработки данных привело к расслоению кода доступа к данным и управления ими и, как следствие, к появлению *многозвенных* (или, другими словами, многослойных) архитекур "клиент–сервер" (рис. 1.3). Такие архитектуры более разумно распределяют модули обработки данных, которые в данном случае выполняются на одном сервере или на нескольких отдельных серверах. Эти программные модули выполняют функции сервера для интерфейсов с пользователями и клиента для серверов баз данных. Кроме того, различные серверы приложений могут взаимодействовать между собой для более точного разделения системы на функциональные блоки, выполняющие

определенные роли. В частности, можно выделить сервер управления персоналом, который будет выполнять все необходимые для управления персоналом функции. Связав с ним отдельную базу данных, можно скрыть от пользователей все детали реализации этого сервера, разрешив им обращаться только к его общедоступным функциям.

Например, в трехуровневой архитектуре тонкий клиент не перегружен функциями обработки данных, а выполняет свою основную роль системы представления информации, поступающей с сервера приложений. Такой интерфейс можно реализовать с помощью стандартных web-технологий – браузера, CGI и Java. Это уменьшит объем данных, передаваемых между клиентом и сервером приложений, что позволит подключать клиентские компьютеры даже по медленным линиям (типа телефонных каналов). Клиентская часть может быть настолько простой, что в большинстве случаев ее реализуют с помощью универсального браузера.

Трехуровневая архитектура "клиент–сервер" позволяет более точно назначать полномочия пользователей, так как они получают права доступа не к самой базе данных, а к определенным функциям сервера приложений. Это повышает защищенность системы (по сравнению с обычной двухуровневой архитектурой) не только от умышленного вторжения, но и от ошибочных действий персонала.

**Согласование транзакций.** Следующим шагом оптимизации архитектур "клиент–сервер" стало использование специальных серверов для согласования совместной работы клиентов с интенсивно используемыми общими ресурсами.

*Менеджер транзакций* (МТ) – это программа или комплекс программ, с помощью которых можно согласовать работу различных компонентов информационной системы. Логически МТ делится на несколько частей:

- коммуникационный менеджер, контролирующий обмен сообщениями между компонентами информационной системы;
- менеджер авторизации, обеспечивающий аутентификацию пользователей и проверку их прав доступа;
- менеджер транзакций, управляющий распределенными операциями;
- менеджер ведения журнальных записей, следящий за восстановлением и откатом распределенных операций;
- менеджер блокировок, обеспечивающий правильный доступ к совместно используемым данным.

Коммуникационный менеджер обычно объединен с авторизационным, а менеджер транзакций работает совместно с менеджерами блокировок и системных записей.

## 1.3. Системы, ориентированные на сервисы

Дальнейшим развитием распределенных информационно-вычислительных систем после многоуровневых архитектур "клиент–сервер" стали распределенные системы, ориентированные на сервисы. Наиболее распространенными среди сервисных архитектур являются:

- web-ориентированные;
- композитные сервис-ориентированные;
- событийно упрявляемые.

Следует отметить, что web-ориентированные архитектуры постепенно замещаются сервис-ориентированными, а событийно-управляемые только зарождаются.

### *1.3.1. Web-Сервисы*

Web-Сервис – это приложение, доступное через Интернет и предоставляющее определенные услуги, форма которых не зависит от поставщика (так как используются универсальные форматы данных и платформы функционирования). Существует несколько различных технологий, поддерживающих концепцию распределенных объектных web-систем. К ним относятся технологии EJB, CORBA и DCOM.

В этом случае между серверами и клиентами нет различий и систему можно представить как набор взаимодействующих объектов, местоположение которых не имеет особого значения. Главным отличием от многоуровневых клиент–серверных архитектур здесь является то, между поставщиками сервисов и их потребителями не существует различий.

### *1.3.2. Сервис-ориентированная архитектура*

Сервис-ориентированная архитектура – парадигма, основанная на превращении функций и компонентов информационной системы в "услуги", к которым можно обратиться через стандартный интерфейс независимо от местоположения или технической составляющей функции или части данных.

СОА отвечает следующим характеристикам:

- Позволяет потребителям услуг и поставщикам взаимодействовать независимо от технической составляющей или от местоположения; позволяет потребителям идентифицировать и обнаруживать услуги, которыми

они интересуются, с минимальным уровнем взаимодействия (являющимся необходимым) между потребителем и поставщиком этих услуг.

- Поддерживает взаимодействия, в которых определенная система запрашивает часть информации или выполнение функции от определенного поставщика услуги и этот поставщик впоследствии посылает ответ, обеспечивающий работу системы.
- Поддерживает синхронное обращение и выполнение услуг. Это означает, что когда потребитель запрашивает часть информации или вызывает функцию, связь между этими двумя системами должна осуществляться, пока ответ не будет получен.

### *1.3.3. Событийно-управляемая архитектура*

Событийно-управляемая архитектура – парадигма, основанная на использовании событий как пусковых механизмов, которые инициируют немедленную доставку сообщения, информирующего многочисленных получателей о событии, для предпринятия ими соответствующих действий. Когда этот механизм используется для управления, совокупность событий может быть проанализирована для идентификации релевантности и затем собрана для создания информации, которая необходима для действенного решения задачи.

Событийно-управляемая архитектура обладает следующими чертами:

- Позволяет осуществлять взаимодействия между системами, в которых "издатель" сообщения не знает, кем являются подписчики, и, наоборот, – взаимодействия полностью основаны на информации, отправленной и полученной, а не на взаимоотношениях между участниками обмена.
- Поддерживает взаимодействия "многие к многим", в которых системы "издают" информацию о некотором событии таким образом, чтобы другие многочисленные системы (которые подписаны и уполномочены для получения таких сообщений) могли получить информацию. После получения сообщения система-получатель принимает решение о необходимости дальнейшей работы и выполняет полезные действия либо порождает новые события.
- Поддерживает асинхронные взаимодействия, в которых информация посылается без ожидания немедленного ответа или требования осуществления постоянной связи между двумя системами в момент ожидания ответа.

Существуют противоречивые мнения по поводу развития сервис-ориентированных и событийно-управляемых архитектур. Одни аналитики считают, что событийно-управляемые архитектуры придут на смену сервис-ориенти-

рованным, другие полагают, что предстоит их интеграция, которая позволит использовать преимущества каждого из подходов.

## 1.4. Контрольные вопросы

1. Какие архитектуры программных систем вы знаете?
2. В чем преимущество двузвенной архитектуры "клиент–сервер" от монолитной архитектуры?
3. Почему двузвенной архитектуры недостаточно для построения распределенных приложений?
4. Чем обусловлено появление нескольких уровней в архитектурах "клиент–сервер"?
5. Что такое согласование транзакций?
6. Какие сервисные архитектуры вам известны?
7. Что такое web-сервис?
8. Дайте определение сервис-ориентированной архитектуры.
9. Что такое событийно-управляемая архитектура?

# 2. РАСПРЕДЕЛЕННЫЕ СИСТЕМЫ. ПРИНЦИПЫ ОРГАНИЗАЦИИ

Понятие распределенной системы достаточно широко используется в области информационных технологий и имеет несколько смыслов в различном контексте. По ходу изложения материала, будет пользовано следующее определение распределенной системы:

*распределенная система* — это набор независимых аппаратных и программных средств, предназначенных для координированного выполнения определенного круга задач, представляющихся пользователю единой системой.

## 2.1. Свойства распределенных систем

Основная задача, которую пытаются решить с помощью распределенных систем, – обеспечение максимально простого, но в то же время эффективного доступа как можно большему числу пользователей к всем ресурсам, которые могут им понадобиться. При этом важными свойствами распределенных систем являются следующие:

- прозрачность;
- открытость;
- масштабируемость;
- безопасность.

Рассмотрим каждое из перечисленных свойств отдельно [4].

### *2.1.1. Прозрачность*

Прозрачность проявляется как способность системы скрыть от пользователя физическое распределение ресурсов, трудности, возникающие при одновременной работе нескольких пользователей с одним ресурсом, ошибки при доступе к ресурсам и в работе самих ресурсов.

Степень прозрачности может быть различной, поскольку скрывать от пользователя все эффекты, возникающие при работе распределенной системы, неразумно. Кроме того, прозрачность системы и ее производительность обычно находятся в обратной зависимости – например, при попытке скрыть отказы в соединении с сервером большинство Интернет-клиентов пытается установить такое соединение несколько раз, а для пользователя это выглядит, как значительно замедленная реакция системы.

### *2.1.2. Открытость*

Открытость системы определяется как полнота и четкость описания интерфейсов работы с ней и служб, которые она предоставляет через эти интерфейсы. Такое описание должно включать в себя все, что необходимо знать для того, чтобы пользоваться этими службами независимо от реализации данной системы и базовой платформы.

Открытость системы важна как для обеспечения ее переносимости, так и для облегчения использования системы и возможности построения других систем на ее основе.

### *2.1.3. Масштабируемость*

Масштабируемость – свойство системы развиваться в направлении увелечения количественных и качественных характеристик своих основных свойств и функций (например, это может быть зависимость производительности от числа пользователей или числа подключенных ресурсов, а также возможность увеличения географической распределенности системы).

В число основных характеристик при этом попадают производительность; стоимость; трудозатраты на разработку, на внесение изменений, на сопровождение, на администрирование; удобство работы с системой. Для некоторых из них почти наилучшая возможная масштабируемость обеспечивается линейной зависимостью, для других хорошая масштабируемость означает, что при изменении масштабов системы показатель не меняется вообще или изменяется незначительно. Воспрос масштабирования современных программных систем является одним из ключевых.

Существует два подхода к масштабированию – вертикальное масштабирование и горизонтальное. Вертикальное масштабирование подразумевает увеличение прозводительности за счет использования больших симметричных систем обработки (как правило, SMP-системы). При альтернативном, горизонтальном масштабировании компонетны распределенной системы соединяются через сеть или объединяются в кластер (при этом используются дешевые комплектующие).

При реализации больших систем хорошая масштабируемость может быть достигнута с помощью следующих методов:

- *децентрализацией служб*, состоящей в использовании нескольких узлов для обработки поступающих запросов;
- *децентрализацией данных*, состоящей в использовании нескольких хранилищ данных или нескольких копий одного хранилища;
- *децентрализацией алгоритмов*, состоящей в использовании для обработки и перенаправления запросов алгоритмов, не требующих полной информации о состоянии системы, не разрушающихся при сбое одного из ресурсов системы и не предполагающих единого хода времени на всех узлах, входящих в систему;
- *использованием асинхронной связи* в виде передачи сообщений без приостановки работы до прихода ответа;
- *использованием комбинированных систем* организации взаимодействия, основанных на иерархической организации систем (хорошо масштабирующей задачи поиска), на репликации (построении копий данных и их распределении по системе для балансировки нагрузки на разные ее элементы) и ее частном случае – кэшировании (организующем хранение результатов наиболее часто используемых запросов как можно ближе к клиенту), на взаимодействии "точка-точка" (обеспечивающем независимость от других компьютеров в системе).

Для масштабируемой производительности обычно требуется, чтобы параметры задач, решаемых системой за одно и то же время, можно было увеличивать достаточно быстро (лучше – линейно или еще быстрее, но это возможно не для всех задач) при возрастании количества имеющихся ресурсов, в частности отдельных компьютерах. Однако очень плохо, если внесение изменений в систему становится все более трудоемким при ее росте – желательно, чтобы трудоемкость внесения одного изменения почти не возрастала. На практике достичь этого требования чрезвычайно трудно.

Иногда в масштабируемость включают административную масштабируемость системы – зависимость удобства работы с ней от числа административно независимых организаций, связанных с этой системой.

### *2.1.4. Безопасность*

Поскольку распределенные системы вовлекают в свою работу множество пользователей, компьютеров и географически разделенных элементов, безопасность работы таких систем имеет бóльшее значение, чем при работе обычных систем, сосредоточенных на одном физическом узле. Понятие безопасности включает целостность данных, защиту, отказоустойчивость.

**Сохранность и целостность данных.** При обеспечении групповой работы многих пользователей с одними и теми же данными нужно обеспечивать сохранность последних, т. е. предотвращать исчезновение данных, введенных одним из пользователей, и в то же время целостность данных, т. е. выполнение всех присущих данным ограничений (их непротиворечивость).

Это непростая задача, не имеющая полного решения, удовлетворяющего все заинтересованные в нем стороны во всех ситуациях. При одновременном изменении одного и того же элемента данных разными пользователями итоговый результат должен быть непротиворечив, и поэтому часто может совпадать только с одним из введенных вариантов. Как будет обработана подобная ситуация и возможно ли ее возникновение вообще, зависит от дополнительных требований к системе, от принятых протоколов работы, от того, какие риски (потерять данные одного из пользователей или значительно усложнить методы работы пользователей с системой) будут сочтены более важными.

**Защищенность данных и коммуникаций.** При работе с коммерческими системами, с системами, содержащими большие объемы персональной и бизнес-информации, с системами обслуживания пользователей государственных служб достаточно важна защищенность как информации, постоянно хранящейся в системе, так и информации одного сеанса работы. Для распределенных систем обеспечить защищенность гораздо сложнее, поскольку нельзя физически изолировать все элементы системы и разрешить доступ к ней только особо проверенным пользователям.

**Отказоустойчивость и способность к восстановлению после ошибок.** Одним из достоинств распределенных систем является возможность построения более надежно работающей системы из не вполне надежных компонентов. Однако для того чтобы это достоинство стало реальным, необходимо тщательное проектирование систем во избежание зависимости работоспособности си-

стемы от ее отдельных компонентов, поскольку оно превратится в недостаток, так как в распределенной системе элементов больше, а значит выше вероятность того, что хотя бы один элемент выйдет из строя и хотя бы один ресурс окажется недоступным.

Еще важнее для распределенных систем – уметь восстанавливаться после сбоев. Уровни этого восстановления могут быть различными. Так, обычно, данные одного сеанса работы считается возможным не восстанавливать, поскольку такие данные часто малозначимы или легко восстанавливаются, но так называемые постоянно хранимые (persistent) данные обычно требуется восстанавливать до последнего непротиворечивого их состояния.

### *2.1.5. Аспекты разработки распределенных систем*

Далее перечислены основные аспекты, которые рассматривают при разработке и анализе распределенных систем.

**Связь и передача данных между компонентами системы.** В данном аспекте определяются протоколы организации связи; способы реализации обращения к процедурам и методам объектов некоторого потока из других; организация синхронной и асинхронной передач сообщений; организация сохранности (асинхронных) сообщений в то время, когда и отправитель и получатель сообщения могут быть неактивны; организация передачи непрерывных потоков данных (аудио- и видеоданные, смешанные потоки).

**Работа процессов и потоков.** В рамках данного аспекта рассматривается разделение работ в системе на отдельные процессы; определение ролей процессов (пример – клиентские и серверные); вопросы организации исполняемых агентов, способных мигрировать между отдельными компьютерами.

Вопросы организации работы процессов и потоков – особая тема; она более подробно раскрывается в дальнейшем изложении.

**Именование и поиск ресурсов.** К этому аспекту относятся вопросы, связанные с присвоением имен и идентификаторов различным ресурсам, с поиском ресурсов в системе по именам и атрибутам, с размещением и поиском мобильных ресурсов, изменяющих в ходе работы свое физическое положение.

Сюда же относят и организацию и поддержку сложных ссылочных структур вообще – например, вопросы удаления ставших никому не доступными ресурсов.

В некоторых распределенных системах используются технологии управления временем жизни объектов, в том числе специализированные реестры ссылок и сборщики мусора.

**Синхронизация.** В рамках этого аспекта рассматриваются вопросы, связанные с организацией взаимодействия между параллельно работающими для получения общего результата потоками и процессами; алгоритмы синхронизации времени и организации работ в том случае, если в системе нельзя непротиворечиво определить глобальное время; вопросы организации транзакций.

**Поддержка целостности и непротиворечивости данных.** Данный аспект касается способов организации целостности данных и поддерживаемых при этом моделей непротиворечивости (определяющих, на основе каких требований формируются результаты выполняемых одновременно изменений, и что доступно клиентам, выполнявшим эти изменения). В связи с этим определяются протоколы обеспечения непротиворечивости, создания и согласования реплик и кэшей, обеспечивающие выполнение указанных моделей.

**Защита данных и коммуникаций.** Сюда относятся вопросы обеспечения защищенности системы в целом (при этом большее значение, чем технические аспекты, имеют проблемы определения процедур проведения работ, обеспечивающих нужный уровень защищенности, и проблемы соблюдения людьми таких процедур); организации защиты данных от несанкционированного доступа; обеспечения защиты каналов связи с двух сторон – препятствование в несанкционированном доступе к передаваемой информации и препятствование в подмене информации в каналах связи; вопросы аутентификации пользователей, протоколы подтверждения идентичности и авторства.

## 2.2. Асинхронная обработка информации

Для понимания того, как функционируют распределенные системы, необходимо знать принципы асинхронной обработки данных. Важными являются такие ее аспекты, как разделение времени, синхронизация, модели диспетчеризации и связи между асинхронными потоками и процессами. Далее рассматриваются основные принципы и технологии асинхронной обработки данных.

### *2.2.1. Процессы*

Все, что выполняется в вычислительных системах, организовано как набор процессов. Понятие процесса характеризуется совокупностью набора исполняющихся команд, ассоциированных с ним ресурсов (выделенные для исполнения память или адресное пространство, стеки, используемые файлы и устройства ввода–вывода и т. д.) и текущего момента его выполнения (значе-

ния регистров, программного счетчика, состояние стека и значения переменных), находящейся под управлением операционной системы. Не существует взаимно однозначного соответствия между процессами и программами, обрабатываемыми вычислительными системами.

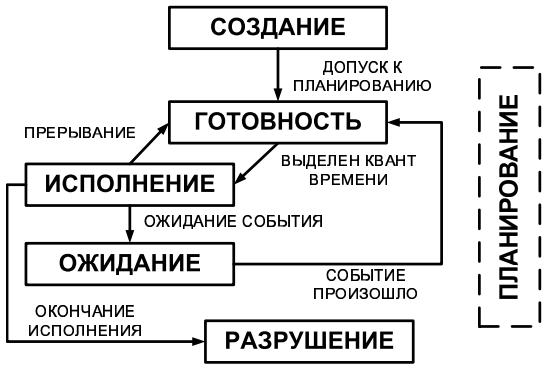


*Рис. 2.1*

Обычно терминология процессов и потоков сильно связана с понятием операционной системы (ОС). Именно в данной области появились и были развиты основные концепции распределенных систем. Понятия распределенных систем, которым посвящено данное пособие, существенно шире, однако общие идеи организации асинхронной обработки схожи. На рис. 2.1 представлена диаграмма состояний процесса. При состоянии рождения процесс получает адресное пространство, в которое загружается программный код; выделяются стек и системные ресурсы; устанавливается начальное значение программного счетчика и т. д., и он переводится в состояние готовности.

Операционная система или сервер диспетчеризации (планировщик), пользуясь каким-либо алгоритмом планирования, выбирают один из готовых процессов и переводят его в состояние "исполнение", в котором происходит непосредственное выполнение программного кода. Остальные процессы, существующие в системе, для которых не хватает исполнительных ресурсов переводятся в состояние "ожидание". Выйти из состояния исполнения процесс может по трем причинам:

- система прекращает его выполнение;
- он не может продолжать выполнение, пока не произойдет некоторое событие, и переводится в состояние ожидания;
- в результате возникновения прерывания в вычислительной системе он возвращается в состояние "готовность".

Из состояния "ожидание" процесс попадает в состояние "готовность": если ожидаемое событие произошло, он снова может быть выбран для исполнения. По своем завершении процесс из состояния "исполнение" попадает в состояние завершения исполнения.

Изменением состояния процессов занимаются операционная система или сервер диспетчеризации, совершая операции над ним. Операции объединяются в три пары:

- создание процесса ⇔ завершение процесса;
- приостановка процесса ⇔ запуск процесса;
- блокирование процесса ⇔ разблокирование процесса.

В некоторых моделях планирования задач есть дополнительная операция изменения приоритета процесса.

Каждый процесс представляется некоторой структурой данных, содержащей специфическую информацию о нем:

- состояние, в котором находится процесс;
- программный счетчик процесса, т. е. адрес команды, которая должна быть выполнена следующей;
- содержимое регистров процессора или исполняемого контектста;
- данные, необходимые для планирования использования процессора и управления памятью (приоритет процесса, размер и расположение адресного пространства и т. д.);
- учетные данные;
- сведения об устройствах ввода–вывода, связанных с процессом.

Состав и строение структуры описания зависят от конкретной программной реализации. Информация может храниться не в одной, а в нескольких связанных структурах данных. Такие структуры могут иметь различные наименования, содержать дополнительную информацию или, наоборот, лишь часть описанной информации.

Принято считать, что состояние процесса определяется его описателем – блоком управления процессом (Process Control Block, PCB).

Блок управления процессом является моделью процесса. Любая операция, производимая операционной системой над процессом, вызывает в нем определенные изменения. В рамках принятой модели состояний процессов содержимое PCB между операциями остается постоянным. Информацию, для хранения которой предназначен блок управления процессом, разделяют на две части. Содержимое всех регистров процессора (включая значение программного счетчика) называется *регистровым контекстом* процесса, а все остальное – *системным контекстом*.

С точки зрения пользователя, наибольший интерес представляет содержимое адресного пространства процесса наряду с регистровым контекстом

определяющее последовательность преобразования данных и полученные результаты. Код и данные, находящиеся в адресном пространстве процесса, называются его *пользовательским контекстом*. Совокупность регистрового, системного и пользовательского контекстов принято называть просто контекстом процесса. В любой момент времени процесс полностью характеризуется своим контекстом.

### *2.2.2. Потоки*

Модель процесса базируется на двух независимых концепциях – группировании ресурсов и выполнении программы. Понятие потока появляется, когда необходимо их разделить.

Создание потоков требует меньших накладных расходов. Все потоки одного процесса используют общие файлы, таймеры, устройства, область оперативной памяти, адресное пространство и разделяют одни и те же глобальные переменные. Каждый поток может иметь доступ к любому виртуальному адресу процесса, один поток может использовать стек другого потока. Между потоками одного процесса нет полной защиты. Для организации взаимодействия и обмена данными не требуется обращаться к глобальному планировщику, достаточно использовать общую память: один поток записывает данные, а другой – читает их. При этом потоки разных процессов по-прежнему хорошо защищены друг от друга.

Мультипрограммирование более эффективно на уровне потоков, а не процессов. Каждый поток имеет собственный счетчик команд и собственный стек.

Модель процесса переносима на модель потока. В многопоточной системе при создании процесса создается как минимум один поток выполнения. При создании потока так же, как и при создании процесса, генерируется, специальная информационная структура – описатель потока, содержащий идентификатор потока, данные о правах доступа, о приоритете, о состоянии потока и другую информацию.

Планирование потоков осуществляется на основе информации, хранящейся в описателях процессов и потоков. При планировании могут приниматься во внимание приоритет потоков, время их ожидания в очереди, накопленное время выполнения, интенсивность обращений к вводу–выводу и другие факторы. Планирование выполнения потоков выполняется независимо от того, принадлежат ли они одному процессу или разным, однако трудоемкость сохранения и восстановления контекста зависит от условий функционирования.

Существуют следующие причины необходимости применения потоков:

- существенное упрощение схемы программы при разбиении приложения на несколько последовательных потоков, запущенных в квазипараллельном режиме;
- возможность совместного использования параллельными объектами адресного пространства и всех содержащихся в нем данных;
- легкость их создания и уничтожения (так как с потоком не связаны никакие ресурсы);
- увеличение производительности при совмещении во времени различных видов деятельности.

Потоки могут быть реализованы на уровне пользователя или на уровне ядра; существуют и смешанные модели. Они используются в распределенных системах.

Примером этого может служить обработка входящих сообщений, например запросов на обслуживание. Традиционный подход к решению этой задачи заключается в наличии процесса или потока, которые блокируются по системному запросу "recieve", ожидая входящего сообщения. Когда сообщение доставляется получателю, оно принимается и обрабатывается.

Другой подход характеризуется тем, что по прибытии сообщения система создает новый поток для его обработки – всплывающий поток. Основным преимуществом всплывающих потоков является отсутствие истории: регистров, стека и прочей информации, которую нужно восстанавливать. Новый поток обрабатывает входящее сообщение – это позволяет значительно сократить промежуток времени между прибытием сообщения и началом его обработки. При использовании всплывающих потоков необходимо предварительное планирование.

Достоинства использования потоков для решения такого рода задач – в том, что создание всплывающих потоков в пространстве ядра быстрее и проще, чем в пространстве пользователя; всплывающему потоку проще получить доступ к всем таблицам ядра и устройств ввода–вывода, что может оказаться полезным при обработке прерываний. Некдостаток – наличие ошибок в потоке, расположенном в пространстве ядра, что может нанести значительный ущерб устойчивости системы.

### *2.2.3. Методы синхронизации*

*Синхронизация* (от *греч.* synchronos – одновременный) – приведение двух или нескольких процессов к такому их протеканию, когда одинаковые или со-

ответствующие элементы процессов совершаются с неизменным сдвигом во времени либо одновременно. Другими словами, – это механизм согласования событий в единой временно́й шкале для процессов или потоков, выполняющихся асинхронно.

Синхронизация необходима процессам для организации совместного использования ресурсов, таких, как файлы или устройства, а также для обмена данными.

В однопроцессорных системах решение задач взаимного исключения, критических областей и других проблем синхронизации осуществлялось с использованием специализированных объектов, таких, как *семафоры* и *мониторы*. Однако эти методы не совсем подходят для распределенных систем, поскольку все они базируются на использовании разделяемой оперативной памяти (например, два процесса, которые взаимодействуют, используя семафор, должны иметь доступ к нему). Если оба процесса выполняются на одном и том же компьютере, они могут иметь совместный доступ к семафору, хранящемуся, например в ядре, делая системные вызовы. Однако если процессы выполняются на разных компьютерах, то этот метод не применим – для распределенных систем нужны новые подходы. Далее рассматриваются основные подходы к синхронизации в распределенных системах [4].

### *2.2.4. Синхронизация логических часов*

В централизованной однопроцессорной системе важно только относительное время и не важна точность часов. В распределенной системе, где каждый процессор имеет собственные часы со своей точностью хода, ситуация резко меняется: программы, использующие время, становятся зависимыми от того, часами какого компьютера они пользуются.

В распределенных системах синхронизация физических часов является сложной проблемой, но, с другой стороны, очень часто в этом нет никакой необходимости: т. е. процессам не нужно, чтобы во всех компьютерах было правильное время, для них важно, чтобы оно было везде одинаковое, более того, для некоторых процессов важен только правильный порядок событий. В этом случае пользователь имеет дело с *логическими часами*.

Для двух произвольных событий вводится отношение "случилось до". Выражение $a < b$ читается "$a$ случилось до $b$" и означает, что все процессы в системе считают, что сначала произошло событие $a$, а потом – событие $b$. Отношение "случилось до" обладает свойством транзитивности: если выражения $a < b$ и $b < c$ истинны, то справедливо и выражение $a < c$.

Для двух событий одного и того же процесса всегда можно установить отношение "случилось до", аналогично может быть установлено это отношение и для событий передачи сообщения одним процессом и приемом его другим, так как прием не может произойти раньше отправки. Ставится задача создания такого механизма ведения времени, который бы для каждого события мог указать значение времени $T(a)$, с которым бы были согласны все процессы в системе. При этом должно выполняться условие: если $a < b$ , то $T(a) < T(b)$. Кроме того, любые корректировки времени могут выполняться только добавлением положительных значений.

Алгоритм решения этой задачи предложил Лампорт (Lamport) [5]. Для отметок времени в нем используются события. На рис. 2.2 показаны три процесса, выполняющихся на разных компьютерах, каждый из которых имеет свои часы, идущие со своей скоростью. Как видно из рис. 2.2, *а*, когда часы процесса 1 показали время 6, в процессе 2 часы показывали 8, а в процессе 3 – 10. Предполагается, что все эти часы идут с постоянной для себя скоростью.

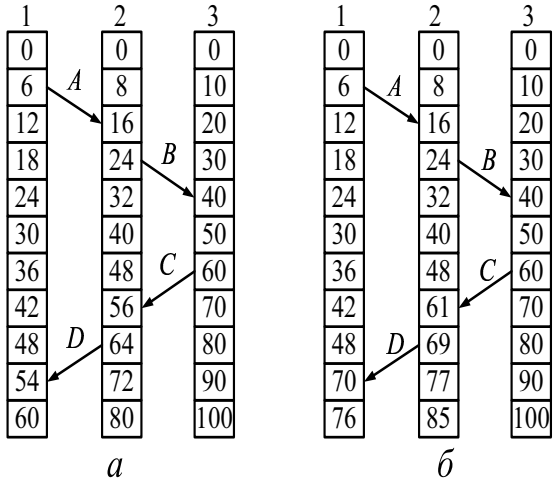


*Рис. 2.2*

В момент времени 6 процесс 1 посылает сообщение $A$ процессу 2. Это сообщение приходит к процессу 2 в момент времени 16 по его часам. В логическом смысле это вполне возможно, так как $6 < 16$. Аналогично, сообщение $B$, посланное процессом 2 процессу 3, пришло к последнему в момент времени 40, т. е. его передача заняла 16 единиц времени, что также является правдоподобным.

Сообщение $C$ от процесса 3 к процессу 2 было отправлено в момент времени 64, а поступило в место назначения в момент времени 54. Очевидно, что это невозможно. Решение вытекает непосредственно из отношений "случилось до". Так как сообщение $C$ было отправлено в момент 60, то оно должно дойти до места назначения в момент 61 или позже. Следовательно, каждое сообщение должно нести с собой время своего отправления по часам узла-отправителя. Если в компьютере-получателе часы показывают время, которое меньше времени отправления, то эти часы переводятся вперед так, чтобы они показали время, большее времени отправления сообщения.

### *2.2.5. Транзакции*

Компьютерная транзакция полностью аналогична многосторонней банковской операции, которая завершается только после того, как все стороны, ведущие переговоры, договариваются о конкретном платеже.

Распространив это положение на процессы или потоки, имеем: один процесс объявляет, что он хочет начать транзакцию с одним процессом или более. Эти процессы могут некоторое время создавать и уничтожать разные объекты, выполнять какие-либо операции. Затем процесс-инициатор объявляет, что он хочет завершить транзакцию. Если все остальные процессы с ним соглашаются, то результат фиксируется.

Если один процесс или более отказываются от завершения транзакции (или они прекратили свою работу еще до выработки согласия), тогда измененные объекты возвращаются точно к тому состоянию, в котором они находились до начала выполнения транзакции.

Для программирования с использованием транзакций требуется некоторый набор *примитивов*, которые должны быть предоставлены программисту либо операционной системой, либо языком программирования.

Примеры примитивов такого рода:

- BEGIN_TRANSACTION – команды, которые следуют за этим примитивом, формируют транзакцию;
- END_TRANSACTION – завершает транзакцию и пытается зафиксировать ее;
- ABORT_TRANSACTION – прерывает транзакцию, восстанавливает предыдущие значения;
- READ – читает данные из файла (или из другого объекта);
- WRITE – пишет данные в файл (или в другой объект).

Первые два примитива используются для определения границ транзакции. Операции между ними представляют собой тело транзакции, причем либо все они должны быть выполнены, либо ни одна из них. Это могут быть системный вызов, библиотечная процедура или группа операторов языка программирования.

Транзакции обладают следующими свойствами: упорядочиваемостью, неделимостью, постоянством и непротиворечивостью.

Упорядочиваемость гарантирует, что если две или более транзакции выполняются в одно и то же время, то конечный результат выглядит так, как если бы все транзакции выполнялись последовательно в некотором (в зависимости

от системы) порядке. Неделимость означает, что когда транзакция находится в процессе выполнения, то никакой другой процесс не видит ее промежуточные результаты. Постоянство означает, что после фиксации транзакции внесенные ее изменения становятся постоянными.

Если программное обеспечение гарантирует перечисленные свойства, то это означает, что в системе поддерживается механизм транзакций. Выделяют транзакции следующих типов: плоские, вложенные, распределенные.

*Плоские* транзакции – серия операций, удовлетворяющая свойствам транзакций. Плоские транзакции – это наиболее простой и наиболее часто используемый тип транзакций. Однако они имеют множество ограничений, основным из которых является то, что они не могут давать частичного результата в случае своего завершения или прерывания.

*Вложенные* транзакции подразумевают, что транзакция верхнего уровня может разделяться на дочерние транзакции, работающие параллельно на различных машинах для повышения производительности или упрощения программирования. Каждая из этих дочерних транзакций может состоять из транзакции одной или более, каждая из которых, в свою очередь, может делиться на дочерние транзакции.

*Распределенные* транзакции логически представляют собой плоские неделимые транзакции, которые работают с распределенными данными. Основная проблема распределенных транзакций в том, что для блокировки данных и подтверждения транзакции необходимы отдельные распределенные алгоритмы.

### *2.2.6. Механизмы синхронизации*

В распределенных системах используется целый спектр объектов синхронизации. Далее рассматриваются самые наиболее используемые.

**Семафоры.** Одним из первых механизмов, предложенных для синхронизации поведения процессов, стали семафоры, концепцию которых описал Дейкстра в 1965 г. [6].

В современных распределенных системах семафоры используются очень активно, особенно тогда, когда требуется максимальное быстродействие при минимальных накладных расходах. Семафор представляет собой целую переменную, принимающую неотрицательные значения, доступ любого процесса к которой, за исключением момента ее инициализации, может осуществляться только через две атомарные операции: $P$ (от *дат.* proberen – проверять) и $V$ (от *дат.* verhogen – увеличивать).

Классическое определение этих операций выглядит следующим образом:

$P(S)$:
  пока $S == 0$
    процесс блокируется;
  $S = S - 1$;
$V(S)$:
  $S = S + 1$;

Эта запись означает следующее: при выполнении операции $P$ над семафором $S$ сначала проверяется его значение. Если оно больше $0$, то из $S$ вычитается $1$. Если оно меньше или равно $0$, то процесс блокируется до тех пор, пока $S$ не станет больше $0$, после чего из $S$ вычитается $1$. При выполнении операции $V$ над семафором $S$ к его значению просто прибавляется $1$.

Переменные-семафоры могут быть с успехом применены для решения различных задач организации взаимодействия процессов. В ряде языков программирования они были введены непосредственно в синтаксис языка, в других случаях применяются через использование системных вызовов. Соответствующая целая переменная располагается внутри адресного пространства ядра операционной системы. Операционная система обеспечивает атомарность операций $P$ и $V$, используя, например метод запрета прерываний на время выполнения соответствующих системных вызовов. Если при выполнении операции $P$ заблокированными оказались несколько процессов, то порядок их разблокирования может быть произвольным (например, FIFO).

**Мониторы.** Несмотря на свою простоту семафоры в том виде, в котором они предложены Дейкстрой, сложно анализировать на современных платформах из-за непредсказуемости чередования атомарных операций, которые планируются на уровне исполнительного узла (процессора). В 1974 г. Хоаром (Hoare) было предложено использовать более эффективную технику [7], названную впоследствии *монитором Хоара*.

Мониторы представляют собой тип данных, который может быть с успехом внедрен в объектно-ориентированные языки программирования. Монитор обладает своими собственными переменными, определяющими его состояние. Значения этих переменных извне монитора могут быть изменены только с помощью вызова функций-методов, принадлежащих монитору. В свою очередь, эти функции-методы могут использовать в своей работе только данные, находящиеся внутри монитора, и свои параметры. Важной особенностью мониторов является то, что в любой момент времени только один процесс может

быть активен, т. е. находиться в состояниях "готовность" или "исполнение" внутри данного монитора. Поскольку мониторы представляют собой особые конструкции языка программирования, то компилятор может отличить вызов функции, принадлежащей монитору, от вызовов других функций и обработать его специальным образом, добавив к нему пролог и эпилог, реализующие взаимоисключение. Так как обязанность конструирования механизма взаимоисключений возложена на компилятор, а не на программиста, работа программиста при использовании мониторов существенно упрощается, а вероятность появления ошибок становится меньше.

Однако для того чтобы в полном объеме реализовать решение задач, возникающих при взаимодействии процессов, одних только взаимоисключений не достаточно. Дополнительно необходимы средства организации очередности процессов. Для этого в мониторах было введено понятие условных переменных (condition variables), над которыми можно совершать две операции – wait и signal, до некоторой степени похожие на операции $P$ и $V$ над семафорами.

Если функция монитора не может выполняться дальше, пока не наступит некоторое событие, она выполняет операцию wait над какой-либо условной переменной. При этом процесс, выполнивший операцию wait, блокируется, становится неактивным и другой процесс получает возможность войти в монитор.

Когда ожидаемое событие произошло, другой процесс внутри функции-метода совершает операцию signal над той же самой условной переменной. Это приводит к пробуждению ранее заблокированного процесса, и он становится активным. Если несколько процессов дожидались операции signal для данной переменной, то активным становится только один из них. Для того чтобы не оказалось двух процессов – разбудившего и пробужденного, одновременно активных внутри монитора, Хоар предложил, чтобы пробужденный процесс подавлял исполнение разбудившего процесса, пока последний сам не покинет монитор. Несколько позже Хансен (Hansen) предложил другой механизм: разбудивший процесс покидает монитор немедленно после исполнения операции signal.

Реализация мониторов требует разработки специальных языков программирования и компиляторов для них. Мониторы встречаются в таких языках, как параллельный Евклид, параллельный Паскаль, Java и т. д. Эмуляция мониторов с помощью системных вызовов для обычных широко используемых языков программирования не так проста, как эмуляция семафоров. Поэтому

можно пользоваться еще одним механизмом со скрытыми взаимоисключениями – передачей сообщений. Такой подход используется в некоторых ОС реального времени, например QNX.

**Сообщения.** Для прямой и непрямой адресаций достаточно двух примитивов, чтобы описать передачу сообщений по линии связи, – send и receive.

Примитивы send и receive уже имеют скрытый механизм взаимоисключения. Более того, в большинстве систем они уже имеют скрытый механизм блокировки при чтении из пустого буфера и при записи в полностью заполненный буфер. Надо отметить, что несмотря на простоту использования передача сообщений в пределах одного компьютера происходит существенно медленнее, чем работа с семафорами и мониторами.

### *2.2.7. Методы диспетчеризации*

Между принципами организации распределенных информационных систем и традиционных централизованных существует ряд принципиальных отличий. Одними из колючевых являются управление несколькими одновременно выполняемыми процессами, обрабатывающими информацию, и необходимость применять средства передачи сообщений между этими процессами, а также средства синхронизации данных процессов. Далее приведены основные задачи, решаемые алгоритмами планирования:

- общие задачи:
  - справедливость распределения времени,
  - применение политик планирования,
  - обеспечение приоритетов выполнения,
  - поддержание балланса,
  - контроль использования вычислительных ресурсов;
- задачи систем пакетной обработки:
  - оптимизация пропускной способности системы,
  - минимизация времени обработки,
  - оптимизация нагрузки;
- задачи интерактивных и событийно-управляемых систем:
  - минимизация времени отклика на внешние события,
  - обеспечение оперативности отклика интерфейса, пользователя при максимизации загрузки процессора фоновыми задачами;
- задачи систем реального времени:
  - гарантирование времени отлика на прерывания,
  - применение нескольких планировщиков для задач разных уровней.

В случае нескольких процессов, каждый из которых разделен на несколько потоков, реализуются два уровня параллелизма – на уровне потоков и на уровне процессов. Планирование в таких системах существенно зависит от того, поддерживаются ли потоки на уровне пользователя, на уровне ядра либо и на том и на другом уровнях. Основное различие между реализацией потоков на уровне пользователя и реализацией на уровне ядра состоит в производительности. С другой стороны, при реализации потоков на уровне пользователя блокировка потока на устройстве ввода–вывода блокирует весь процесс, чего не случается с потоками на уровне ядра.

Еще одним важным фактором является возможность совместного использования потоков на уровне пользователя и специализированного планировщика потоков. В целом специализированные планировщики потоков лучше управляют приложениями, чем ядро.

На мультипроцессорных системах планирование двумерно, поэтому нужно решать, какой процесс и на котором центральном процессоре запустить. В некоторых системах все процессы являются независимыми, тогда как в других системах они формируют группы. Примером первой ситуации является система реального времени, в которой независимые пользователи запускают независимые процессы. Эти процессы не связаны друг с другом, и планирование каждого из них не зависит от остальных процессов. Простейший алгоритм планирования независимых процессов (или потоков) состоит в поддержании единой структуры данных для готовых процессов – списка или множества списков для процессов с различными приоритетами.

Достоинствами единой структуры данных планирования, используемой всеми центральными процессорами, являются обеспечение процессорам режима разделения времени подобно однопроцессорной системе, а также исключение ситуации, когда один центральный процессор простаивает, в то время как другие процессоры перегружены.

Недостатки – потенциальный рост конкуренции за структуру данных планирования по мере увеличения числа центральных процессоров; рост накладных расходов на выполнение переключения контекста, когда процесс блокируется, ожидая выполнения операции ввода–вывода (переключение контекста также может случиться, когда истечет квант процесса).

### *2.2.8. Двухуровневое планирование*

При двухуровневом алгоритме планирования в момент создания процесс назначается конкретному центральному процессору, например наименее

загруженному в данный момент, – это верхний уровень алгоритма. В результате каждый центральный процессор получает свой набор процессов. Действительное планирование процессов находится на нижнем уровне алгоритма и выполняется отдельно каждым центральным процессором с использованием приоритетов или других средств. Выполнение процессов на одном и том же центральном процессоре максимизируют родственность кэша. Но если у какого-либо центрального процессора нет работы, у загруженного работой процессора процесс отнимается и отдается центральному.

Достоинства двухуровневого планирования: равномерное распределение нагрузки среди имеющихся центральных процессоров по возможности использует преимущество родственности кэша; у каждого центрального процессора есть свой собственный список свободных процессов; конкуренция за списки свободных процессов минимизируется, так как попытки использовать список другого процессора происходят относительно нечасто.

### *2.2.9. Разделение пространства*

Иной подход к планированию мультипроцессоров может быть использован, если процессы связаны друг с другом каким-либо способом. Планирование нескольких потоков на нескольких центральных процессорах называется совместным использованием пространства, или разделением пространства. В момент создания группы связанных потоков планировщик проверяет, есть ли свободные центральные процессоры по количеству создаваемых потоков. Если свободных процессоров достаточно, каждому потоку выделяется собственный (т. е. работающий в однозадачном режиме) и все потоки запускаются. Если процессоров недостаточно, ни один из потоков не запускается, пока не освободится достаточное количество центральных процессоров. Каждый поток выполняется на своем процессоре вплоть до завершения, после чего все центральные процессоры возвращаются в пул свободных процессоров. Если поток оказывается заблокированным операцией ввода–вывода, он продолжает удерживать центральный процессор, который простаивает до тех пор, пока поток не сможет продолжать свою работу. При появлении следующего пакета потоков применяется тот же алгоритм.

### *2.2.10. Бригадное планирование*

В любой момент времени множество центральных процессоров статически разделяется на несколько подмножеств, на каждом из которых выполняются потоки одного процесса. Другой подход к этой модели состоит в том, что-

бы активно управлять степенью распараллеливания процессов и динамически изменять размер групп процессоров, чтобы добиться лучшего соответствия текущей нагрузке. Бригадное планирование подразумевает следующее:

- Группы связанных потоков планируются как одно целое (бригада).
- Все члены бригады запускаются одновременно на разных центральных процессорах с разделением времени.
- Все члены бригады начинают и завершают свои временны́е интервалы вместе.

Бригадное планирование работает благодаря синхронности работы всех центральных процессоров. Это означает, что время разделяется на дискретные кванты. В начале каждого нового кванта все центральные процессоры перепланируются заново и на каждом процессоре запускается новый поток, в середине кванта планирование не выполняется. Если какой-либо поток блокируется, его центральный процессор простаивает до конца кванта времени. Все потоки процесса работают, по возможности, вместе.

### *2.2.11. Планирование многомашинных систем*

На мультипроцессоре все процессы располагаются в общей памяти. В многомашинной системе у каждого узла есть своя собственная память и свой собственный набор процессов. Как только процесс назначается какому-либо узлу, может использоваться любой локальный алгоритм планирования.

**Балансировка нагрузки.** К известным свойствам процессов относятся процессорные требования, количество памяти и объем обмена информацией с другими процессами. Возможные задачи: минимизация потерянного процессорного времени, вызванного отсутствием локальной работы; минимизация суммарного сетевого трафика; гарантирование справедливого распределения ресурсов для пользователей и процессов.

**Детерминистический графовый алгоритм.** Если количество процессов больше количества центральных процессоров, то некоторые процессы должны быть назначены каждому процессору. Идея заключается в том, чтобы минимизировать сетевой трафик. Система может быть представлена в виде взвешенного графа, каждая вершина которого представляет собой процесс, а каждая дуга – поток сообщений между двумя процессами. Математически проблема сводится к тому, чтобы найти способ разбиения графа на $k$ непересекающихся подграфов при определенных ограничениях, накладываемых на подграфы (например, суммарные требования центральных процессоров и памяти для подграфа не должны превышать некоторого установленного пре-

дела). Дуги, идущие от одного подграфа к другому, представляют сетевой трафик.

Цель состоит в том, чтобы найти такой вариант разбиения графа на подграфы, который минимизирует сетевой трафик при выполнении всех требований. Задачи такого типа решаются при помощи классических алгоритмов оптимизации потоков и путей на графах с использованием автоматных формализмов.

## 2.3. Память в распределенных системах

Системы управления памятью можно разделить на два класса: перемещающие процессы между оперативной памятью и диском во время их выполнения, (т. е. осуществляющие подкачку процессов целиком (swapping) или использующие страничную подкачку (paging)) и неперемещающие.

В мультипроцессорных системах с общей оперативной памятью каждый центральный процессор обладает равным доступом к всей физической памяти. Программа, работающая на любом центральном процессоре, видит разбитое на страницы виртуальное адресное пространство. Основное свойство межпроцессорного обмена информацией: один центральный процессор пишет данные в память, а другой – считывает их из памяти.

У всех мультипроцессоров каждый центральный процессор может адресоваться к всей памяти. По характеру доступа к памяти эти машины делятся на два класса: мультипроцессоры, у которых каждое слово данных может быть считано с одинаковой скоростью, называемые UMA-мультипроцессорами (Uniform Memory Access – однородный доступ к памяти), и мультипроцессоры NUMA (NonUniform Memory Access – неоднородный доступ к памяти), не обладающие этим свойством.

Простейшая архитектура мультипроцессоров основана на идее общей шины. Несколько центральных процессоров и несколько модулей памяти одновременно используют одну шину, которая при большом количестве центральных процессоров будет постоянно занята, а производительность системы будет полностью ограничена ее пропускной способностью. Решение проблемы состоит в том, чтобы добавить каждому центральному процессору кэш. Обращения к шине снижаются, и система сможет поддерживать большее число центральных процессоров.

Еще один вариант архитектуры мультипроцессоров: у каждого центрального процессора имеется не только кэш, но и локальная собственная память,

с которой он соединен по выделенной шине. Для оптимального использования подобной конфигурации компилятор должен поместить все неизменяемые данные и локальные переменные в локальные модули памяти. При этом общая память используется только для общих модифицируемых переменных. В большинстве случаев такая схема использования памяти сильно снижает трафик по шине, но для ее реализации требуются специальные действия со стороны компилятора.

Довольно простой схемой соединения центральных процессоров и $k$ модулей памяти является координатный коммутатор. *Координатный коммутатор* представляет собой неблокирующую сеть, ни один центральный процессор которой не получает отказов соединений по причине занятости какого-либо переключателя (при условии, что сам требующийся модуль памяти свободен). При такой схеме не требуется планирования доступа к памяти. Координатный коммутатор имеет один недостаток – число переключателей растет пропорционально квадрату от числа центральных процессоров.

**Память в многомашинных комплексах и системах.** В многомашинных системах применяются две коммутационные схемы. В первой из них каждое сообщение сначала разбивается (либо программным обеспечением пользователя, либо сетевым интерфейсом) на отдельные фрагменты – *пакеты*.

В коммутационной схеме, называемой *коммутацией пакетов с промежуточным хранением*, пакет сначала посылается сетевой картой источника первому узлу. Он передается побитно, и, когда прибывает весь пакет, копируется на следующий коммутатор. Когда пакет прибывает на коммутатор, соединенный с пунктом назначения, он копируется на сетевую карту узла-получателя и наконец попадает в оперативную память этого узла.

Гибридные сети, создаваемые для уменьшения задержки передачи сообщения по сети, обладают свойствами коммутации пакетов и свойствами коммутации каналов. Каждый пакет может логически разбиваться на более мелкие единицы. Как только первая такая единица прибывает на коммутатор, она может пересылаться дальше на следующий коммутатор, прежде чем прибудет остальная часть пакета.

Другой режим коммутации – *коммутация каналов*, заключается в том, что первый коммутатор сначала устанавливает путь через все коммутаторы до коммутатора-получателя. Как только путь установлен, биты передаются от источника к приемнику без остановок. При этом промежуточного хранения не производится. Для коммутации каналов требуется фаза начальной установки,

занимающая некоторое время, но по ее завершении весь процесс передачи данных идет быстрее. Когда сообщение отправлено, путь должен быть снова разорван.

Существует вариант коммутации каналов, называемый *чередующейся маршрутизацией*, при которой каждый пакет разбивается на подпакеты и первый подпакет начинает передаваться еще до того, как был установлен полный путь. Многие соединительные сети являются синхронными, поэтому, как только начинается передача одного пакета, его биты должны передаваться с постоянной скоростью. Если пакет находится в оперативной памяти, постоянство потока гарантировать невозможно, поскольку на шине памяти может существовать и другой трафик.

Наличие выделенной памяти в интерфейсной карте устраняет эту проблему. Недостатком высокопроизводительной связи в многомашинных системах является излишнее копирование пакетов. Простейшая схема многомашинной системы представляет собой две интерфейсные карты, одна из которых отображается на пространство пользователя, а вторая – на пространство ядра и используется операционной системой. Процессы на разных центральных процессорах многомашинной системы общаются, отправляя друг другу сообщения.

Операционная система предоставляет способ отправки и получения сообщения, а библиотечные процедуры обеспечивают доступность этих системных вызовов для пользовательских процессов. В более сложной форме передача сообщений скрыта от пользователя под видом вызова удаленной процедуры.

Служба связи может быть минимизирована до двух стандартных вызовов – один для отправки сообщений и один для их получения. Так как мультикомпьютеры статичны, число их центральных процессоров фиксировано, и проще всего использовать адреса, состоящие из двух частей: номера центрального процессора и номера процесса или порта на данном центральном процессоре.

Описанные вызовы представляют собой *блокирующие* вызовы (иногда называемые *синхронными* вызовами). Когда процесс обращается к процедуре send, он указывает адресат и буфер, данные из которого следует послать указанному адресату. Пока сообщение посылается, передающий процесс блокирован (т. е. приостановлен). Команда, следующая за обращением к процедуре send, не выполняется до тех пор, пока все сообщение не будет послано. Такой механизм активно используется в системах жесткого реального времени.

Альтернатива этому процессу – *неблокирующие (асинхронные)* вызовы. Если процедура send является неблокирующей, то она возвращает управление вызывающему ее процессу практически немедленно, прежде чем сообщение будет отправлено. Главное достоинство заключается в том, что отправляющий процесс может продолжать вычисления параллельно с передачей сообщения. Это позволяет избежать простоя центрального процессора (при условии, что других готовых к работе процессов нет). Недостаток – отправителю нельзя изменять содержимое буфера сообщения до тех пор, пока это сообщение не будет полностью отправлено. Если у отправляющего процесса нет возможности узнать, что передача уже выполнена, то он никогда не будет уверен, можно ли уже пользоваться буфером.

Как и операция send, операция receive также может быть блокирующей и неблокирующей. Блокирующий вызов просто приостанавливает процесс до прибытия сообщения. Если на центральном процессоре одновременно работают несколько потоков, то такой подход является наиболее простым. Неблокирующий вызов receive сообщает ядру, где расположен буфер, и почти сразу же возвращает управление. По прибытии сообщения для извещения процесса может использоваться прерывание. Более приемлемым решением является периодическое обращение к почтовому ящику при помощи процедуры опроса, сообщающей, пришли ли какие-либо сообщения.

Информация между вызывающим процессом и вызываемой процедурой может передаваться через параметры, а также возвращаться в результате процедуры. Такая техника, называется вызовом удаленной процедуры (Remote Procedure Call, RPC). Традиционно вызывающую процедуру называют клиентом, а вызываемую – сервером.

## 2.4. Механизмы защиты

В распределенных системах применяется множество моделей безопасности, которые определяют то, каким образом осуществляется доступ к ресурсам различными категориями потребителей.

| Домен | F1 | F2 | F3 | Printer |
|---|---|---|---|---|
| D1 | read | – | – | – |
| D2 | – | – | – | – |
| D3 | – | read | execute | print |
| D4 | read, write | – | read, write | – |

Наиболее часто используемой моделью обеспечения безопасности доступа к объектам является *дискреционный доступ*. Главное его достоинство – гибкость, основные недостатки – рассредоточенность управления и сложность

централизованного контроля. Понятие дискреционного доступа базируется на концепции *домена безопасности* (protection domain). Каждый домен определяет набор объектов и типов операций, которые могут производиться над каждым объектом. Возможность выполнять операции над объектом есть права доступа, каждое из которых есть упорядоченная пара <имя_объекта, набор_прав>. Домен, таким образом, есть набор прав доступа. Например, если домен D имеет права доступа <файл F, read, write>, это означает, что процесс, выполняемый в домене D, может только читать или писать в файл F (см. таблицу).

Связь конкретных субъектов, функционирующих в операционных системах, может быть организована следующим образом:

- Каждый пользователь может быть доменом. В этом случае набор объектов, к которым может быть организован доступ, зависит от идентификации пользователя.
- Каждый процесс может быть доменом. В этом случае набор доступных объектов определяется идентификацией процесса.
- Каждая процедура может быть доменом. В этом случае набор доступных объектов соответствует локальным переменным, определенным внутри процедуры. Заметим, что когда процедура выполнена, происходит смена домена.

**Матрица доступа.** Описанная модель безопасности имеет вид матрицы, которая называется *матрицей доступа*. В общем случае матрица доступа будет разреженной, т. е. большинство ее клеток будут пустыми. Хотя существуют структуры данных для представления разреженной матрицы, они не слишком полезны для приложений, использующих возможности защиты. Поэтому на практике матрица доступа применяется редко.

Эту матрицу можно разложить по столбцам, в результате чего получаются списки прав доступа (Access Control List, ACL). В результате разложения по строкам получаются мандаты возможностей (capability list, или capability tickets).

**Список прав доступа (Access Control List).** Каждая колонка в матрице может быть реализована как список доступа для одного объекта. Очевидно, что пустые клетки могут не учитываться. В результате для каждого объекта имеется список упорядоченных пар <домен, набор_прав>, который определяет все домены с непустыми наборами прав для данного объекта. Элементами списка могут быть процессы, пользователи или группы пользователей. При

его реализации широко применяется предоставление доступа по умолчанию для пользователей, права которых не указаны. Например, в Unix все субъекты-пользователи разделены на три группы (владелец, группа и остальные) и для членов каждой группы контролируются операции чтения, записи и исполнения (rwx). В итоге имеем ACL – 9-битный код, который является атрибутом разнообразных объектов Unix.

**Мандаты возможностей (capability list).** Если матрицу доступа хранить по строкам, т. е. если каждый субъект хранит список объектов и для каждого объекта – список допустимых операций, то такой способ хранения называется "мандаты", или "перечни возможностей" (capability list). Примерами систем, использующих перечни возможностей, являются Hydra, Cambridge CAP System.

**Другие способы контроля доступа.** Иногда применяется комбинированный способ. Например, в том же Unix на этапе открытия файла происходит анализ ACL (операция open). В случае благоприятного исхода файл заносится в список открытых процессом файлов и при последующих операциях чтения и записи проверки прав доступа не происходит. Список открытых файлов можно рассматривать как перечень возможностей.

Существует также схема "lock-key" [8], которая является компромиссом между списками прав доступа и перечнями возможностей. В этой схеме каждый объект имеет список уникальных битовых шаблонов (patterns), называемых locks. Аналогично каждый домен имеет список уникальных битовых шаблонов, называемых ключами (keys). Процесс, выполняющийся в домене, может получить доступ к объекту, только если домен имеет ключ, который соответствует одному из шаблонов объекта.

Как и в случае мандатов, список ключей для домена должен управляться ОС. Пользователям не разрешается проверять или модифицировать списки ключей (или шаблонов) непосредственно. Более подробно данная схема изложена в [9].

## 2.5. Контрольные вопросы

1. Дайте определение процесса.
2. Дайте определение потока.
3. Какие свойства распределенных систем вам известны?
4. Приведите пример прозрачности для распределенной системы.
5. Почему открытость – необходимое свойство любой распределенной системы?
6. Масштабируемость каких параметров наиболее важна для распределенной системы?
7. Что в себя включает понятие безопасности?
8. Что такое синхронизация?

9. Какая обработка называется асинхронной?
10. В каких состояниях может находиться процесс?
11. В чем заключаются отличия потоков и процессов?
12. Перечислите задачи планировщика.
13. Каковы причины применения потоков?
14. Дайте определение логических часов.
15. Приведите пример работы алгоритма Лампорта для согласования времени между двумя системами.
16. Что такое транзакция?
17. Какие примитивы достаточны для описания транзакции?
18. Какие механизмы синхронизации вам известны?
19. Приведите алгоритм работы семафора.
20. В чем отличие монитора от семафора?
21. Приведите основные задачи алгоритмов планирования.
22. Что такое двухуровневое планирование?
23. Когда оправдано применение бригадного планирования? Приведите примеры.
24. Что такое балансировка нагрузки?
25. Что такое домен безопасности?

# 3. СЕРВИС-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА

Применение СОА для построения информационных систем востребовано в сфере автоматизации деятельности крупных предприятий и неразрывно связано с их функциональной структурой. Информационная модель предприятия должна строиться не столько для того, чтобы повторить организационную структуру, сколько для того, чтобы модифицировать имеющиеся подходы к организации бизнеса и привести их в стройную систему взаимодействующих компонентов. Подтверждением может служить тот факт, что в основе всех современных стандартов управления качеством лежат те же самые идеи, что и в основе построения крупных информационных систем.

Сама по себе СОА не является информационным продуктом или законченной технологией, готовой для применения. Это – набор принципов и понятий, позволяющих создавать информационные системы поддержки бизнес-процессов. В основе архитектуры лежит понятие процессного подхода к организации бизнес-процессов, который достаточно полно определен серией стандартов ISO 9000:2000[1].

## 3.1. Модели, основанные на процессном подходе

Стандарт ISO 9000:2000 определяет процесс как совокупность взаимосвязанных и взаимодействующих действий, преобразующих входы в выходы.

[1]ISO 9000 Definitions, 2000, http://praxiom.com/iso-definition.htm

Для понимания природы бизнеса важно понимать механизмы *генезиса* его процессов, т. е., какие процессы инициируются в связи с взаимодействием заинтересованных сторон. Это значит, что для разумной организации деятельности необходимо выделить требования потребителя (цели, по которым оцениваются результаты деятельности), ресурсы и последовательность операций, направленных на преобразование ресурсов в продукцию или в услуги, соответствующие заявленным целям.

Такая последовательность действий и представляет собой процесс, который может быть описан с различной степенью подробности. Причем данный способ описания обладает свойством *фрактальности*: определенные части процесса верхнего уровня сами обладают всеми свойствами процессов. Хотя такое деление не бесконечно, обычно имеет смысл выделять несколько уровней. Так, например, можно говорить о процессе на уровне организации, на уровне структурного подразделения организации, на уровне команды и на уровне рабочего места. С появлением идеи описания бизнес-процессов и соответствующих им информационных систем на основе процессного подхода появился ряд преимуществ, позволяющих делать это эффективнее.

Первое преимущество связано с возникновением языка описания деятельности, доступного и понятного всем вовлеченным в процесс. Это позволило разрушить или по крайней мере снизить барьеры, обусловленные различиями в терминологии, в статусе внутри организации, в образовании ее работников и т. п.

Второе преимущество – возможность простой и наглядной графической интерпретации деятельности, что существенно облегчает, например реализацию такого важного требования стандарта ISO 9000:2000, как прослеживаемость. Кроме того, наглядность создает предпосылки для обеспечения прозрачности информационных потоков.

Третье преимущество обусловлено более простым и надежным определением точек контроля и критических точек в процессе. Облегчение возникает за счет разбиения процесса на подпроцессы.

Процессный подход [10] означает, что управление сложными системами поддержки бизнес-процессов, каковыми являются композитные информационные сервис-ориентированные приложения, основано на управлении процессами. Для этого процессы должны быть определены, их параметры должны быть описаны и документированы. Процессы выстраиваются в цепочку, выход одного процесса является входом другого процесса, таким образом процессы

описывают всю систему целиком. Процессы обладают специфическими свойствами:

- процесс должен быть однозначно идентифицирован;
- должны быть описаны объекты на входе и на выходе процесса;
- должны быть описаны ресурсы, потребляемые процессом;
- для оценки качества должны быть определены *метрики*.

Каждый элемент системы привязан к процессу через определенную роль, например: *владелец процесса; потребитель процесса; наблюдатель*.

К описанию и оптимизации бизнес-процессов существует множество подходов. Рассмотрим три наиболее известных три из них[1].

Первый подход ориентирован на подробное описание последовательности действий, направленных на достижение результата; второй позволяет провести синтез бизнеса компании и сгруппировать работы; третий рассматривает процессы как последовательный вклад в создание продукта, предназначенный для поиска конкурентного преимущества, иными словами, полагается на цепочку создания ценностей.

### *3.1.1. Oracle Business Models*

Первый из указанных подходов наиболее характерно отражен в модели Oracle Business Models (OBM). Построение такой модели и использование OBM применяются в качестве инструмента при работе над проектами внедрения ERP-систем на крупных предприятиях для сравнения существующих и будущих бизнес-процессов.

Модель практически распределяет деятельность предприятия по тем областям деятельности (по предмету деятельности), которые нашли свое отражение в IT-решениях Oracle. Она в общем виде полностью описывает предприятие вплоть до процессов нижнего уровня, т. е. отражает последовательность абстрактных операций условного предприятия с указанием логических операторов. Иначе говоря, после локализации (адаптации под конкретную деятельность) модели OBM получается сбалансированное описание бизнес-процессов в едином формате.

В то же время модели типа OBM отличаются существенным недостатком: предприятие описывается в терминах функциональной деятельности (выделение по предмету деятельности). Поэтому при декомпозиции модели процессы и операции на нижнем уровне описываются как деятельность, рас-

[1]Более подробный обзор можно найти на сайте www.management.com.ua

пределенная по различным функциональным подразделениям и специалистам, что нарушает главный принцип реинжиниринга: "один процесс – одно подразделение – один бюджет – один владелец процесса". Принцип "процессного" управления впервые сформулирован был Саймоном [11], а в дальнейшем развит Хаммером и Чампи [12].

С другой стороны, у описанной модели есть существенное преимущество – она позволяет сохранять компетенции и организационное построение компании, придавая ему новый смысл и автоматизируя наиболее трудоемкие и рутинные операции.

### *3.1.2. Универсальные 13- и 8-процессные модели*

Несмотря на фундаментальность трудов Хаммера, Чампи и Саймона, предложенный ими подход сложен для технической реализации, так как трудоемность детальной декомпозиции бизнес-окружения на процессы в условиях неоднозначности функций высока. Для того чтобы повысить скорость разработки информационных систем данного уровня, были созданы упрощенные, или "типовые" модели процессной декомпозиции. Примерами такого подхода служат универсальные 13- и 8-процессные модели. В отличие от предыдущей модели, в данных моделях осуществляется выделение процессов по результатам деятельности (а не по предмету).

Особенностью универсальных процессных моделей являются четкое агрегирование работ *по результату* и точное следование принципу "один процесс – один результат – одно подразделение – один бюджет – один владелец процесса". Такие модели позволяют разрабатывать и внедрять *плоские* организационные структуры, ограничиваясь тремя уровнями декомпозиции, но опять же предъявляют крайне жесткие требования к квалификации исполнителей, плохо понимаются заказчиками и крайне сложны в разработке – в силу высокой абстрагированности принципов и понятий, применяемых при моделировании. В то же время эти модели в случае их внедрения позволяют существенно сокращать персонал, действительно оптимизировать деятельность предприятия, придавать прозрачность и управляемость бизнесу.

### *3.1.3. Модель Портера*

Третий подход основывается на описанной Портером цепочке создания ценности [13], [14], в которой выделяются основные бизнес-процессы, обеспечивающие операционный цикл производства продукта, выполняющиеся последовательно и поддерживающие бизнес-процессы, которые обеспечи-

вают функционирование бизнес-системы и сопровождают создание продукта на всем его протяжении.

Однако данная модель является демонстрационной, служащей для понимания сущности деятельности, так как не получила дальнейшего теоретического развития и обоснования.

Отдельно следует остановиться на определении границ звеньев цепочки. Портер предположил, что границы находятся там, где каждый внутренний подпроцесс что-то добавляет к ценности продукта, но это не делает работу по разграничению процессов проще. Вместе с тем цепочка создания ценности является важным инструментом, при помощи которого выявляются стратегические аспекты, уникальные с точки зрения конкурентного преимущества.

Приведенными моделями область моделирования бизнес-процессов не исчерпывается, однако они отражают ключевые подходы к моделированию. Развитие моделей, применяемых для проектирования основной структуры корпоративных систем, продолжается, и, скорее всего, в ближайшее время будут появляться новые подходы. Следует отметить, что все представленные модели предполагают наличие некоторой *стратегии* как основного условия для разделения бизнеса по выделяемым процессам. Понятие стратегии тесно связано с понятием *оркестровки сервисов*, которое будет раскрыто далее.

## 3.2. Архитектура, ориентированная на сервисы

Появление архитектуры, ориентированной на сервисы, является следствием новых задач и потребностей, возникающих при создании и эксплуатации современных информационных систем:

- оперативного реагирования на изменение условий ведения бизнеса и быстрой адаптации под новые бизнес-задачи;
- оптимизации управления бизнес-процессами;
- эффективного обеспечения внешних взаимодействий.

В основе СОА лежит понятие сервисов, являющихся базовыми элементами для построения бизнес-приложений и обеспечения взаимодействия между ними. В архитектуре СОА приложение рассматривается как сервис, который можно найти и далее получить доступ к нему через локальную сеть или Интернет. Приложение может реализовать определенную функцию или набор функций как самостоятельно, так и обращением к другим сервисам. При этом данное приложение может быть доступно сторонним пользователям в качестве сервиса. Сервисы являются автономными, но для того чтобы их можно

было найти и использовать, они снабжены соответствующими интерфейсами. Идея сервисов в информационных системах имеет довольно длинную историю. Хорошо известны следующие подходы:

- Java RMI (Java Remote Method Invocation);
- CORBA(Common Object Request Broker Architecture);
- DCE (Distributed Computing Environment);
- Microsoft DCOM (Distributed Component Object Model).

Следует отметить, что СОА имеет имеет одно существенное отличие от вышеперечисленных, а именно – наивысшую степень абстракции понятия сервиса. Это отличие позволяет вести речь о разработке композитных распределенных приложений без учета каких бы то ни было деталей реализации.

### *3.2.1. Определения СОА*

Разработка композитных приложений, ориентированных на сервисы, – довольно молодая область программной инженерии и терминология, используемая различными специалистами, зачастую является противоречивой. В качестве введения в понятие сервис-ориентированной архитектуры приведем взгляды известных аналитиков и разработчиков на определение СОА, опубликованые в [15].

**"Stencil Group".** СОА, как считают специалисты данной компании, представляет собой закономерный этап эволюции корпоративных систем. Они выделяют такие ключевые черты СОА:

- Архитектура является распределенной, т. е. функциональные элементы приложений могут быть распределены по множеству вычислительных систем и способны к взаимодействию с использованием локальных или глобальных сетей. В частности, web-сервисы позволяют использовать существующие протоколы, например HTTP.
- Архитектура строится с использованием слабосвязанных интерфейсов. Обычно приложения проектируются в расчете на жесткую связь всех элементов. Как следствие система должна иметь целостный проект, его изменения в процессе эксплуатации затруднительны. Работу компонентов в слабосвязанных системах проще координировать, системы проще реконфигурировать.
- Архитектура базируется на общепринятых отраслевых стандартах.
- Архитектура проектируется с ориентацией на процессы (process-centric) с использованием сервисов, каждый из которых ориентирован на решение отдельных задач (task-centric).

Такое определение СОА несколько механистично и затрагивает только технические вопросы практической реализации с исключением базовой идеи.

**"Gartner Group".** В "Gartner" в качестве основной черты СОА видят деление ее на два или более уровня. Обращенный к пользователю презентационный уровень отделен от внутреннего, где реализуется бизнес-логика. Последний создается из крупных блоков (coarse-grained chunk), которые обеспечивают сервис клиентским программам презентационного уровня. В логический уровень могут включаться компоненты управления бизнес-процессами (Business Process Management, BPM).

Практически, как считают аналитики этой компании, СОА можно реализовать в том числе и многозвенной клиент-серверной архитектурой, но не двухуровневой архитектурой с толстым клиентом, который совмещает как логику, так и презентацию.

**"International Data Corp".** Аналитики "International Data Corp" определяют СОА как архитектуру, предназначенную для свободного размещения функциональных модулей, каждый из которых способен выполнять определенные действия. Эти модули представляют собой описывающие сами себя программные компоненты, достижимые через сеть. Модули публикуют свои интерфейсы таким образом, что их применение не требует знания использованных в них решений и технологий; их удобно воспринимать как черный ящик.

СОА можно реализовать и без использования web-сервисов, однако сервисы рассматриваются в качестве основного средства для создания подобной архитектуры.

**"Barry & Associates".** Данная консалтинговая компания определяет СОА как набор взаимодействующих между собой модулей. Модули могут просто обмениваться между собой данными или каким-то образом координировать свои действия.

Архитектура, ориентированная на сервисы, не представляет собой ничего нового, считают здесь. Ее первыми примерами стали брокеры объектных запросов, основанные на спецификации CORBA.

**"ThoughtWorks".** В компании "ThoughtWorks" утверждают: СОА представляет собой набор сервисов, которые совместно образуют систему, призванную заменить монолитные корпоративные приложения типа ERP. Примерно такого же мнения придерживаются другие аналитики, которые считают, что СОА представляет собой набор модулей для обеспечения бизнес-процессов,

дополняя их методологией и набором средств интеграции приложений. Такой разброс во мнениях экспертов закономерен. СОА – молодая технология, и взгляд на ее многие технические и идеологические детали еще не стабилизировался.

Итак, можно говорить о том, что *сервис-ориентированная архитектура* – это определенная тенденция развития распределенных информационных систем, направленного на высокоуровневое манипулирование автономными и композитными сервисами, позволяющими организовать построение приложений управления бизнес-процессами в гетерогенной среде.

### *3.2.2. Базовая модель взаимодействия*

Сервис-ориентированная архитектура определяется моделью взаимоотношений между тремя основными сторонами – поставщиком, потребителем и посредником сервиса. Поставщик сервиса публикует описание сервиса и обеспечивает его реализацию. Потребитель сервиса для нахождения описания сервиса может напрямую использовать универсальный идентификатор ресурса (URI) или же может найти описание в реестре сервиса с последующей его привязкой и вызовом. Посредник сервиса обеспечивает и обслуживает реестр сервиса.

СОА определяет набор параметров и рекомендаций для создания слабосвязанных бизнес-ориентированных сервисов. Эти сервисы из-за разделения участия между описанием, реализацией и привязкой обеспечивают высокую гибкость в реагировании на новые бизнес-ситуации. Более формально взаимодействие компонетов СОА-системы можно представить в виде UML-диаграммы, приведенной на рис. 3.1.

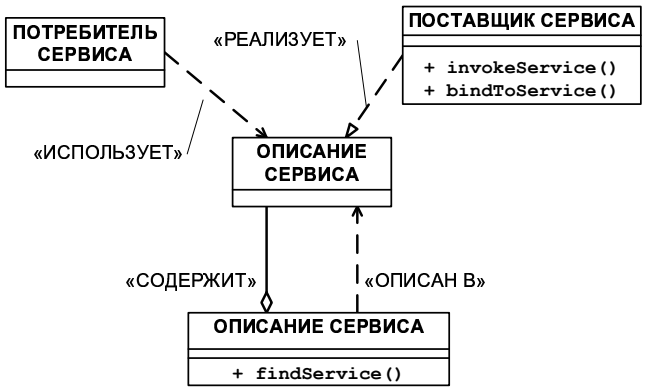


*Рис. 3.1*

СОА является масштабируемой на уровне предприятия для соединения ресурсов по требованию. Она состоит из набора ориентированных на бизнес IT-сервисов, которые коллективно удовлетворяют задачам и бизнес-процессам организации. Эти сервисы можно группировать в композитные приложения и вызывать их через стандартные протоколы.

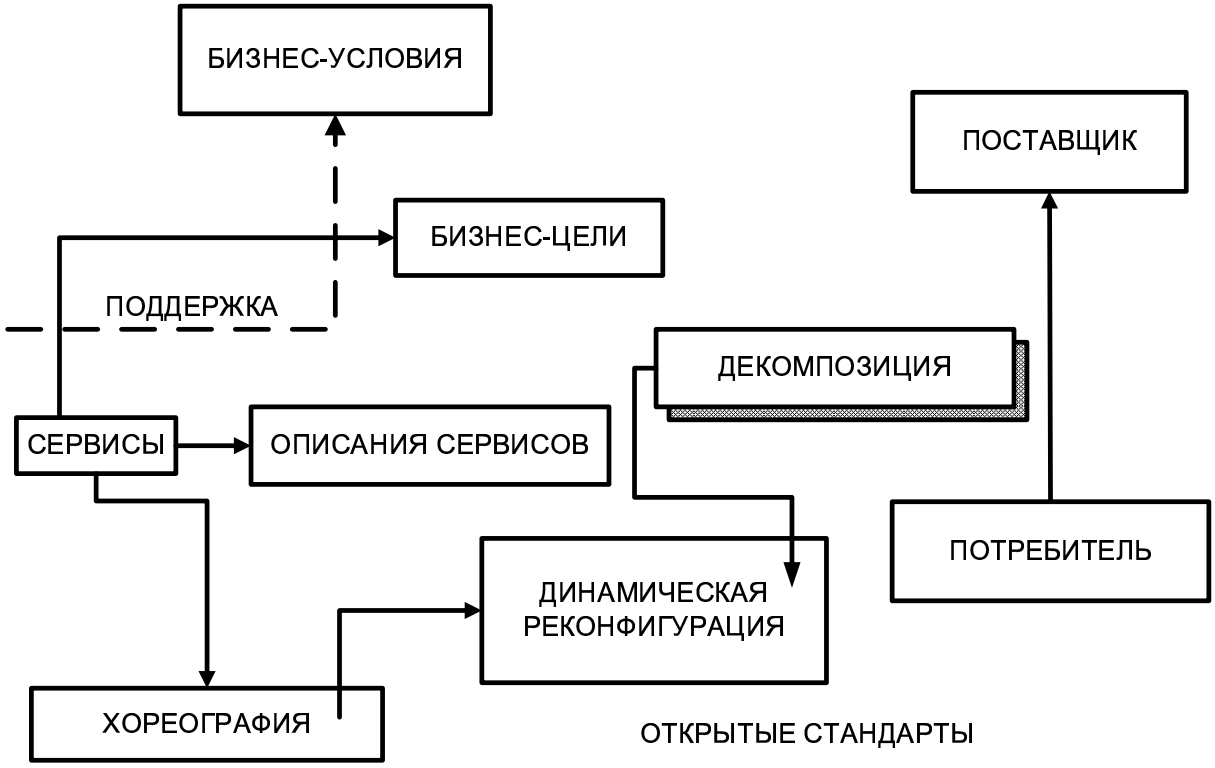


*Рис. 3.2*

На рис. 3.2 приведена схема сборки композитных приложений на основе сервисов.

Сервис является программным ресурсом, поддающимся обнаружению, с собственным внешним описанием. Это описание сервиса доступно для поиска, привязки и вызова потребителем. Поставщик сервиса выпускает реализацию описания сервиса и кроме того предоставляет потребителю описание требований по качеству его обслуживания. Сервисы, в идеале, должны быть управляемы декларативными политиками и, соответственно, поддерживать динамически перенастраиваемый архитектурный стиль[1]. (Именно эта возможность позволяет разрабатывать композитные приложения на основе сервисов.)

Быстрота адаптации бизнеса к изменениям достигается IT-системами, предлагаемыми СОА. Эти системы являются гибкими преимущественно в части разделения интерфейса, реализации и привязки (протоколов). Подобная гибкость позволяет снизить зависимость от выбора поставщика сервиса. Здесь требования могут быть функциональными и нефункциональными.

[1]См.: A. Arsanjani, Service-oriented modeling and architecture, 2004, http:// www-128.ibm.com /developerworks/ webservices/library/ws-soa-design/

Сервисы можно повторно использовать между внутренними единицами бизнеса или между распределенными бизнес-партнерами в так называемой модели фрактальной реализации. *Фрактальная реализация* заключается в возможности композитного применения архитектурным стилем своих моделей и ролей, связанных с участниками, к собственной модели взаимодействий. Это применимо как к одному уровню в архитектуре, так и к множеству уровней во всей архитектуре предприятия. В проектах такие взаимодействия могут осуществляться между бизнес-единицами и бизнес-партнерами в распределенной сети однородным и концептуально масштабируемым способом.

**Реестры сервисов.** Модель СОА не зависит от технологий, использующихся для ее реализации, а основным методологически значимым ее компонентом является реестр сервисов. В приведенном на рис. 3.1 асинхронном протоколе общения провайдера и потребителя сервисов он выполняет функции посредника. Провайдер размещает информацию о своих сервисах в реестре, что дает возможность потребителю в любой момент найти необходимый ему сервис. За таким процессом общения скрывается основное качество СОА – *слабая связанность*.

Благодаря этому свойству сервисы обретают мобильность, способность перемещаться с одного сервера на другой, не требуя согласования и координации со всеми потребителями. Естественно, что потребители сервисов в ряде случаев не способны и не должны принимать во внимание регулярное перераспределение ресурсов, обеспечивающих функционирование сервисов.

Позднее связывание также позволяет отложить момент конечной сборки связей до времени исполнения, а не времени разработки программы, что характерно для традиционных монолитных систем. Во время исполнения можно менять параметры связи (такие, как адрес, протокол и канал взаимодействия). Это придает несколько измерений гибкости самой связке между провайдером и потребителем сервиса – соответственно, вызываемым и осуществляющим вызов объектами. В частности, провайдер и потребитель могут исполняться на сколь угодно физически удаленных инфраструктурах. Каждая из систем может иметь собственные параметры жизненного цикла, а любые изменения в системах, не затрагивающие интерфейс сервиса, не требуют остановки ни одной из них.

В СОА сервисы рассматриваются как автономные объекты, управление которыми не централизовано. Это позволяет взаимодействующим посредством сервисов информационным системам развиваться в соответствии с по-

требностями бизнеса, которые потребителям сервисов, как правило, ни только не известны, но и неинтересны. Однако такое развитие было бы невозможно, если бы интерфейс сервиса не был прочно закреплен обоюдным соглашением провайдера и потребителя сервиса.

Одной из отличительных черт СОА является наличие контрактов, описывающих интерфейсы сервисов. *Контракт* представляет собой документ, специфицирующий ожидания сервиса по отношению к его потребителям, и наоборот. Контракты web-сервисов описываются WSDL-документом, в нотации XML определяющим, как потребители должны обращаться к сервису. Использование XML на этом этапе имеет принципиальное значение, позволяя и провайдеру, и потребителю сервиса не зависеть от определенной платформы.

Подобные контракты существовали и до появления web-сервисов. Например, в архитектуре CORBA для описания интерфейса объектов использовался язык IDL, который уступает WSDL по ряду существенных параметров. Главный из них – отсутствие поддержки XML и XML Schema, ставших наиболее распространенными языками разметки передаваемых по сети сообщений и представления моделей данных.

Технические контракты, формулируемые провайдером сервисов, должны быть доступны потенциальным потребителям для интерпретации, анализа и реализации интеграции. Для этого используется специальный реестр, каталогизирующий доступные сервисы.

**Оркестровка и хореография сервисов.** Средства обмена сообщениями, с помощью которых несколько независимых агентов стремятся достичь желаемого состояния, получили название *"хореографии"*, а взаимодействие сервисов – *"оркестровки"*. Для оркестровки (т. е. по сути описания бизнес-логики) разработаны (с участием крупнейших вендоров, таких, как IBM, "Microsoft", "Oracle" и "BEA Systems") специальные средства программирования: BPEL4WS, XLANG, WSFL.

Оркестровка относится к определению бизнес-процесса, который может взаимодействовать с внешними и с внутренними web-сервисами. Происходящие на основе обмена сообщениями взаимодействия включают бизнес-логику и порядок выполнения задач; они могут выходить за границы приложений и организаций, определяя долговременную, транзакционную, многошаговую бизнес-модель.

Оркестровка всегда представляет собой управление с позиций одного участника процесса. Хореография позволяет каждому участнику описать свою

часть взаимодействия. При использовании хореографии отслеживаются последовательности сообщений между несколькими участниками и источниками. Предлагаемые стандарты оркестровки и хореографии должны удовлетворять нескольким требованиям, относящимся к языку описания потока работ бизнес-процесса и к инфраструктуре выполнения процесса. К числу этих требований относятся:

- асинхронный вызов;
- управление исключительными ситуациями;
- обеспечение транзакционной целостности;
- динамичность;
- гибкость
- адаптируемость;
- возможность композиции сервисов более высокого уровня из существующих процессов.

COA как система сложнее других информационных систем, и поэтому она требует для своего описания не одну модель, а несколько (рис. 3.3). Наличие разных моделей позволит обеспечить согласованную работу специалистов разных профилей, согласование различных стандартов, образование структуры стандартов.

Существует несколько архитектурных моделей COA, предложенные консорциумом W3C[1], а именно:

- модель, ориентированная на сообщения (Message Oriented Model, MOM). Сосредоточена на сообщениях, их структуре, способах транспортировки и других компонентах, никак не связанных с причинами обмена сообщениями и их семантикой;
- модель, ориентированная на сервисы (Service Oriented Model, SOM). Сосредоточена на действиях, выполняемых сервисами;
- модель, ориентированная на ресурсы (Resource Oriented Model, ROM). Сосредоточена на системных ресурсах;
- модель политик (Policy Model, PM). Определяет политики, связанные с архитектурой (в основном она представляет собой ограничения, накладываемые на поведение агентов и сервисов, в том числе ограничения, связанные с требованиями безопасности, качества обслуживания);
- модель управления (Management Model, MM).

[1]См.: Web Services Architecture W3C Working Draft 8, August 2003, http:// www.w3.org / TR / 2003 / WD-ws-arch-20030808 /

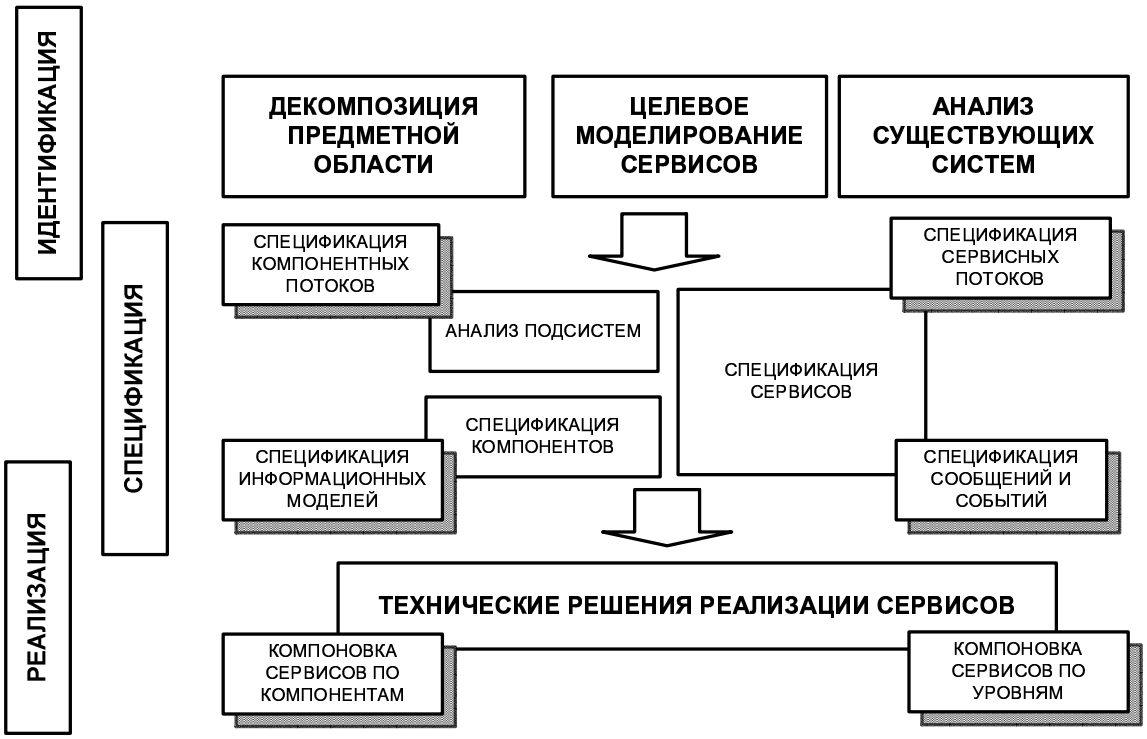


*Рис. 3.3*

Привычное представление о такой простой структуре стандартов, как стек, в случае СОА явно не подходит. Для установления связей между компонентами и сборки единой системы из этих слабосвязанных и асинхронных компонентов требуются совершенно иные приемы средства – оркестровка и хореография.

### *3.2.3. Вертикальная архитектура СОА-приложений*

В ряде случаев СОА удобно рассматривать как многоуровневую архитектуру композитных (составных) сервисов, согласованных с бизнес-процессами. На рис. 3.4 приведено укрупненное представление уровней организации СОА.

Отношения между сервисами и компонентами заключаются в том, что компоненты уровня предприятия реализуют сервисы и несут ответственность за обеспечение их функциональности и качества обслуживания. Потоки бизнес-процессов могут опираться на хореографию раскрытых сервисов в композитные приложения. Выделяют следующие уровни организации СОА-систем:

- операционной системы;
- корпоративных компонентов;

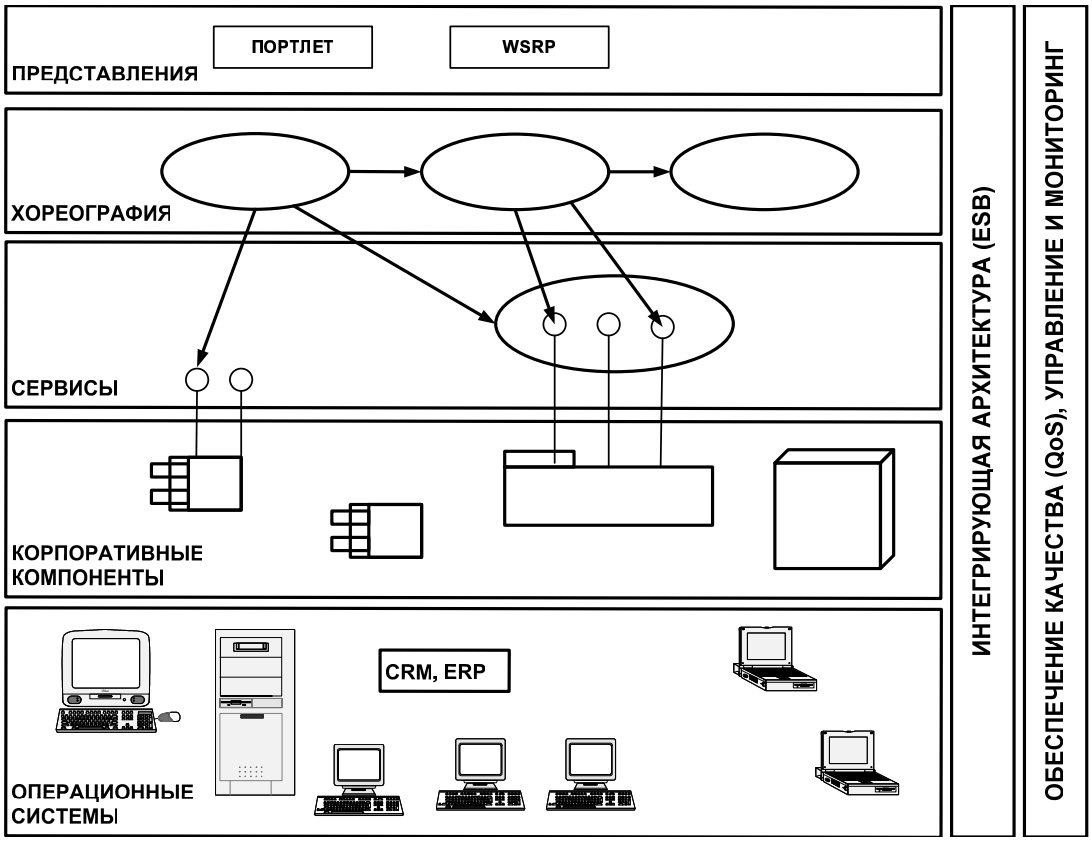


*Рис. 3.4*

- сервисов;
- хореографии;
- презентации;
- интеграции (ESB);
- качества обслуживания (QoS);

Архитектура интеграции поддерживает маршрутизацию, посредничество и трансляцию сервисов, компонентов и потоков с использованием корпоративной сервисной шины (Enterprise Service Bus, ESB). Развернутые сервисы должны контролироваться и управляться для обеспечения качества сервиса и строгого соблюдения нефункциональных требований.

**Уровень операционной системы.** Состоит из существующих заказных приложений, называемых также *унаследованными* системами. Включает в себя существующие упакованные приложения системы управления взаимосвязями (Customer Relationship Management, CRM) и планирования ресурсов предприятия (Enterprise Resource Planning, ERP), а также ранние объектно-ориентированные реализации системы и приложения для управления бизнесом. Многоуровневая СОА может помочь улучшить уже существу-

ющие системы и способствовать их интеграции с использованием сервис-ориентированных методов.

**Уровень корпоративных компонентов.** Корпоративные компоненты несут ответственность за обеспечение функциональности и поддержание качества обслуживания (QoS) сервисов. Эти компоненты являются управляемым, регулируемым набором корпоративных средств, которые расположены на уровне корпоративной или бизнес-единицы. Как средства корпоративного масштаба они несут ответственность за обеспечение соответствия соглашениям об уровне услуг (Service Level Agreement, SLA) путем применения лучших методов проектирования. Для реализации компонентов, для управления рабочей нагрузкой, отказоустойчивостью и для балансирования нагрузки данный уровень обычно использует такие технологии, основанные на использовании контейнеров, как серверы приложений.

**Уровень сервисов.** В этом уровне находятся сервисы, отобранные бизнесом. Они могут быть выявленными или же статически связанными и впоследствии собранными в композитный (составной) сервис. Данный уровень раскрытия сервисов обеспечивает механизм для принятия компонентов масштаба предприятия, специальных бизнес-компонентов, а в некоторых случаях – и компонентов для определенного проекта. Кроме того, он выводит подмножество их интерфейсов в форму описания сервисов. Таким образом, корпоративные компоненты обеспечивают реализацию сервиса в период работы, используя функциональность, предоставленную их интерфейсами. В данном уровне интерфейсы экспортируются как описания сервиса, в котором они раскрыты для использования. Они могут существовать отдельно или как композитный (составной) сервис.

**Уровень объединения (хореографии) бизнес-процессов.** В этом уровне определяются способы объединения сервисов, находящихся на соответствующем уровне. Сервисы связаны в поток путем группировки (хореографии) и, следовательно, они действуют совместно как отдельное приложение. Подобные приложения поддерживают особые ситуации и бизнес-процессы. Здесь для проектирования потоков приложения могут использоваться такие визуальные инструменты компоновки, как IBM WebSphere Business Integration Modeler или Websphere Application Developer Integration Edition. Эти приложения будут рассмотрены далее.

**Уровень доступа (презентации).** Несмотря на то, что этот уровень опри обсуждении СОА бычно упускается, он постепенно становится все более зна-

чимым. Здесь он обсуждается по причине возрастающей конвергенции стандартов (таких как, Web Services for Remote Portlets Version) и других технологий, стремящихся вывести web-сервисы на уровень интерфейса приложения или презентации. Этот уровень можно представить как уровень, который необходимо принять во внимание в последующих разработках. Важно также иметь в виду, что СОА отделяет пользовательский интерфейс от компонентов, и в качестве альтернативы может потребоваться обеспечение сквозного решения между каналом доступа и сервисом или набором сервисов.

**Интеграция (ESB).** Этот уровень допускает интеграцию сервисов предоставлением проверенного набора таких возможностей, как интеллектуальная маршрутизация, посредничество протоколов, и других механизмов преобразований, обычно описанных как ESB. Язык описания web-сервисов определяет связь, которая заключает в себе местонахождение предоставляемого сервиса. С другой стороны, ESB обеспечивает механизм интеграции, независимый от местонахождения.

**Качество обслуживания (QoS).** Этот уровень предоставляет возможности для мониторинга и поддержки таких аспектов качества обслуживания, как обеспечение безопасности, производительности и доступности, и управления ими. Он является фоновым процессом, использующим механизмы запросов и ответов, а также инструменты, контролирующие общее состояние приложений СОА.

Сюда включены все важные стандартные реализации систем управления службами и других значимых протоколов и стандартов, реализующих качество обслуживания для СОА.

## 3.3. Сервис-ориентированное моделирование и анализ

Процесс сервис-ориентированного моделирования и построения архитектуры состоит из трех основных шагов: идентификации, спецификации и реализации сервисов, компонентов и потоков (обычно путем объединения сервисов).

### *3.3.1. Идентификация сервиса*

Для идентификации сервисов используют следующие методы декомпозиции – нисходящую, восходящую и исходящую декомпозиции. При этом решаются задачи сегментации сферы влияния, анализа существующих средств, моделирования задач и средств для ее решения, а также моделирования сервисов.

При *нисходящем* анализе проект возможных случаев использования бизнеса обусловливает спецификацию для бизнес-сервисов. Такой нисходящий процесс часто называют декомпозицией домена. Он заключается в декомпозиции сферы влияния (домена) бизнеса на его функциональные области и подсистемы включая потоки или декомпозицию процесса в процессы, подпроцессы и высокоуровневые случаи использования бизнеса.

Эти случаи использования часто являются хорошими кандидатами для раскрытия бизнес-сервисов на стороне предприятия или для использования последних в пределах предприятия во всей бизнес-стратегии.

При анализе существующей системы анализируется унаследованная система. При этом выбираются подходящие кандидаты для обеспечения менее затратных решений для реализации функциональности сервиса, лежащей в основе бизнес-процесса. Здесь анализируются и используются для достижения цели различные API, транзакции и модули от унаследованных и упакованных приложений. В некоторых случаях с целью поддержки функциональности сервиса необходимо разбиение на копоненты унаследованных систем для переработки существующих средств.

*Исходящий* подход заключается в моделировании типа "задача-сервис" для подтверждения и извлечения сервисов, не найденных при проведении восходящей и нисходящей идентификации. Он разбивает сервисы на задачи и подзадачи, ключевые показатели производительности и исходные параметры.

### *3.3.2. Классификация сервиса*

Классификация сервиса осуществляется при идентификации сервисов. Очень важно начать классификацию сервиса в иерархии сервисов, отражая композитную или фрактальную их природу: сервисы могут и должны быть скомпонованы из более тонкоструктурных компонентов и сервисов. Классификация помогает определить композицию и иерархическое представление, а также скоординировать построение взаимозависимых сервисов и их иерархии. Кроме этого, она помогает смягчить синдром роста сервисов, при котором увеличивающееся количество тонкоструктурных сервисов определяется, проектируется и размещается с небольшим управлением.

### *3.3.3. Анализ подсистемы*

Для анализа подсистемы берутся подсистемы, найденные в результате декомпозиции домена, и определяются взаимозависимости и потоки между ними. Кроме того выявляются случаи использования, идентифицированные

во время декомпозиции домена как раскрытые сервисы в интерфейсе подсистемы. Анализ подсистемы заключается в создании моделей объекта для представления внутренних выработок и конструкций подсистем, раскрывающих и анализирующих сервисы. Проектная конструкция подсистемы впоследствии реализуется, как конструкция реализации крупноструктурного компонента, реализующего сервисы в данном мероприятии.

### *3.3.4. Спецификация компонента*

При спецификации компонента определяются следующие детали компонента, реализующего сервисы:

- данные;
- правила;
- сервисы;
- настраиваемые профили;
- вариации (возможно, полиморфизм).

Здесь также задаются спецификации обмена сообщениями, спецификации событий и дается определение управления.

### *3.3.5. Размещение сервиса*

Размещение сервиса заключается в распределении сервисов по подсистемам, которые уже были идентифицированы. В этих подсистемах присутствуют корпоративные компоненты, реализующие их опубликованную функциональность. Можно представить, что подсистема обладает однозначным соответствием корпоративным компонентам. Структурирующие компоненты появляются при использовании шаблонов для создания корпоративных компонентов в комбинации:

- с посредниками (медиаторами);
- моделями представления (фасадами);
- правилами;
- настраиваемыми профилями;
- реестрами служб.

Размещение сервиса заключается также в распределении сервисов и реализующих их компонентов по соответствующим уровням SOA. Такое размещение является ключевой задачей, требующей документирования и анализа основных архитектурных решений, относящихся не только к архитектуре приложения, но и к операционной архитектуре, разработанной и используемой для поддержки реализации SOA в процессе работы.

### *3.3.6. Реализация сервиса*

На этапе реализации сервиса выявляется, что программное обеспечение, реализующее данный сервис, должно быть выбрано или создано на заказ. Другие доступные альтернативы включают в себя интеграцию, преобразование, подписку и привлечение внешних ресурсов функциональных частей с использованием web-сервисов. Здесь принимается решение, какой модуль унаследованной системы будет использоваться для реализации данного сервиса, а какой – будет построен заново, "с нуля". Кроме этого, существуют решения реализации и для других сервисов, не включающих бизнес-функциональность, а именно – обеспечение безопасности, управление сервисами и их контроль.

Нисходящая декомпозиция области, какой являются моделирование процесса и декомпозиция, вариантно-ориентированный анализ, анализ политик и бизнес-правил проекта, а также моделирование поведения области (с использованием грамматик и диаграмм), проводятся параллельно с восходящим анализом существующих унаследованных средств, которые являются кандидатами для разбиения на модули и раскрытия сервиса. Чтобы понять задачи бизнес-проекта и настроить сервисы в соответствии с этими задачами, используют моделирование сервисов.

## 3.4. Уровни адаптации СОА

Переход к СОА – сложный процесс, который связан не только с серьезными трансформациями ИТ-инфраструктуры, но и с изменениями во взаимосвязях между бизнес-процессами и ИТ. Для выполнения такого перехода IBM предлагает модель уровней адаптации принципов СОА. Ниже приводятся черты каждого уровня адаптации и предполагаемые инструментальные средства, которые можно использовать [16].

**Уровень 1. Реализация отдельных web-сервисов.** Это начальный уровень развертывания СОА, на котором технологии web-сервисов используются для разработки новых приложений или преобразования существующих, например, для интеграции с помощью WSDL-интерфейсов систем, написанных на C++, Cobol и Java. Здесь компании должны реализовать этапы создания и развертывания сервисов. Для создания предлагается инструментарий WebSphere Studio Application Developer, а также набор средств Emerging Technology Toolkit, который позволяет разработчикам опробовать новые решения в области web-служб. Развертывание web-сервисов поддерживается, например сервером приложений WebSphere Application Server.

**Уровень 2. Сервисно-ориентированная интеграция бизнес-функций.** На этом уровне уже достигнуто преобразование приложений в сервисы и ставится вопрос об интеграции их таким образом, чтобы реализовать определенную бизнес-задачу. Одно из основных преимуществ СОА состоит в том, что эта архитектура, в отличие от многих традиционных программных моделей, нацелена на поддержку не программы, а процесса. В программе, написанной исходя из представлений программиста об оптимальности, логика процесса могла быть произвольным образом распределена между компонентами (например, для того чтобы добиться многократного использования нужных компонентов, программисты прибегают к самым разным приемам – копированию кода, использованию разделяемых библиотек, наследованию объектов и т. д.).

В СОА приложение разрабатывается исходя из логики бизнес-процесса. Процесс разбивается на некоторую последовательность шагов, каждый из которых реализуется как сервисный компонент приложения. Эти компоненты интегрируются таким образом, чтобы их выполнение в определенной последовательности приводило к нужному бизнес-результату.

Когда речь идет об интеграции, подразумевается взаимодействие между сервисами в СОА на уровне интерфейсов. Однако надо иметь в виду, что в реальной ИТ-инфраструктуре, где будет происходить переориентация на сервисы, проблема интеграции может оказаться гораздо шире и необходимо будет учитывать ее различные типы и стили. Далеесы приведены некоторые из них:

- *Интеграция на уровне пользовательского интерфейса.* Получение удобного и эффективного интерфейса для взаимодействия пользователя со средой интегрированных сервисов. Эта область интеграции связана с развитием портальных технологий.
- *Информационная интеграция*. Обеспечение согласованного доступа к данным без каких-либо ограничений, связанных с форматом, с логическим и с физическим размещениями данных.
- *Поддержка различных способов коммуникаций низкого уровня между приложениями.* Речь идет о таких механизмах, как синхронные и асинхронные коммуникации, маршрутизация, трансформация и высокоскоростное распределение данных, шлюзы и конвертеры протоколов, виртуализация ввода–вывода и т. д.
- *Интеграция процессов.* Поддержка нужной последовательности сервисов для реализации бизнес-процесса, интеграция процессов с другими процессами.

- *Интеграция унаследованных систем.* Создание информационной среды для интеграции старых приложений, которые невозможно изъять из технологического процесса.

Существует еще одна архитектурная концепция, используемая для сервисно-ориентированной интеграции. Это – сервисная шина предприятия, или *корпоративная сервисная шина* (Enterprise service bus, ESB). Ее задача – предоставить единый механизм передачи запросов и получения результатов сервисов, выполнения необходимых преобразований сообщений и транспортных протоколов, обеспечения требований безопасности доступа и, что наиболее важно, управления потоком обращений к сервисам. Благодаря такому управлению выполняется нужная последовательность вызовов сервиса для реализации бизнес-процесса. Определение процесса как серии обращений к сервисам поддерживается, например в разработанном усилиями IBM и "Microcoft" языке Business Process Execution Language (BPEL).

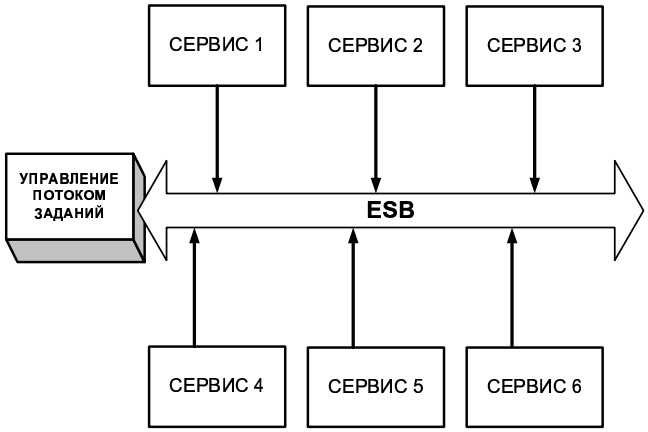


*Рис. 3.5*

Схема ESB приведена на рис. 3.5. Из него можно видеть, что такой подход решает одну из главных проблем интеграции – проблему минимизации интерфейсов. Добавление нового сервиса к общей картине приведет к появлению одного и только одного дополнительного интерфейса для интеграции с остальными компонентами архитектуры.

Все задачи интеграции и отображения бизнес-процессов компании в сервисы являются предметом реализации на втором и на третьем уровнях перехода к СОА в трактовке IBM. На этих уровнях вступают в действие все четыре этапа жизненного цикла сервисов и используется множество программных продуктов. Данный уровень – это реализация СОА для ограниченного числа подразделений в компании. На этапе интеграции используются такие инструменты, как WebSphere Host Access Transformation Services.

Для развертывания используется поддерживающий язык BPEL сервер интеграции бизнес-процессов WebSphere Business Integration Server Foundation и шлюзы CICS Tranaction Gateway или IMS Connect. Для использования по-

лученных возможностей можно воспользоваться WebSphere Portal, а функции управления возлагаются на модули семейства Tivoli – Access Manager и Monitoring for Transaction Performance.

**Уровень 3. Трансформация ИТ-инфраструктуры в корпоративном масштабе.** На данном уровне выполняется сервисно-ориентированная интеграция приложений и процессов уже в масштабах всей компании. Согласованный сервисный подход к ИТ-инфраструктуре распространяется не только на внутренние подразделения, но и на партнеров и поставщиков.

Для трансформации IT-инфраструктуры корпоративного масштаба находят применение системы, обеспечивающие более глубокую детализацию разработки и интеграцию сервисов с учетом всех уже рассмотренных типов интеграции.

Инструментарий уровня таков: WebSphere Business Integration Modeler и Rational Rose XDE для этапа создания сервисов, WebSphere Business Integration Message Broker для развертывания, DB2 Information Integrator и Lotus Workplace для стадии использования.

Управление полноценной средой COA реализуется с помощью инструментов семейства Tivoli – Identity Manager, Business System Manager и Monitoring for Business Integration, а также WebSphere Business Integration Monitor.

По статистике [16] (до 2003 г. включительно) большинство организаций, проявивших практический интерес к технологиям web-сервисов, тратили свои средства и усилия на разработку отдельных сервисов и изучение возможностей их использования в корпоративной инфраструктуре. Следующими этапами являются интеграция сервисов в единую среду, решение задач управления ею и обеспечение ее безопасности. Таким образом, второй и третий уровни адаптации COA имееют вполне практический смысл.

**Уровень 4. Оптимизация процесса и бизнеса.** Последний уровень связан с изменениями в самих способах ведения бизнеса в ответ на глобальные трансформации ИТ-инфраструктуры. Существует сильная связь между COA и стратегией on-demand computing, которой подчинены стратегии развития программных и аппаратных решений крупных вендоров.

Данная стратегия подразумевает использование моделей зрелости процессов, таких, как CMM, которые, в свою очередь, являются основополагающими при организации процессного подхода к организации бизнеса. В указанных моделях последней, наивысшей стадией развития процесса является

уровень самоорганизации и самооптимизации. Предполагается, что при достижении последнего уровня адаптации СОА предприятие способно начать оптимизационный процесс, опираясь на созданную информационную инфраструктуру.

## 3.5. Контрольные вопросы

1. В чем выражается процессный подход при описании бизнес-деятельности?
2. Какие стандарты определяют терминологию процессного подхода?
3. Что такое свойство фрактальности?
4. Какими свойствами обладают процессы?
5. Какие роли подразумевает процессное описание деятельности?
6. Приведите основные черты OBM.
7. В чем отличие OBM от универсальных процессных моделей?
8. Какие недостатки модели Портера вам известны?
9. Дайте определение сервис-ориентированной архитектуры.
10. Какие предпосылки обусловили развитие СОА?
11. Как понятие СОА согласуется с процессным подходом?
12. Какие сервисно-ориентированные платформы вам известны?
13. Опишите базовую модель взаимодействия СОА.
14. Приведите формальную модель взаимодействия компонентов СОА на языке UML.
15. Что такое композитное приложение?
16. Что такое реестр сервисов, и в чем его назначение?
17. Что такое оркестровка сервисов?
18. Что такое хореография сервисов?
19. Какие языки оркестровки вам известны?
20. Каким требованиям должны удовлетворять сервисы для возможности их оркестровки?
21. Какие уровни организации СОА вам известны? Опишите каждый из них.
22. Приведите список уровней адаптации СОА.
23. Перечислите задачи, решаемые на уровне трансформации IT-инфраструктуры.

# 4. ИНТЕГРАЦИЯ СЕРВИСОВ И ПРИЛОЖЕНИЙ

Интеграция корпоративной информации в рамках компьютерных информационных систем исследуется довольно давно. Ряд достижений в этой области стал основой для изменения подходов к построению программных систем в целом. Далее рассматриваются современные подходы к интеграции корпоративных данных в аспекте сервисного подхода и сервис-ориентированной архитектуры.

## 4.1. История подходов к интеграции

Как правило, на этапе осознания необходимости объединения приложений и данных, разрозненных по подразделениям предприятия, становится очевидной необходимость методического подхода к разработке методов инте-

грации. Рассмотрим генезис подходов к интеграции приложений и данных. На его примере можно проследить развитие подходов к программной интеграции.

### *4.1.1. Ранние подходы*

Под *интеграцией приложений* понимается процесс налаживания взаимосвязей между независимыми технологическими решениями, существующими внутри организации. Поскольку изначально не предполагалось, что приложения должны будут работать друг с другом, это не так-то просто. Поиск решения этой проблемы продолжается уже давно.

Первоначально идея интеграции развивалась по двум направлениям [17].

**Частная интеграция.** Это традиционный и наиболее распространенный в прошлом подход к интеграции систем, заключающийся в создании специализированных интерфейсов к приложениям для каждой отдельной ситуации. Проблема заключалась в том, что в результате частной интеграции возникала путаница из связующих звеньев интеграции приложений, что, в конечном счете, заводило ее в тупик.

По мере интеграции новых систем количество необходимых частных интерфейсов увеличивается пропорционально квадрату числа использующихся приложений. С увеличением числа новых интерфейсов осуществление интеграции для создания новых приложений становится сложнее. Частная интеграция требует больших объемов программирования, отнимает много времени, приводит к огромным структурным сложностям, а проведение подобных изменений финансово невыгодно. Фактически, частная интеграция приводит к созданию еще одной независимой информационной системы и увеличивает стоимость внедрения каждого нового приложения.

Проблема указанного подхода к интеграции усугубляется тем, что многие традиционные критичные приложения слишком хрупки, чтобы их модифицировать.

**Интеграция на уровне пользовательских интерфейсов.** Интеграция на уровне пользовательских интерфейсов заключается в том, что приложения могут взаимодействовать так же, как их используют люди, а именно, через пользовательский интерфейс. Простейшая форма такой интеграции под названием screen scraping уже более десяти лет используется для расширения возможностей мэйнфрейм-приложений.

Как правило, технологии интеграции на уровне пользовательских интерфейсов используют некоторые ограничения в презентационном слое для связывания систем. Более новый вариант интеграции на уровне пользователь-

ских интерфейсов – HTML-scraping – основан на результатах работы web-приложений, выведенных в браузере.

Инструменты этого класса, такие, как Composite Application Platform компании "CrossWeave", идентифицируют компоненты HTML-документа, полученного в результате работы web-приложения, и предоставляют эти компоненты для повторного использования и интеграции.

Такой подход предлагает доступ к интегрированным системам "только для чтения". Кроме того, этот подход сильно ограничен, поскольку для не web-приложений имитация пользователя в качестве парадигмы интеграции – сложная задача. Для этого требуются программы, способные воспринимать вызовы GUI, соотнося их с объектами приложений и предоставляя интерфейсы в качестве программируемых средств управления в оперативном режиме.

Хотя ранние подходы решают непосредственные бизнес-проблемы, они отнюдь не всегда оптимальны и часто приводят к созданию негибких независимых приложений, в результате чего увеличивается стоимость и время разработки будущих интеграционных модулей. В лучшем случае ранние подходы представляют собой промежуточный мост для перехода к более гибким, масштабируемым и надежным решениям по интеграции приложений.

### *4.1.2. Интеграция на уровне данных*

Базовым подходом на уровне интеграции данных явился подход на основе *хранилищ данных* (data warehouses). Он подразумевает поддержку данных в специальных хранилищах независимо от бизнес-логики, их породившей. Доступ к хранилищам могут получать различные приложения. Например, база данных по сотрудникам может служить общим фундаментом для нескольких HR-приложений (от *англ*. Human Resources), таких, как распределение премий и пособий, планировщик направления сотрудников на тренинги и отслеживание эффективности работы сотрудника.

Интеграция приложений по такой схеме подразумевает составление запросов и проведение обновлений по общей хорошо документированной модели данных. Во многом благодаря успеху продуктов, созданных на основе реляционных баз данных, и сопутствующих стандартов (таких, как SQL и ODBC) интеграция на уровне данных продолжает господствовать в качестве способа оптимизации взаимосвязей между различными системами.

Стоит отметить, что данные в хранилищах могут вызвать некоторые проблемы. Во-первых, они подразумевают использование общей модели данных для всего набора приложений, что ограничивает гибкость ориентированных

на них приложений и замедляет разработку новых приложений. Даже в четко определенной области, например в HR, приложения самообслуживания, предназначенные для сотрудников, и инструменты поддержки принятия решений, нацеленные на менеджеров, скорее всего будут использовать различные модели данных. Кроме того, интеграция на уровне данных сама по себе не предлагает средств для связи с традиционными приложениями. Иногда слои данных таких систем хорошо представлены и документированы, однако чаще для связи с ними требуются специальные коннекторы, что влечет за собой дополнительные расходы, организацию внесения изменений и "привязку" к поставщику. Кроме того, компоненты приложений не всегда хорошо взаимодействуют в реальном времени, что является критически важным требованием современных бизнес-приложений.

Существует набор решений, позволяющий разрешить эти и другие проблемы интеграции данных. Например, хранилища метаданных могут уменьшить потребность в общей модели данных с помощью использования абстрактного слоя поверх отдельных баз данных.

### *4.1.3. Интеграция на уровне корпоративных приложений*

В достаточно детерминированной информационной корпоративной среде, когда интерфейсы приложений достаточно хорошо формализованы, эффективен подход интеграции на уровне приложений (Enterprise Application Integration, EAI). Он подразумевает совместное использование исполняемого кода, а не внутренних данных.

EAI устраняет необходимость в выполнении больших объемов программирования. Вместо этого подход интеграции на уровне приложений разбивает программы на компоненты и интегрирует их с помощью стандартизированных программных интерфейсов, распределенных объектных технологий и связующего ПО. В состав средств EAI входят традиционные библиотеки функций и API, созданные для объединения программ на одной платформе, а также межплатформные инструменты, такие, как RPC (Remote Procedure Call – удаленный вызов процедуры) и ORB (Object Request Brokers – брокер объектных запросов).

EAI осуществляет интеграцию на функциональном уровне, а не только на уровне пользовательских интерфейсов или данных. В результате новые интегрированные приложения могут осуществлять функции как "считывания", так и "записи", т. е. поддерживать полноценные транзакции со всеми интегрированными системами.

Вместо специализированных интерфейсов между отдельными программами EAI полагается на связующую среду с возможностью многократного использования, которая играет роль универсального программного ядра, соединяющего все приложения. Эта интеграционная структура может многократно использоваться для быстрого создания новых приложений. Таким образом, интеграторы приложений должны создать только один интерфейс для связи со структурой интеграции приложений, вместо того чтобы создавать все новые интерфейсы для связи с каждым новым приложением. Полученную в результате систему легче поддерживать и расширять. Повторное использование функций в рамках имеющейся среды позволяет значительно снизить время и стоимость разработки приложений, при этом скудные ресурсы развития можно направить на важные бизнес-задачи, а не воссоздавать заново уже существующие функции. Интеграционные среды такого рода получили название "middleware" (*промежуточное ПО*, или *инфраструктурное ПО*). С учетом того, что бизнес-логика, как правило, расположена на средних ярусах распределенной архитектуры, термин "middleware" хорошо описывает суть процессов.

## 4.2. Web-Услуги

Подход с применением web-услуг основан на использовании объектно-ориентированных технологий для заключения данных и программных элементов в web-"оболочку" таким образом, чтобы к ним могли получать доступ различные web-приложения или клиенты (например, используя SOAP, браузер может сравнить цены на нескольких сайтах и предоставить клиенту сравнительный отчет).

Повсеместное использование Интернет-стандартов, наличие стандартной индустриальной платформы взаимодействия J2EE и быстрое распространение XML в качестве независимого от платформы формата передачи сообщений способствуют активному продвижению такого подхода.

Java-программирование также способствует интеграции процессного уровня, которая напрямую поддерживается многими сервисами J2EE (Java 2 Enterprise Edition) включая RMI (Remote Method Invocation), JMS (Java Messaging Service) и JCA (Java Connector Architecture). Перечисленные сервисы в настоящее время являются промышленной основой для создания распределенных Интернет-приложений. Кроме того, они составляют промышленную основу для построения распределенных корпоративных приложений.

В рамках СОА наиболее активно применяются следующие технологии интеграции:

- интеграция корпоративных приложений (Enterprise Application Integration, EAI);
- интеграция корпоративной информации (Enterprise Information Integration, EII);
- программное обеспечение для извлечения, преобразования и загрузки данных (Extract, Transform and Load, ETL).

Эти технологии могут быть использованы для широкого круга задач: от интеграции в режиме реального времени до пакетной интеграции и от интеграции данных до интеграции приложений. На рис. 4.1 показано положение названных технологий по отношению к этим двум спектрам задач. Для интеграции данных в режиме реального времени лучше всего подходит технология EII; для пакетной интеграции данных – ETL, а для интеграции приложений в режиме реального времени или в пакетном режиме наиболее подходящим инструментом является технология EAI.

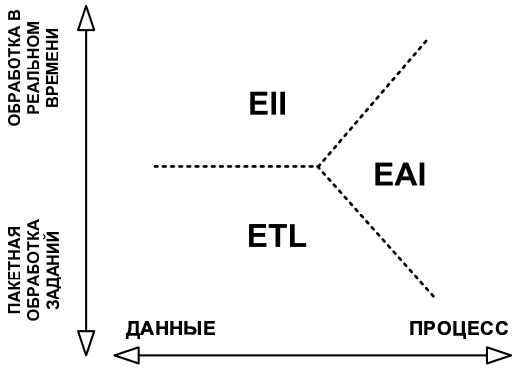


*Рис. 4.1*

Предназначение и область применимости каждой технологии можно определить следующим образом:

*EAI* – это технология, с помощью которой организация добивается централизации и оптимизации интеграции корпоративных приложений, обычно используя те или иные формы технологии оперативной доставки информации (push technology), которая управляется внешними событиями (event-driven);

*ETL* – это технология, которая преобразует данные (обычно с помощью их пакетной обработки) из операционной среды, включающей гетерогенные технологии, в интегрированные, согласующиеся между собой данные, пригодные для использования в процессе поддержки принятия решений. Технология ETL ориентирована на базы данных;

*EII* – это технология для интеграции в режиме реального времени несопоставимых типов данных из многочисленных источников как внутри, так и за пределами корпорации. Инструменты EII обеспечивают универсальный уровень доступа к данным и используют технологию поиска информации (pull

technology) или возможности работы по запросам. Технология EII ориентирована на конкретных сотрудников, которые получают информацию через инструментальную панель или отчет.

Технология EAI наиболее функциональна тогда, когда необходимо связать приложения в реальном времени для автоматизации бизнес-процессов. Другой случай применения EAI – это ситуация, когда необходимо, чтобы изменения, внесенные в одно приложение (обычно, небольшой набор записей), были отражены во всех других. Данная технология очень хорошо справляется с задачей фиксации изменений и их переноса в соответствующие приложения или системы.

Технология ETL оказывается наиболее полезной в тех случаях, когда необходимо создать хранилище данных, содержащее хорошо документированные и надежные данные для исторического анализа, например, для анализа временны́х рядов или многомерных запросов. Эта технология также используется для интеграции ключевых справочных данных. Технология ETL незаменима для таких задач, как удаление дублирующихся данных, осуществление процессов проверки качества данных и т. п. Эти инструменты используются и для создания отдельных витрин данных, обслуживающих конкретный отдел или бизнес-процесс или предназначенных для каких-либо долгосрочных целей. Инструменты ETL дают пользователю возможность запустить повторяющиеся процессы для большей слаженности действий и для их многократного использования. Такие процессы включают создание точных технических метаданных, поддерживающих общую целостность бизнес-среды. Совокупность перечисленных средств принято называть Business Intelligence (BI).

Технология EII лучше всего подходит в тех случаях, когда необходимо создать общий шлюз (gateway) с едиными языком и точкой доступа к несогласованным источникам данных. Данные инструменты предоставляют приложениям и конечным пользователям возможности более гибкого, а также незапланированного доступа к данным, при этом не требуя постоянного использования данных или долговременных целей для получения этого доступа. Наряду с традиционными реляционными базами данных инструменты EII могут работать с XML- и LDAP-файлами, плоскими файлами и другими нереляционными данными. Эти инструменты также способны представлять реляционные данные в формате XML или формате web-сервисов. Особенно полезны инструменты EII, если есть необходимость добавить к справочным данным хранилища дополнительные детали, в частности детальную инфор-

мацию в реальном времени (например, сопоставление исторических данных с текущей ситуацией).

Следует отметить, что внедрение этих технологий требует от IT-персонала глубокого понимания тех требований, которые предъявляются к данным для принятия как тактических, так и стратегических решений. Применительно к технологии ETL это означает, что необходимые данные извлекаются, преобразуются и загружаются в виде, пригодном для использования непосредственно аналитиками или EII-сервером. В случае EII-технологии способы представления данных должны удовлетворять отчетным требованиям аналитиков, т. е. данные должны быть пригодны для использования в аналитических отчетах. Во всех случаях понимание того, что является источниками данных и требований, предъявляемых к данным, является необходимым шагом при внедрении этих технологий и безусловно оправдывает то время, которое приходится тратить, чтобы достичь такого понимания.

Внедрение названных инструментов в уже сложившуюся архитектуру требует от бизнес- и IT-персонала разработки указанной стратегии управления данными и приложениями, которая будет постоянно поддерживать этот процесс в активном состоянии. Обязательной составляющей такой стратегии должно быть осознание того, что в этой связи повышается важность механизмов архивирования, а также того, что с самого начала должны быть созданы контрольные журналы. Это необходимо для обеспечения слаженности и надежности интегрированных данных и приложений.

Важен и постоянный мониторинг производительности и эффективности данных технологий в условиях конкретной инфраструктуры. Их производительность в значительной степени будет зависеть от скорости архивирования данных, их размеров и детальности, а также от эффективности функционирования системы в условиях полной нагрузки. При определении производительности также следует оценить влияние, которые эти инструменты могут оказывать на операционные приложения и системы. Вот почему необходим постоянный мониторинг и этого влияния.

## 4.3. Контрольные вопросы

1. Какие подходы к интеграции приложений вам известны?
2. В чем заключается частная интеграция?
3. В чем суть интеграции на уровне пользовательских интерфейсов? В каких условиях она оправданна?
4. Что такое интеграция на уровне данных?
5. Что такое интеграция на уровне корпоративных приложений?

# 5. СТАНДАРТЫ СОА

Одним из факторов, обеспечивающих активное развитие СОА, является активная деятельность специальных групп и комитетов по разработке стандартов, формализующих протоколы взаимодействия и интерфейсы, необходимые для обеспечения интероперабельности, безопасности, мобильности (возможности переноса) и динамической интеграции информационных систем. По мнению ведущей по внедрению сервис-ориентированных технологий "UnitSpace", в стандартизации в первую очередь нуждаются следующие ключевые аспекты ведения бизнеса:

- описание организаций и их сервисов;
- обнаружение и потребление сервисов;
- обмен документами и сообщениями между различными приложениями электронного бизнеса;
- универсальная интерпретация документов и сообщений;
- обеспечение конфиденциальности информации и взаимодействий, безопасности информационных систем.

Среди основных направлений работы в области развития стандартов для электронного бизнеса следует выделить ряд магистральных направлений:

- Создание на базе Web services и XML технологии UDDI (Universal Description, Discovery and Integration). UDDI обеспечивает создание реестров организаций и сервисов, которые являются интегральными компонентами сервис-ориентированной архитектуры любой информационной среды. Такой средой может быть информационная инфраструктура отдельно взятой организации или сообщества организаций, объединенных по отраслевому, региональному либо иному признакам.
- Создание на базе технологии UDDI глобального регистра UDDI (Business Registry, UBR). UDDI Business Registry является глобальным общедоступным реестром организаций и их сервисов. Он рассматривается в качестве ключевого компонента единой информационной среды электронного бизнеса, обеспечивающего доступ к бизнес-информации, поиску партнеров и клиентов, взаимодействие и интеграцию с ними.
- Разработка спецификаций электронного бизнеса ebXML. Группа стандартов ebXML строится на основе модульной архитектуры и с учетом мирового опыта применения стандартов EDI и web-services. ebXML обеспечивает ряд компонентов технической инфраструктуры, непосредственно нацеленной на осуществление коммерческих операций и синхронизацию

бизнес-процессов в рамках межорганизационного взаимодействия. Данное направление является ключевым.

- Разработка и продвижение современных стандартов электронного правительства (e-Government) для обеспечения взаимодействия государственных структур друг с другом, с коммерческими компаниями, с общественными организациями и с рядовыми гражданами.

Следует отметить организации, наиболее активно участвующие с выработке стандартов электронного бизнеса: организация по продвижению стандартов структурированной информации (Organization for the Advancement of Structured Information Standards, OASIS) и центр ООН по содействию торговле и электронному бизнесу (группы UN/CEFACT и UN/EDIFACT), который вовлечен в развитие ebXML совместно с OASIS. OASIS является правопреемником отраслевого консорциума UDDI, в рамках которого "UnitSpace" участвовал в создании первых версий спецификаций UDDI, продолжая работу по развитию стандартов реестра электронного бизнеса. А созданный при OASIS технический комитет e-Government ставит перед собой задачи координации инициатив национальных органов власти для более эффективного применения современных информационнных технологий.

## 5.1. Классификация соглашений и стандартов

Первыми и наиболее очевидными соглашениями, позволяющими реализовать web-сервисы, стали стандарты SOAP, WSDL и UDDI. Фактически, это не стандарты, а три предложения от группы крупных производителей, заинтересованных в развитии SOA. Существует мнение, что web-сервисы – это только то, что реализуется стеком из SOAP, WSDL и UDDI. Имеется альтернативный подход, называемый REST [18], который позволяет реализовать функции web-служб без привлечения этих стандартов.

В таблице приведена классификация сервисов и поддерживающих их соглашений и стандартов.

| Характеристика | Java RMI | CORBA | DCE | Web-Сервисы |
|---|---|---|---|---|
| Механизм вызова | Java RMI | CORBA RMI | RPC | JAX-RPC, .Net |
| Формат данных | Serialized Java | CDR | NDR | XML |
| Формат обмена | Stream | GIOP | PDU | SOAP |
| Протокол передачи | JRMP | IIOP | RPC CO | HTML, SMTP |
| Описание интерфейса | Java Interface | CORBA IDL | DCE IDL | WSDL |
| Механизм обнаружения | Java Registry | COS naming | CDS | UDDI |

Над вопросами стандартизации сервисных архитектур работают две группы – в рамках комитета W3C и в составе OASIS. Работа первой груп-

пы сосредоточена на основных спецификациях сервисной инфраструктуры, а второй – на расширении ее функциональности.

В рамках W3C работы по стандартизации web-сервисов начались в 2000 г., когда была создана рабочая группа XML Protocol Working Group (XMLP), а затем сложилось объединение Web Services Activity, состоящее из трех рабочих групп и одной координирующей. Деятельность Web Services Architecture Working Group (WSAWG) сосредоточена на развитии стандартов для трех компонентов сервисов (включая транспорт, описание и обнаружение), максимально близких к SOAP, WSDL и UDDI. Работа групп WS-Desc и XMLP непосредственно связана с разработкой XML-спецификаций для описания web-сервисов.

OASIS имеет в своем составе более 30 технических комитетов, деятельность которых связана архитектурой web-сервисов; в большинстве своем они были созданы после 2000 г. Web Security Technical Committee разрабатывает стандарты, обеспечивающие безопасную передачу данных средствами, построенными на основе SOAP. XML-Based Security Service Technical Committee работает над стандартизацией механизма авторизации и аутентификации с использованием XML, в том числе языка Security Assertion Markup Language (SAML). Есть комитеты, отвечающие за выработку языков управления ресурсами, за обмен биометрической информацией и целый ряд других.

Для разработки стандартов, описывающих функционирование web-сервисов была создана организация Web Services-Interoperability[1]. Ею был выпущен документ WS-I Basic Profile, определяющий требования к различным компонентам SOA, которые могут гарантировать их совместимость и прояснить тонкости использования web-сервисов.

Программный интерфейс реестра сервисов составляет часть стека протоколов взаимодействия. В наборе технологий web-сервисов таким стандартом является UDDI (Universal Description, Discovery and Integration). Его спецификация – единственная из ядра основополагающих стандартов web-сервисов, разработанная вне рамок консорциума World Wide Web Consortium (W3C). Таким ядром принято считать спецификации, входящие в профиль WS-I Basic Profile, призванный обеспечить общую для различных инструментальных платформ базу взаимно совместимых технологий описания, публикации, обнаружения и вызова сервисов. UDDI обладает весьма развитой функциональностью, существенно более богатой, чем аналогичный компонент набора

[1] http://www.ws-i.org/

стандартов CORBA – CORBA Naming Service. В отличие от предыдущих поколений реестров, UDDI был изначально нацелен на применение как внутри организаций, так и между ними, поэтому реестры UDDI одинаково удобны для ведения информации о нескольких или о тысячах сервисах. Для этого UDDI предусматривает гибкую информационную модель и средства распределения доступа.

С точки зрения применимости UDDI в SOA, наиболее методологически значимым элементом информационной модели UDDI является возможность стандартизации типов сервисов. Интерфейс сервиса, описанный документом WSDL, или даже отдельную его характеристику (например, стоимость или поддержка некоего протокола, такого, как HTTP Basic или WS-Security для авторизации) можно представить самостоятельным объектом метаданных в UDDI. Совокупность ссылок на такие объекты характеризует профиль интероперабельности данного сервиса. Используя те или иные параметры, потребитель может найти в реестре сервис, отвечающий его техническим или деловым потребностям.

С момента публичного представления первой версии UDDI функционирует общедоступный реестр UDDI Business Registry (UBR), который сейчас состоит из четырех географически распределенных реплицируемых узлов: Microsoft (западное побережье США), IBM (восточное побережье США), SAP (Европа) и NTT Telecom (Азия).

Наиболее популярным применением UDDI все же остается организация закрытого сообщества взаимодействующих информационных систем либо внутри компании, либо в строго ограниченном кругу ее деловых партнеров. Очевидно, что частный реестр UDDI при этом является центральным звеном корпоративной сервис-ориентированной архитектуры.

Рабочая группа, ответственная за выработку практических решений для всей отрасли, Web Services Architecture Working Group (WSANG) предлагает следующее определение:

*web-сервис* – это реализуемая программными средствами система для поддержки межмашинного взаимодействия через сеть. Интерфейс сервиса описывается на языке, читаемом машиной, например WSDL. Другие системы взаимодействуют с web-сервисом способом, указанным в его описании, используя сообщения в стандарте SOAP, передаваемые с использованием HTTP и XML и в сочетании с другими стандартами, относящимися к web. Физически, web-сервис представляет собой фрагмент программного обеспечения,

называемый *агентом*. Агент способен передавать и принимать сообщения; он реализует абстрактную функциональность сервиса. Следует проводить различие между агентом и сервисом, один и тот же сервис может быть обеспечен разными агентами.

Web-Сервис предназначен для предоставления некоторой функциональности от лица ее владельца (поставщика). Это могут быть человек или организация (provider entity), которая использует соответствующий собственный агент (provider agent). Запрашивающая сторона (requester entity) тоже может быть человеком или организацией, желающими использовать сервис. Для этого они также используют свой агент (requester agent). Агенты взаимодействуют между собой посредством обмена сообщениями. Для того чтобы этот обмен был успешным, стороны прежде должны достичь соглашения об общем понимании семантики *механизма обмена сообщениями*.

Механизм обмена сообщениями документирован в описании сервисов, которое представляет собой читаемую машиной спецификацию интерфейса, включающую: форматы сообщений, типы данных, транспортные протоколы, форматы сериализации, используемые при обмене между агентами заказчика и поставщика услуг. Она также содержит указание на одну точку в сети (endpoint) или несколько, откуда поставщик доступен и еще может содержать информацию о предполагаемом шаблоне (pattern) сообщения.

Семантика сервиса представляет собой содержание так называемого *контракта* между сторонами; она также содержит некоторые дополнительные сведения о механизмах обмена сообщениями, которые не отражены в WSD. Важно отметить, что хотя контракт содержит всеобъемлющее представление о сервисе, он не должен быть документально зафиксированным. Различие между описанием сервиса и контрактом заключается в том, что первое определяет механизм взаимодействия с сервисом, а второе – смысл и предназначение этого взаимодействия.

Web-Сервисы в основном предназначены для автоматизации процессов управления, которые могли бы быть выполнены вручную, однако в архитектуре web-сервисов остается место и человеку.

Во-первых, его участие требуется при достижении соглашения по семантике и описанию. Человек (или организация) является юридическим владельцем web-сервисов; именно люди должны согласиться с тем, что данная семантика и данное описание будут управлять взаимодействием между сервисами. Возможно, что со стороны поставщика участие может ограничиться

публикацией и предоставлением возможности использовать сервис только в той форме, в какой тот существует, иными словами, контракт предполагает неизменяемые условия со стороны поставщика. Есть и другие формы установления отношений, в том числе через координирующие организации или даже по запросу заказчика.

Во-вторых, агенты заказчика и поставщика сервисов создаются людьми, которые отвечают за исполнение соглашений об описании и о семантике услуги.

На рис. 5.1 представлена схема обмена между потребителем сервисов и поставщиком сервисов с учетом предложений, сделанных WSAWG.

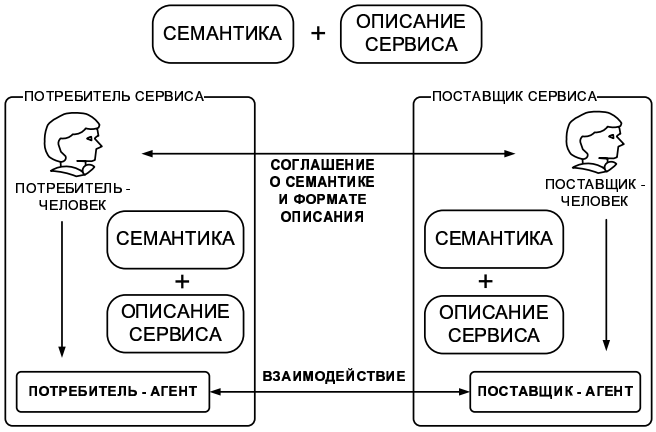


*Рис. 5.1*

Интеграция корпоративных приложений потребовала от разработчиков стандартизации создаваемых программных интерфейсов. На начальном этапе такие стандарты создавались спонтанно – каждый вендор программного обеспечения стремился разработать и внедрить свой стандарт интеграции. Однако конкуренция в данном случае не являлась полезной и до появления специальных комитетов, координирующих работы по созданию межотраслевых стандартов, их было разработано большое количество. В этих стандартах несмотря на разобщенность были заложены основные идеи интеграции. Далее рассматриваются некоторые из них.

**XLANG.** В декабре 2000 г. корпорация "Microsoft" опубликовала XLANG – язык бизнес-процессов на базе XML, используемый в Microsoft BizTalk Server для управления приложениями и web-сервисами XML.

Разработанный "Microsoft" язык XLANG первоначально предназначался для описания последовательных, параллельных и многовариантных потоков работ для BizTalk Server. При описании интерфейсов web-сервиса XLANG использует язык Web Services Description Language (WSDL), предложенный консорциумом "World Wide Web Consortium" (W3C). Основное назначение XLANG состоит в определении бизнес-процессов и организации обмена сообщениями между web-сервисами. Кроме того, он имеет надежные средства

обработки исключительных ситуаций с поддержкой длительных многоуровневых транзакций.

**WSFL.** Язык потоков данных web-сервисов WSFL (Web Services Flow Language корпорации "Microsoft") является приложением языка XML для описания web-сервисов как части определения бизнес-процесса. IBM разработала WSFL как часть инфраструктуры технологии web-сервисов, опираясь на спецификации таких существующих стандартов, как SOAP, UDDI, WSDL и XMLP. WSFL позволяет описывать как публичные, так и частные процессы. Он определяет обмен данными, последовательность выполнения (модель потока) и выражение каждого шага потока в виде конкретных операций (глобальная модель). WSFL поддерживает интерфейс WSDL (что позволяет решать задачи рекурсивной композиции), располагает средствами обработки исключительных ситуаций, но не поддерживает транзакции.

**ebXML.** Центр ООН по содействию торговле и электронному бизнесу (UN/CEFACT) на основе XML разработал язык Electronic Business Extensible Markup Language. Стандарт ebXML обеспечивает набор программных компонентов промежуточного слоя, облегчающих сотрудничество деловых партнеров. В него входит схема спецификаций бизнес-процессов (Business Process Specification Schema, BPSS) . Протокол BPSS позволяет определять как хореографию, так и коммуникационные протоколы между web-сервисами.

**WSCL.** Компания "Hewlett-Packard" разработала стандарт моделирования последовательности взаимодействий web-сервисов – язык Web Services Conversation Language. В марте 2002 г. WSCL был опубликован как "Примечание W3C" (W3C Note).

**SOAP.** Simple Object Access Protocol (SOAP) – основанный на XML протокол, предназначен для обмена информацией в распределенных системах. SOAP устанавливает стандарт взаимодействия "клиент–сервер" и регламентирует, как должен осуществляться вызов, а также, как должны передаваться параметры и возвращаемые значения. Для представления любой информации, передаваемой от клиента к серверу, и наоборот, используется XML [19].

SOAP не накладывает ограничений на используемый транспорт. Для передачи сообщений могут использоваться любые протоколы (например, протоколы HTTP, HTTPS, SMTP) и продукты. Данные могут передаваться через Microsoft Message Queueing, IBM MQ Series и т. д. Однако чаще всего используется протокол HTTP. SOAP описывает преобразование в XML следующих элементов вызова: запрос (SOAP Request); отклик (SOAP Response); сообще-

ние об ошибке (SOAP Fault). Вызов метода по протоколу SOAP преобразуется в XML следующим образом:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope ...>
  <SOAP-ENV:Body ...>
  <SOAPSDK4:Add ...>
    <x>1</x>
    <y>2</y>
  </SOAPSDK4:Add>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

В приведенном примере запроса вызывается метод Add, которому передаются два параметра: 1 и 2. Информация о том, какой метод и у какого объекта необходимо вызвать, передается в заголовке. В случае протокола HTTP заголовок может выглядеть так:

```
<HTTPHeaders>
  <soapaction>"http://tempuri.org/Sample1/action/Adder.Add"</soapaction>
  <content-type>text/xml; charset="UTF-8"</content-type>
  <user-agent>SOAP Toolkit 3.0</user-agent>
  <host>ivan:8080</host>
  <content-length>516</content-length>
  <connection>Keep-Alive</connection>
  <cache-control>no-cache</cache-control>
  <pragma>no-cache</pragma>
</HTTPHeaders>
```

В теге SoapAction указывается, какое действие необходимо выполнить на сервере.

Ответ сервера (SOAP Response) содержит значения возвращаемых параметров:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<SOAP-ENV:Envelope ...>
  <SOAP-ENV:Body ...>
  <SOAPSDK4:AddResponse ...>
    <Result>3</Result>
  </SOAPSDK4:AddResponse>
```

```
  </SOAP−ENV:Body>
</SOAP−ENV:Envelope>
```

В данном случае сервер ответил на вызов метода Add с параметрами 1 и 2 значением 3.

В случае возникновения ошибок до или во время обработки запроса сервер возвращает информацию об ошибке (SOAP Fault). Ошибки могут быть двух типов:

- Клиентские ошибки. Неправильный запрос к серверу (запрос содержит неверные значения параметров, неправильно сформирован и т. д.).
- Серверные ошибки. Могут быть связаны как с функционированием самого сервера (закончились свободные ресурсы), так и с обработкой запроса (компонент сервиса вернул ошибку).

Пример серверной ошибки:

```
<?xml version="1.0" encoding="UTF−8" standalone="no" ?>
<SOAP−ENV:Envelope ...>
  <SOAP−ENV:Body ...>
  <SOAP−ENV:Fault>
    <faultcode>SOAP−ENV:Server</faultcode>
    < faultstring >Can add no more numbers</ faultstring >
    < faultactor >http: // ivan:8080/Sample1/Sample1.ASP</faultactor>
    <detail>
    < mserror:errorInfo    ... >
      <mserror:returnCode>−2147467259 : Unspecified error
      </mserror:returnCode>
      < mserror:serverErrorInfo >
      < mserror:description >Can add no more numbers</ mserror:description >
      <mserror:source>Sample1.Adder.1</mserror:source>
      </ mserror:serverErrorInfo >
      <mserror:callStack>
      <mserror:callElement>
        <mserror:component>WSDLOperation</mserror:component>
        < mserror:description >Executing method Add failed
        </ mserror:description >
        <mserror:returnCode>−2147352567 : Exception occurred.
        </mserror:returnCode>
```

```
        </ mserror:callElement >
         ...
        </ mserror:callStack >
      </ mserror:errorInfo >
      </ detail >
   </SOAP−ENV:Fault>
   </SOAP−ENV:Body>
</SOAP−ENV:Envelope>
```

В теге faultcode указывается источник ошибки – клиент или сервер. Faultstring содержит строковое описание ошибки. Faultfactor указывает URL, по которому обращался клиент. Тег detail содержит дополнительную информацию об ошибке (например, call stack).

## 5.2. Интерфейс управления хореографией

Спецификация языка описания интерфейсов Web Services Choreography Interface (WSCI) разработана компаниями "Sun Microsystems", SAP, BEA и "Intalio". Она устанавливает расширение языка WSDL, направленное на организацию совместной работы. В общем виде WSCI определяет хореографию, или обмен сообщениями между web-сервисами. Спецификация обеспечивает корреляцию сообщений, правила упорядочения, обработку исключительных ситуаций, транзакции и динамическое взаимодействие.

Как видно из рис. 5.2, WSCI описывает лишь наблюдаемое поведение взаимодействующих web-сервисов. При этом в отличие от BPEL не затрагиваются определения исполняемых бизнес-процессов. Кроме того, интерфейс WSCI описывает участие в обмене сообщениями с позиции лишь одного из партнеров. Хореография WSCI должна включать в себя набор интерфейсов WSCI – по одному для каждого партнера, участвующего во взаимодействии. В WSCI отдельный процесс взаимодействием не управляет.

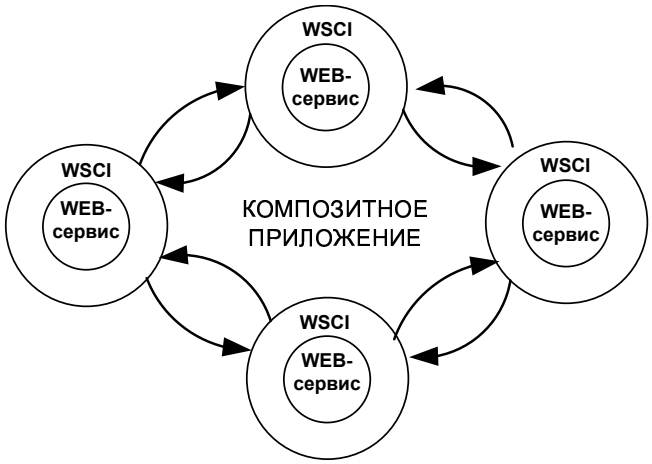


*Рис. 5.2*

Каждое элементарное действие WSCI представляет собой единицу работы, которая связана с определенной операцией WSDL. Спецификация WSCI расширяет возможности WSDL, позволяя упорядочить выполнение доступных операций WSDL. Другими словами, WSDL для каждого доступного сервиса описывает точки входа, а WSCI определяет взаимодействия операций WSDL.

Спецификация интерфейса относится только к наблюдаемому поведению взаимодействующих web-сервисов и не затрагивает определений исполняемого бизнес-процесса.

WSCI поддерживает как базовые, так и структурные действия. К примеру, тег action определяет базовое сообщение для запроса или ответа. Каждое действие определяет соответствующую операцию WSDL и конкретного участника, который ее выполняет. К внешним сервисам можно обратиться с помощью тега call. WSCI поддерживает разнообразные структурные действия включая выполнение циклов, последовательную и параллельную обработки.

Следующий интерфейс WSCI определяет процесс закупки, в который входят два последовательных действия – Receive Order ("получить заказ") и Confirm ("подтвердить"). Каждое действие отображается в набор абстрактных операций WSDL, а WSCI устанавливает между ними корреляцию:

```
<process name="Purchase" instantiation ="message">
<sequence>
        <action name="ReceiveOrder"
                role="Agent" operation="tns:Order">
        </action>
        <action name="Confirm"
                role="Agent" operation= "tns:Confirm">
                <correlate correlation = "tns:ordered"/>
                <call process="tns: Purchase"/>
        </action>
</sequence>
</process>
```

WSCI поддерживает бизнес-транзакции и обработку исключительных ситуаций. Разработчик бизнес-процесса может установить определенный контекст транзакции в рамках интерфейса WSCI, подобный контексту действия в BPEL4WS. Этот контекст определяет группу действий, неудачное завершение любого из которых "откатит" назад всю группу.

## 5.3. Язык исполнения бизнес-процессов BPEL4WS

BPEL4WS (Business Process Execution Language for Web Services) построен на основе XML и web-сервисов. Он использует язык, базируемый на XML, который поддерживает технологический стек web-сервисов включая SOAP, WSDL, UDDI, WS-Reliable Messaging (WS-надежная передача сообщений), WS-Addressing (WS-адресация), WS-Coordination (WS-координация) и, наконец, WS-Transaction (WS-транзакция). BPEL представляет конвергенцию (convergence) двух более ранних языков автоматизации документооборота (workflow) – языка потоков данных услуг Web (WSFL) и XLANG. Основу BPEL составляют три ключевых свойства:

- асинхронность;
- координация потоков;
- управление исключительными ситуациями.

Первая версия BPEL была разработана в августе 2002 г. С тех пор много основных вендоров соединилось в разработке модификаций и усовершенствований BPEL, что привело к принятию в марте 2003 г. версии 1.1. В апреле 2003 г. BPEL представлен OASIS с целью стандартизации и был сформирован Технический комитет по исполнительному языку бизнес-процессов на web-сервисах (Web Services Business Process Execution Language Technical Committee, WSBPEL TC). Это привело к еще более широкому применению BPEL в промышленности. Текущая версия спецификации BPEL – 1.1[1]. Официальными представителями Комитета по стандартизации для языка BPEL являются фирмы IBM, "BEA Systems", "Microsoft", SAP AG, "Siebel Systems".

В модели, принятой в BPEL4WS, процесс – есть центр мира. Внешние системы из него лишь вызываются. Без ведома процесса может произойти только одно событие – его инициация. Любые другие сообщения из внешнего мира ожидаются только тогда, когда они требуются логикой процесса.

Взаимодействие с партнерами всегда происходит через web-сервисы (т. е. через обмен сообщениями), а команды и данные для этого взаимодействия инкапсулируются в структуре под названием *"партнерская связь"*. Партнерские связи опираются на WSDL-совместимые описания операций и сервисов. BPEL4WS использует переменные для хранения данных, связанных с состоянием процесса; как правило, они содержат сообщения, полученные от партнеров. Глобальные переменные всего процесса называются свойствами – они

[1]См.: http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

применяются как механизм передачи данных между потоками процесса и как средства сохранения его состояния в целом. BPEL4WS делит все процессы на два типа:

- абстрактные (т. е. бизнес-протоколы);
- развернутые (instantiated).

BPEL4WS-Процесс представляет все связи с партнерами как взаимодействие через абстрактный WSDL-интерфейс (типы портов и операции), – на реальные сервисы ссылок не делается. BPEL поддерживает два различных способа описания бизнес-процессов, которые поддерживают оркестровку и хореографию:

- Выполнимые процессы (Executable processes) позволяют определять точную детализацию бизнес-процессов. Они находятся в парадигме оркестровки и могут быть выполнены механизмом оркестровки.
- Абстрактные бизнес-протоколы (Abstract business protocols) допускают спецификацию общего обмена сообщениями только между сторонами. Они находятся в парадигме хореографии, не включают внутренние детали потока процессов и не являются выполнимыми.

BPEL-Процесс определяет точный порядок, в котором должны быть вызваны – последовательно или параллельно – задействованные web-сервисы. На BPEL можно выразить условные переходы. Например, обращение к web-сервису может зависеть от значения предыдущего вызова. Можно также создавать циклы, объявлять переменные, копировать и назначить значения, определить обработчиков ошибки и т. д. Комбинируя все эти конструкции, можно алгоритмическим образом определять сложные бизнес-процессы. Фактически, поскольку бизнес-процессы – это, по существу, графы действий, могло бы быть полезным выразить их в виде диаграмм активности UML.

В типичном случае BPEL-бизнес-процесс получает запрос. Чтобы его выполнить, процесс вызывает задействованные web-сервисы и затем отвечает вызвавшему его абоненту. Поскольку BPEL-процесс связывается с другими web-сервисами, то он в значительной степени полагается на WSDL-описание web-сервисов, вызванных структурным web-сервисом.

BPEL-Процесс состоит из шагов; каждый шаг является действием ("activity"). BPEL поддерживает как примитивные (primitive), как и структурные действия (structure activities).

Примитивные действия представляют собой основные конструкции, которые используются для общих задач, например следующей:

- *invoke* – вызов других web-сервисов;
- *receive* – ожидание клиента, чтобы вызвать бизнес-процесс посредством посылки сообщения, и получение запроса;
- *reply* – генерация ответа для синхронных операций;
- *assign* – управление переменными данных;
- *throw* – индикация ошибок и исключений;
- *wait* – ожидание в течение некоторого времени;
- *terminate* – завершение процесса.

Спецификация поддерживает как структурные действия для управления потоком работ бизнес-процесса в целом, так и базовые действия, которые включают взаимодействия с внешними web-сервисами.

## 5.4. Контрольные вопросы

1. Какова роль стандартов в развитии распределенных приложений?
2. Какие организации занимаются разработкой отраслевых стандартов в области СОА?
3. Что такое UDDI?
4. Какие стандарты используются для описания реестров сервисов?
5. Что такое web-сервис?
6. Что такое контракт в терминах СОА?
7. Перечислите основные спецификации в области сервисных архитектур.
8. Каково функциональное назначение SOAP?
9. Что общего у SOAP и RPC?
10. Опишите назначение языка BPEL4WS.

# 6. ПРОГРАММНЫЕ СИСТЕМЫ И СРЕДЫ

Далее рассматриваются наиболее распространенные платформы интеграции приложений и создания композитных сервис-ориентированных систем. Основное внимание уделяется программным продуктам Oracle, IBM, BEA.

## 6.1. BEA WebLogic

BEA WebLogic Integration – это инструмент интеграции, входящий в платформу BEA WebLogic Platform, которая строится по принципу "все включено". Используя единую оболочку, его пользователь может осуществлять работы, связанные с интеграцией приложений и информационным взаимодействием с бизнес-партнерами (B2B), прописывать бизнес-процессы, а также создавать бизнес-логику программ на языке Java. Всего в платформу входит пять главных компонентов:

- виртуальная машина Java;
- сервер приложений;
- средство построения порталов;

- пакет инструментов интеграции;
- среда разработки.

WebLogic Integration относится к тому классу продуктов, возможности интеграции которых опираются на сервер приложений: он используется и в качестве среды исполнения для логики брокера сообщений, и как система выполнения сценариев бизнес-логики и портала.

Ключевым преимуществом своей платформы BEA считает возможность снижения требований к группе разработки за счет использования трехуровневого подхода к созданию ПО. Это уровни бизнес-аналитика (оперирует понятиями потоков документов), сборщика прикладной логики из готовых компонентов и разработчика компонентов J2EE. В платформу интегрированы также средства генерации программного кода из визуальных описаний, что в принципе позволяет создавать программы на Java специалистам смежных областей, например знатокам Visual Basic или COBOL.

Помимо компонентов разработки в платформу включены инструменты для управления и создаваемой инфраструктуры, в том числе управления ее производительностью, настройками безопасности и жизненным циклом системы и ее сопровождения.

BEA является одним из ведущих вендоров, вкладывающих средства в развитие последних технологических новшеств наподобие семейства стандартов XML (XSLT, XQuery и пр.) и web-сервисов, что нашло отражение в предлагаемой платформе, в которой поддержка новейших стандартов реализована наиболее полно.

В платформе есть развитая функциональность MOM (Message Oriented Middleware – межплатформное ПО, ориентированное на работу с очередями), в основе которой лежат средства гарантированной доставки, совместимые со стандартом JMS (Java Messaging Service – служба сообщений Java), и брокер сообщений, функционирующий под управлением сервера приложений.

WebLogic представляет один из самых полных наборов интерфейсов для интеграции корпоративных приложений, файлов и баз данных разной природы.

Для платформы создано много готовых коннекторов (практически для всех основных ERP-систем и систем документооборота), синтаксических анализаторов форматов файлов, средств для обращения к всем исполняемым модулям программ Windows и Java, а также для взаимодействия с интеграционными платформами других фирм. Есть и среда для ускоренного создания

новых коннекторов. Кроме того, BEA предлагает специализированные расширения (интерфейс Application View) для создания еще одного слоя абстракции, скрывающего от пользователя сложности EAI-адаптеров.

Однако подключение приложений старше 10 лет или использование сетей с унаследованными транспортными протоколами могут потребовать от разработчиков нетиповых подходов, поскольку набор стандартных средств для этого в платформе не очень обширен.

## 6.2. IBM WebSphere

Платформа WebSphere – семейство программных продуктов фирмы IBM. Часто WebSphere употребляется в качестве названия одного конкретного продукта: WebSphere Application Server (WAS). WebSphere относится к категории middleware – промежуточного программного обеспечения, которое позволяет приложениям электронного бизнеса (e-business) работать на разных платформах на основе веб-технологий.

WebSphere использует открытые стандарты J2EE, XML и web-службы. Разработка ведётся в лабораториях IBM по всему миру. В России доступна локализованная версия WebSphere.

Платформа WebSphere корпорации IBM является наиболее полнофункциональной среди всех других наборов инструментов EAI. Она поддерживает разные стили интеграции на уровнях:

- данных;
- обмена сообщениями;
- сквозных бизнес-процессов;
- B2B-интеграции;
- исполнения бизнес-логики программ на языке Java.

В платформе присутствуют следующие интеграционные инструменты:

- две системы MOM:
  - Business Integration InterChange Server (ICS),
  - MQ Business Integration Message Broker (WSMB);
- сервер приложений Application Server (WAS);
- портальное ПО Portal Server, функционирующее на базе WAS;
- система workflow, которая стыкуется с WSMB.

В состав WebSphere был включен Business Integration Workbench – средство проектирования бизнес-процессов и управления ими. Помимо перечисленных основных продуктов выпускаются еще десятки дополнений к ним.

Большинство продуктов, входящих в платформу, являются образцовыми в своих классах. Таковы, например, брокер сообщений WSMB или сервер приложений WAS. Последний предлагает полную поддержку J2EE 1.3 и ряд возможностей, которые можно найти только в платформе BEA WebLogic.

К их числу относятся встроенные возможности высокоуровневого (т. е. без программирования) задания бизнес-правил и сценариев workflow, что позволяет эффективнее связывать компоненты EJB в рамках общего сценария бизнес-логики. Аналогичным образом этот сервер приложений позволяет работать с web-сервисами: собирая их в рамках единого процесса и публикуя его затем как новый web-сервис. Анализ развития сервера приложений показывает, что он все больше ориентируется на интеграцию на уровне бизнес-процессов.

IBM как основной производитель мэйнфреймов уделяет большое внимание поддержке унаследованных платформ, и в WebSphere есть обширные средства для решения этой задачи. С другой стороны, как один из законодателей в области новейших технологий корпорация обеспечила в платформе и широкую поддержку web-сервисов. Средства для работы с ними предлагаются на каждом уровне – от брокера сообщений до портала.

К недостаткам платформы можно отнести то, что весьма сложно организовать защиту информации в системе, опирающейся на интеграционные продукты IBM (в частности, из-за их количества). Поэтому для комплексного решения этой задачи лучше опираться специальные средства управления.

Стоимость продуктов IBM высокая. Хотя (как и компания BEA) корпорация предлагает варианты от $500 для ограниченного по функциональности и числу пользователей решения WebSphere Express, основные продукты типа WebSphere Business Integration Server могут стоить сотни тысяч долларов. Однако главные затраты проекта связаны с консалтингом: без привлечения квалифицированных специалистов в интеграционном проекте обойтись непросто – столь обширен и сложен продуктовый ряд этого вендора.

В дальнейшем, правда, эти начальные затраты должны окупиться (особенно в крупных структурах), так как любую возникающую затем техническую проблему можно решить средствами IBM. Кроме того, крупная организация может сэкономить на стоимости владения создаваемым решением, так как IBM предлагает множество инструментов и средств разработки, упрощающих управление этим решением и его развитие.

### *6.2.1. Компоновки IBM WebSphere*

Далее приводится краткая сводка по комплектациям продуктов IBM в сфере интеграции и СОА.

**WebSphere Business Integration for Automotive.** Поддерживает бизнес-процессы автомобилестроительных компаний, поставщиков и дистрибьюторов, связанные с проектированием, закупками, производством, распространением и обслуживанием автомобилей. Автоматизированные и оптимизированные процессы помогают ускорить выход на рынок новых продуктов, сократить расходы на хранение запасов и производственные затраты, улучшить процессы сбыта и обслуживания для компаний-дилеров, а также усовершенствовать управление цепочками поставок.

**WebSphere Business Integration for Banking.** Предназначен банкам, обслуживающим мелких и корпоративных клиентов, а также фирмам, оказывающим разнообразные финансовые услуги. Обеспечивает быстрый и безопасный доступ клиентов к финансовой информации в ежедневном круглосуточном режиме; позволяет банкам получить единый ракурс транзакций клиентов, что помогает им увеличить объемы продаж "сопутствующих" продуктов и услуг для всех розничных каналов.

**WebSphere Business Integration for Financial Networks.** Позволяет глобальным банкам управлять обработкой платежей путем консолидации разнородных сетей обмена сообщениями, облегчает переход к новым Интернет-сетям, снижает общую стоимость владения.

**WebSphere Business Integration for Electronics.** Ориентирован на компании, производящие электронику; позволяет интегрировать и оптимизировать операции проектирования и производства, обеспечивая ускорение вывода на рынок новых продуктов, сокращение расходов на хранение запасов, оптимизацию цепочек поставок, улучшение исполнения заказов и обслуживания заказчиков. Кроме того, этот продукт позволяет соединить между собой "унаследованные" системы и разнородные приложения, например, решения для управления взаимоотношениями с заказчиками, по планированию ресурсов предприятия и управлению цепочками поставок. Это помогает эффективно передавать нужную информацию (например, проектные обновления и новые сведения о складских запасах) всем заинтересованным сторонам как внутри компании, так и за ее пределами.

**WebSphere Business Integration for Energy and Utilities.** Обеспечивает интеграцию процессов, поддерживающих эксплуатацию, управление актива-

ми и их обслуживание, а также обслуживание заказчиков. Например, использование WebSphere для интеграции системы, отвечающей за бесперебойное электропитание, с другими приложениями помогает сократить время, необходимое для восстановления подачи электричества в случае его аварийного отключения. Кроме того, этот продукт облегчит общение с прессой, должностными лицами и заказчиками в той части, которая касается статуса отключения электроэнергии и ожидаемого времени восстановления ее подачи.

**WebSphere Business Integration Express for Item Synchronization.** Помогает компаниям среднего размера связать информацию из своих цепочек поставок с услугами UCCnet, которые дают основу для проведения через Интернет транзакций по регистрации событий и синхронизации данных в системах B2B.

**WebSphere Business Integration Connect Express.** Помогает клиентам быстро и легко вступать в сложно организованные деловые сообщества, сводя к минимуму риски и затраты, характерные для традиционных сред B2B. Благодаря поддержке широкого спектра форматов обмена данными и протоколов Интернета это решение позволяет компаниям быстро обеспечить выполнение таких отраслевых нормативов, как AS/2.

**WebSphere MQ Express.** Представляет собой ПО обмена сообщениями, обеспечивающее их гарантированную доставку в реальном времени или в асинхронном режиме при повышенной готовности и пропускной способности системы. Для безопасного обмена данными используется протокол Secure Sockets Layer (SSL); предусмотрены также средства организации обмена данными между кластерными системами.

## 6.3. Microsoft .NET/BizTalk

Платформа Microsoft дает разработчику ту же функциональность, что и J2EE, но, только в рамках Windows-платформ. Инструменты, необходимые для реализации различных интеграционных подходов, разнесены в ней по нескольким продуктам, а часть функций включена непосредственно в ОС (например, компонент управления транзакциями MTS, web-сервер Internet Information Server, библиотеки и среда исполнения "управляемого кода" .Net).

Основную функциональность EAI несет BizTalk Server 2004 (далее BTS2004) – сервер интеграции на базе XML. Он может работать и как брокер сообщений, т. е. осуществлять преобразование и коммутацию поступающих в него сообщений, и как механизм выполнения бизнес-сценариев. В отличие от

платформ на базе J2EE, где сервер приложений является основой для исполнения всей бизнес-логики – и низкого (компоненты EJB), и высокого (через механизмы workflow) уровней, BizTalk отвечает только за высокоуровневую бизнес-логику и интеграцию систем, а выполнение логики низкого уровня реализуется моделью COM+ или .Net.

**Создание схемы бизнес-процесса.** Архитектурно BTS2004 состоит из нескольких модулей: графических редакторов карт преобразований XML-сообщений и сценариев бизнес-логики, а также механизмов исполнения карт. В последней версии появились компоненты, обеспечивающие работу пользователей с бизнес-процессами через web-браузер или клиентские программы типа Microsoft InfoPath.

Принятая "Microsoft" модель интеграционной разработки позволяет разделить работу программиста и работу аналитика бизнес-процессов. Она похожа на ту, что заложила фирма BEA в свою платформу WebLogic Integration Platform (последняя опирается на J2EE, а не на COM+). Бизнес-аналитик может графически рисовать бизнес-процесс (т. е. диаграммы workflow – схемы обмена документами и передачи управления), специалист по интеграции определяет точки вызова внешней функциональности (реализуемой COM-объектами, кодом .Net, web-сервисами и т. п.), а разработчик низкого уровня программирует эту функциональность.

В версии 2004 разделение труда программиста и других участников проекта интеграции усилено. Помимо архитектора, задающего общую канву процесса, в настройке системы могут теперь участвовать и пользователи-предметники. Для этого введены два новых механизма – бизнес-правил (Business Rules) и конфигурирования процесса (Business Process Configuration).

По мнению экспертов, BizTalk Server имеет три неоспоримых преимущества перед другими рассмотренными продуктами:

- он крайне прост идеологически;
- позволяет разработчикам решений интеграции наиболее полно использовать свои знания платформы Windows;
- интеграцию систем с помощью BizTalk можно осуществлять постепенно – начать с простого проекта с обменом файлами и нарастить его до сложной системы со сквозными бизнес-процессами.

Однако эти же свойства определяют и слабые стороны продукта: BizTalk функционирует только на платформе Windows и в существенной степени опирается на нее. Поэтому несмотря на то, что партнеры "Microsoft" предлагают

много коннекторов для разных систем и файловых форматов, при попытке использовать BizTalk в гетерогенной среде, особенно содержащей старые платформы и унаследованные протоколы связи, могут возникнуть непредвиденные сложности.

Для снижения сложностей внедрения продукта в ряде вертикальных отраслей в линейке продуктов Microsoft имеются дополнительные платные наборы, состоящие из шаблонов решений и специализированных компонентов, поддерживающих типовые для этих отраслей протоколы обмена данными. Такие ускорители (Accelerators) есть, например, для финансового сектора и сектора здравоохранения.

## 6.4. Oracle 10g

Корпорация "Oracle" поставляет технологии нескольких классов, ориентированные на разные стили интеграции. Это интеграция на уровнях данных (технология Transparent Gateways и коннекторы базы данных), пользовательского интерфейса (портал), сервера приложений и системы MOM. Полный набор средств для интеграции приложений называется Oracle Integration.

В Oracle Application Server 10g Release 2 (OAS10g) объединены продвинутые возможности COA для стратегически важных приложений, позволяющие предложить наиболее сплоченные и всеобъемлющие заказные решения в отрасли для масштабирования, обеспечения безопасности web-сервисов и сети и управления ими, подходящие для любых сред ИТ.

В OAS10g продолжается использование преимуществ двух важных технологических тенденций – корпоративных вычислений на базе сервисов (Service-Oriented Computing) и сетевых вычислений (Grid Computing):

- Сервис-ориентированная архитектура. Архитектура программного обеспечения, облегчающая разработку корпоративных приложений как модульных бизнес-сервисов. OAS10g предлагает всеобъемлющую инфраструктуру SOA, дающую возможность разрабатывать, создавать оболочки, согласовывать, подготавливать к работе, управлять, обеспечивать безопасность, объединять (federate), обнаруживать корпоративные приложения и обеспечивать доступ к ним как к сервисам. Корпоративные вычисления на базе сервисов обеспечивают гибкую инфраструктуру корпоративных приложений. OAS10g в качестве дополнения к COA поддерживает также управляемые событиями вычисления, чтобы сделать возможными в реальном времени приложения для считывания данных с датчиков

и реагирования на их показания (например, системы на базе RFID).

- Сетевые (grid) вычисления. Архитектура программного обеспечения, которая координирует использование большого количества дешевых собираемых из модулей серверов и блоков памяти, чтобы эксплуатировать стратегически важные бизнес-приложения. Сетевые вычисления в значительной степени снижают инвестиции в аппаратные средства и позволяют постоянно наращивать вычислительные мощности. OAS10g может использоваться для разворачивания, масштабирования и обеспечения безопасности приложений и пользователей в сети, и для управления ими.

В качестве базовых технологических решений предлагаются:

- Business Intelligence (системы анализа бизнес-информации предприятия), позволяющие организациям собирать, анализировать и распределять информацию;
- Business Integration (бизнес-интеграция), позволяющие организациям объединять отдельные системы друг с другом и автоматизировать бизнес-процессы;
- Identity Management (управление идентификационными параметрами личности ), позволяющие организациям консолидировать администрирование защиты, чтобы снизить полную стоимость владения ими и сократить число уязвимых мест защиты.

Общим для этих решений является опора на СУБД Oralce 10g, в которой хранятся как все метаданные и настройки интеграции, так и рабочие данные – свойства объектов и очередей сообщений. Использование СУБД для таких целей несколько повышает планку требований к оборудованию, но одновременно увеличивает надежность, защищенность, масштабируемость решения и его способность к инкорпорированию данных из других систем.

В роли второго основного строительного блока платформы выступает сервер приложений OAS10g. Он представляет собой композиционный продукт, включающий инфраструктурные компоненты J2EE, репозиторий данных (в том числе СУБД для репозитория) и т. п. На нем базируется Oracle Integration, в который входят два основных модуля.

Первый, Oracle InterConnect, является брокером сообщений, обеспечивающим асинхронное взаимодействие приложений и построенным по архитектуре hub and spoke. Так же, как и в IBM WebSphere ICS, в нем применяется идеология преобразования сообщений о происходящих в интегрируемых приложениях событиях в некоторый "обобщенный" вид.

Второй – OracleAS ProcessConnect – это средство для интеграции на уровне бизнес-процессов и B2B. В предыдущих версиях для подобной интеграции приходилось применять связку Oracle InterConnect и Oracle Workflow, теперь подобная интеграция делается более прозрачно. Как и в платформах BizTalk, IBM, WebLogic и SAP NetWeaver, интеграция может вестись не только на уровне приложений, но и на уровне бизнес-партнеров – через обмен данными EDI, RosettaNet, UCCnet и HIPAA. Подобно большинству других платформ, OracleAS и Oracle ProcessConnect обеспечивают поддержку web-сервисов.

На базе самого сервера приложений пользователь может строить новое транзакционное и web-ориентированное прикладное ПО, подключая через JCA унаследованные платформы в синхронном режиме. Технология JCA 1.0 применяется и как основная, и для построения адаптеров к приложениям и протоколам для модуля ProcessConnect.

Положительным свойством платформы 10g является наличие средств управления интеграционными метаданными. Помимо того, что метаданные вообще лежат в основе данной платформы интеграции, важно и применение двух репозиториев – для этапов разработки и работы. При этом имеются средства для перемещения конфигурации из одного репозитория в другой, а также для клонирования настроек, хранящихся в репозитории. Указанные репозитории построены на основе СУБД Oracle, что обеспечивает их высокую производительность и надежность.

В линейке продуктов Oracle можно выделить следующие пакеты:

- Oracle Integration InterConnect. Обеспечивает полные функциональные возможности шинной организации служб предприятия ESB для быстрого развертывания интеграционных решений на всем предприятии.
- Oracle BPEL Process Manager. Продукт для управления бизнес-процессом (Business Process Management, BPM), позволяющий разрабатывать, компоновать и отлаживать сквозные бизнес-процессы, включающие людей, партнеров и приложения.
- Oracle Integration B2B. Полное решение B2B, поддерживающее ведущие отраслевые протоколы для комплексной и быстрой интеграции партнеров.
- Oracle Integration BAM. Управляемая событиями платформа для агрегирования, корреляции и представления событий на предприятии в пределах контекста, понимаемого бизнесом.

Oracle Integration B2B предлагает поддержку протоколов, делающую возможной развертывание признанных отраслью стандартов: RosettaNet, элек-

тронного обмена данными (Electronic Data Interchange, EDI), Applicability Statement 2 (AS/2) и заказных конфигураций. Эта поддержка включает:

- Процесс: RosettaNet Partner Interface Process (PIP).
- Документ: EDI X12, EDIFACT EDI, X12-HIPAA, PIP BD, UCCnet.
- Обмен: AS2, структура реализации RosettaNet Framework (RNIF).
- Транспорт: HTTP, SMTP, IMAP, FTP, FTPS, файл.
- Пакетирование: MIME, S/MIME.

Встроенные сервисы интеграции дают разработчикам возможность без труда использовать из стандартных процессов BPEL расширенные технологические процессы и возможности функциональной совместимости и преобразования. К числу этих возможностей относятся поддержка преобразований XSLT и XQuery, а также связывание с сотнями унаследованных систем через адаптеры JCA и "родные" протоколы. Сервисы технологических процессов с участием человека (типа управления задачами, управления уведомлениями и управления идентификационными параметрами личности) обеспечиваются как встроенные сервисы BPEL, чтобы интегрировать в потоки BPEL людей и ручные задачи. Расширяемая структура связывания WSDL делает возможной функциональную совместимость со многими протоколами и форматами сообщения наряду с SOAP. Связывания доступны для JMS, электронной почты, JCA, HTTP GET и POST, а также для многих других протоколов, разрешающих простую функциональную совместимость с сотнями серверных систем. Ниже приводятся некоторые готовые к употреблению сразу же после установки адаптеры, которые делают возможной работу сервисов интеграции:

- пакетированные приложения: SAP, PeopleSoft, Siebel;
- адаптеры для унаследованных систем: CICS, IMS DB, IMS TM, DB2, VSAM;
- адаптеры B2B: RosettaNet, EDI
- технологические адаптеры: HTTP, SMTP, FTP, JMS, Database, Advanced Queuing, web-сервисы

Oracle BPEL Process Manager выполняет стандартные процессы BPEL и обеспечивает возможность "дистилляции", чтобы состояние потоков, выполняющихся длительное время, автоматически поддерживалось в базе данных. Это делает возможной кластеризацию как в целях автоматического преодоления последствий сбоев, так и для достижения масштабируемости. Некоторые расширенные возможности Oracle BPEL Process Manager включают:

- Параллельное выполнение. Oracle BPEL Process Manager обеспечивает

возможность параллельного выполнения ряда задач с целью расширения узких мест процесса.

- $N$-Поточность. Расширение параллельного выполнения. Обеспечивает возможность разбиения процесса на $N$ параллельно выполняющихся ветвей выполнения, где $N$ определяется динамически во время выполнения.
- Компенсация. Oracle BPEL Process Manager обеспечивает поддержку компенсирующих транзакций, которые являются альтернативной моделью транзакции в тех случаях, когда транзакции в стиле XA не могут использоваться (либо из-за долговременной природы "транзакции", либо из-за включения сервисов, которые не поддерживают транзакции стиля XA/JTA).

## 6.5. SAP NetWeaver

Платформа SAP NetWeaver – продукт лидера в секторе ERP-систем крупных предприятий России. NetWeaver позиционируется как платформа интеграции, однако главные ее достоинства лежат в области, смежной с областью технологических задач интеграции, и являются более высокоуровневыми. Ключевые ее свойства принадлежат таким областям, как управление контентом, построение информационных порталов для обеспечения взаимодействия людей, создание аналитического инструментария (управления знаниями), высокоуровневая работа с данными (средства синхронизации справочников).

В документации NetWeaver выделяются слои сервера приложений, интеграции процессов, интеграции информации (реально это средства обработки неструктурированного и структурированного контентов включая управление знаниями) и интеграции людей (т. е. портал со средствами поддержки сотрудничества пользователей).

Большинство из этих инструментов не являются системой интеграции структурированной информации. В этой области платформа NetWeaver ориентирована в первую очередь на модель интеграции на базе сквозных бизнес-процессов. В дальнейшем данное направление будет, очевидно, развито, о чем свидетельствуют соглашения о более тесной интеграции NetWeaver со средствами моделирования БД ARIS фирмы "Sheer AG".

Одна из главных задач NetWeaver связана с интеграцией приложений самой SAP. Компоненты NetWeaver, например сервер приложений, способный исполнять не только код Java, но и код сценариев ABAP), позволяют открыть бизнес-функциональность системы для интеграционных решений.

Основой NetWeaver является сервер приложений SAP Web Application Server, на базе которого развернута инфраструктура обмена данными SAP Exchange Infrastructure (SAP XI).

Фактически SAP XI представляет собой реализацию брокера сообщений и предлагает типичные его функции включая преобразование сообщений, их маршрутизацию, механизмы публикации и подписки. В качестве входного формата данных шины используется XML. Поверх SAP XI функционирует также механизм управления бизнес-процессами (workflow).

К SAP XI через web-сервисный протокол SOAP пристыковываются внешние приложения. Под управлением сервера приложений функционируют JCA-совместимые адаптеры, позволяющие транслировать фирменный интерфейс приложения в SOAP-интерфейс.

Аналогично Oracle 10g, платформа SAP предлагает два репозитория метаданных об интеграционных связях – один для разработки (называемый Repository) и один для развертывания (Directory). Это позволяет вести разработку и тестирование в условиях, максимально близких к штатно функционирующей системе. Кроме того, SAP поставляет уже заполненный Repository для своих собственных приложений.

Из особенностей платформы стоит отметить, что интеграция с .NET и WebSphere не ограничивается уровнем коннектора к шине обмена данными, а распространяется также на более высокие слои: совместимыми оказываются и системы управления контентом, и портал, и даже среды разработки. SAP позволяет использовать среду разработки Eclipse, IBM WebSphere WSAP поддерживает среду выполнения SAP Web Application Server через SAP Java Connector, а Visual Studio .NET поддерживает SAP WAS через SAP .NET Connector.

## 6.6. ИВК "Юпитер"

ИВК "Юпитер" – российский продукт, обеспечивающий функции интеграции на уровне данных и обмена сообщениями. Он разрабатывался для нужд государственных структур, предъявляющих особые требования к защищенности информации и возможностям интеграции унаследованных систем – в первую очередь мэйнфреймов. Но в равной степени ИВК "Юпитер" поддерживает и современные вычислительные платформы. Продукт представляет собой интегрированное средство, сочетающее свойства виртуальной машины, транспортной магистрали.

Архитектурно система состоит из двух больших логических частей: одна обеспечивает стандартную функциональность на изолированном компьютере, а другая – связь разных компьютеров. На каждом компьютере присутствует реализация так называемой унифицированной модели вычислительного процесса: она включает среду исполнения ИВК "Юпитер" и набор библиотек. Объем функциональности библиотек таков, что можно создавать готовые приложения, опираясь только на API ИВК "Юпитер". Эти приложения автоматически становятся кроссплатформенными. С точки зрения интеграции, важно сокращенное подмножество этого API: в наиболее простом варианте интегрируемое приложение для взаимодействия с ИВК "Юпитер" должно напрямую или через "обертку" (wrapper) поддерживать всего три вызова.

Серьезных конкурентных преимуществ у ИВК "Юпитер" несколько. Главное из них то, что особое внимание в нем уделено вопросам безопасности. Это единственный продукт, имеющий полноценный набор сертификатов от государства, подтверждающих возможность его применения в системах, где обрабатывается государственная тайна. Все транзитные хранилища, а также каналы передачи сообщений, защищены. Продукт обеспечивает контроль целостности вычислительного процесса: в начале загрузки среды исполнения ИВК "Юпитер" сканирует все приложения, установленные на ПК, на предмет того, разрешено ли им здесь находиться и не изменена ли их версия. Он также предлагает средства настройки параметров безопасности, выраженные в терминах нормативно-регламентной документации, а не ОС. Здесь, например, стоит упомянуть об удобной для документооборота ИС в организации системе адресации узлов: она иерархическая, а наименования привязаны к должностям и подразделениям. При этом не требуется сервер адресов, необходимый во многих других системах МОМ.

Также важным свойством, отсутствующим в других продуктах, является встроенная возможность эмуляции IP поверх многих унаследованных транспортных протоколов, что дает возможность гарантированной доставки сообщений в гетерогенной сети (гетерогенные транспортные протоколы поддерживает лишь IBM WebSphere MQ). Помимо гарантированной доставки имеются и встроенные средства контроля гарантированной обработки. В принципе, ИВК "Юпитер" предлагает весь традиционный функционал средств МОМ (расширенный возможностями документооборота).

К недостаткам платформы можно отнести следующие. Скуден список и невелика функциональность специализированных графических инструментов,

мало число адаптеров для ее подключения к существующим приложениям: не поддерживаются "из коробки" многие современные ИТ-технологии (например XML, web-сервисы и Java). Вендор предлагает только коннекторы для программ DOS и мэйнфреймов. В случае мэйнфреймов, однако коннектор способен не просто сканировать экран, но и при наличии у мэйнфрейм-приложения прикладного интерфейса включаться в управление структурами данных.

Для множества проектов средства ИВК "Юпитер" будут весьма полезными, особенно если учесть его низкую цену. Более того, благодаря таким функциям, как сертифицированные средства защиты трафика и слежения за безопасностью, его можно использовать вместе с западными серверами приложений для построения защищенных web-ориентированных приложений на Java. В проектах, где могут потребоваться эти качества, придется смириться с относительной закрытостью платформы и вытекающими отсюда минусами – высокими требованиями к квалификации разработчиков и потенциальными проблемами в модификации создаваемого решения.

## 6.7. Контрольные вопросы

1. Перечислите компании, занимающиеся разработкой платформ для сервис-ориентированных систем.
2. Какие средства IBM для этапного внедрения COA вам известны?
3. Чем, на ваш взгляд, обусловлен успех продуктов BEA WebLogic на рынке?
4. Какие из платформ поддерживают концепцию ESB?
5. Проведите сравнительный анализ пакетов интеграции компании "Oracle".
6. Перечислите продукты различных компаний, предоставляющие поддержку технологиям EAI, EII, ETL.
7. Что такое унаследованные транспортные протоколы?
8. Что такое middleware?
9. Проведите сравнительный анализ продукции для сервис-ориентированной разработки среди предложений разных компаний.
10. Продукты какой компании имеют максимальное количество адаптеров к данным?
11. Какая COA-платформа имеет, на ваш взгляд, оптимальное соотношение цена/качество.
12. Какие продукты российских производителей имеются на рынке COA-платформ?
13. В чем основные конкурентные преимущества ИВК "Юпитер" перед зарубежными аналогами?

### Список литературы


1. *Олифер Н. А., Олифер В. Г.* Сетевые операционные системы. — СПб: Питер, 2005. — 544 с.
2. *Кузнецов Д. С.* Основы баз данных: Курс лекций. — М.: Изд-во Интернет-ун-т информ. технол., 2005. — 488 с.
3. *Лодыженский Г. М.* Системы баз данных. Коротко о главном // *СУБД*. — 1995. — № 1-4.
4. *Таненбаум Э., Стеен М.* Распределенные системы. Принципы и парадигмы. — СПб: Питер, 2003. — 877 с.

5. *Lamport L.* Time, clocks and the ordering of events in a distributed system // *Communications of the ACM*. — 1978. — Vol. 21, № 7. — P. 558–564.
6. *Dijkstra E. W.* Solution to a problem in concurrent programming // *Communications of the ACM*. — 1965. — Vol. 8, № 9. — P. 327–336.
7. *Hoare C. A. R.* Monitors: An operating system structuring concept // *Communications of the ACM*. — 1974. — Vol. 17, № 10. — P. 327–331
8. *Hale J., Papa M., Shenoi S.* Programmable security for object-oriented systems // IFIP Workshop on Database Security. — 1998. — P. 109–123.
9. *Silberschatz A., Galvin P. B.* Operating System Concepts. — John Willey & Sons, 1995.
10. *Абеков А.* Процессный подход в бизнесе // *Рынок ценных бумаг Казахстана*. — 2004. — № 5. — С. 12–18.
11. *Саймон Г.* Менеджмент в организациях / Пер. с англ. — М.: Экономика, 1995. — 335 с.
12. *Саймон Г., Хаммер М., Чампи Д.* Реинжиниринг корпорации: Манифест революции в бизнесе / Пер. с англ. — СПб.: Изд-во С.-Петербург. гос. ун-та, 1997. — 327 с.
13. *Porter M.* Competitive Strategy. — New-York: Free Press, 1980.
14. *Porter M.* Competitive Advantage. — New-York: Free Press, 1985.
15. *Черняк Л.* Soa – шаг за горизонт // *Открытые системы*. — 2003. — № 9. — С. 24–26.
16. *Дубова Н.* Soa: подходы к реализации // *Открытые системы*. — 2004. — № 6. — С. 14–17.
17. *Галкин Г.* Интеграция приложений: история подходов // *Сети*. — 2002. — № 6. — С. 19–22.
18. *Fielding R. T.* Architectural Styles and the Design of Network-based Software Architectures / Ph. D. Dissertation. — Irvine: University Of California, 2000.
19. *Андреев И.* Использование протокола SOAP в распределенных приложениях // *RSDN Magazine*. — 2003. — № 1. — С. 5–8.

## ТЕРМИНОЛОГИЧЕСКИЙ СЛОВАРЬ

**Апплет** — небольшое приложение на Java, предоставляемое сервером web. В отличие от полномасштабных программ на Java апплет имеет встроенную систему защиты.

**Интранет** — закрытая частная сеть компании или организации, в которой используются стандартные протоколы Internet, такие, как TCP/IP, в качестве транспортного механизма, серверы HTTP для представления документов и серверы SMTP и POP для почты.

**Процесс** — совокупность набора исполняющихся команд, ассоциированных с ним ресурсов (выделенные для исполнения память или адресное пространство, стеки, используемые файлы, устройства ввода–вывода и т. д.) и текущего момента его выполнения (значения регистров, программного счетчика, состояние стека и значения переменных), находящихся под управлением операционной системы.

**Сервер** — аппаратный или программный компонент вычислительной системы, выполняющий специализированные функции по запросу клиента,

предоставляя ему доступ к определённым ресурсам. Одна из важнейших составляющих концепции "клиент–сервер".

**Сервис** — совокупность средств для обслуживания пользователей; набор функций одного из уровней программной структуры сети, обеспечивающих доступ к объектам вышележащего уровня через интерфейс между этими уровнями.

**Сервис-ориентированная архитектура (СОА)** — тенденция в развитии распределенных информационных систем, направленном на высокоуровневое манипулирование автономными и композитными сервисами, позволяющими организовать построение приложений управления бизнес-процессами в гетерогенной среде.

**Транзакция** – группа операций, которая может быть выполнена либо полностью успешно, с соблюдением целостности данных и независимо от параллельно идущих других транзакций, либо не выполнена вообще – тогда она не должна произвести никакого эффекта. Различают обычные и распределённые транзакции. Распределённые транзакции подразумевают использование больше чем одного контейнера данных и требуют намного более сложной логики (например, two-phase commit – двухфазный протокол подтверждения успеха). Обычно транзакции базируются на свойствах ACID (ACID – Atomicity (атомарность), Consistency (непротиворечивость), Isolation (изоляция), Durability (долговечность)) и наиболее распространены в СУБД. В идеале транзакции разных пользователей должны выполняться так, чтобы создавалась иллюзия, что пользователь текущей транзакции – единственный.

**.NET** – программная технология, предложенная фирмой "Microsoft" в качестве платформы для создания как обычных программ, так и web-приложений. Во многом является развитием идей и принципов, заложенных в технологии Java. Одной из основных идей .NET является совместимость различных служб, написанных на разных языках. Служба, написанная на C++ для .NET, может обратиться к методу класса из библиотеки, написанной на Delphi; на C# можно написать класс, наследующий от класса, написанного на Visual Basic .NET, а исключение, выброшенное методом, написанным на C#, может быть поймано и обработано в Delphi. Каждая библиотека (сборка) в .NET имеет сведения о своей версии, что позволяет устранить возможные конфликты между разными версиями сборок. Кроссплатформенная технология, однако в настоящее время существуют реализации для платформ Microsoft Windows, FreeBSD (от Microsoft) и ограниченный вариант технологии для ОС Linux.

**ActiveX** — программные компоненты OCX (OLE Control Extension), совместимые с DCOM. При включении в документы HTML компоненты ActiveX загружаются и выполняются на клиенте. В настоящее время ActiveX работают только в среде Windows. Объекты ActiveX не поддерживают модель защиты, аналогичную апплетам.

**Business Process Execution Language for Web Services (BPEL4WS)** — язык описания бизнес-процессов. Объединяет возможности языка WSFL (Web services flow language, язык организации потоков web-сервисов), разработанного компанией IBM, и языка XLANG, используемого в Microsoft BizTalk Server 2002. BPEL поддерживает описание графоориентированных процессов, а XLANG – структурные конструкции для процессов. Предназначен для поддержки реализации бизнес-процессов любой сложности, а также для описания интерфейсов бизнес-процессов.

**Common Gateway Interface (CGI)** — стандарт интерфейса (связи) внешней прикладной программы с информационным сервером типа HTTP, web-сервер. Обычно гипертекстовые документы, извлекаемые из web-серверов, содержат статические данные. С помощью CGI можно создавать CGI-программы, называемые шлюзами, которые во взаимодействии с такими прикладными системами, как система управления базой данных, электронная таблица, деловая графика и др., смогут выдать на экран пользователя динамическую информацию. Программа-шлюз запускается web-сервером в реальном масштабе времени, и web-сервер обеспечивает передачу запроса пользователя шлюзу. Программа-шлюз, используя средства прикладной системы, возвращает результат обработки запроса на экран пользователя.

**Component Object Model COM** – компонентная модель объектов. Технологический стандарт компании "Microsoft", предназначенный для создания программного обеспечения на основе взаимодействующих распределённых компонентов, каждый из которых может использоваться во многих программах одновременно. Технология воплощает в себе идеи полиморфизма и инкапсуляции объектно-ориентированного программирования. Является универсальной и платформо-независимой, но закрепилась в основном на операционных системах семейства Windows. В современных версиях Windows COM используется очень широко. На основе COM создано множество других стандартов: OLE Automation, ActiveX, COM+, DCOM.

**Distributed Computing Environment (DCE)** — среда распределенных вычислений. Разработана OSF. Представляет собой API для программиро-

вания распределенных вычислений. Поддерживает контекты вызовов. Имеет возможность прозрачной обработки RPC.

**Enterprise Resource Planning (ERP)** — система планирования ресурсов предприятия и управления ими, необходимых для осуществления продаж, закупок и учета при выполнении заказов клиентов в сферах производства, дистрибьюции и оказания услуг.

**Enterprise Service Bus (ESB)** — концепция организации единого логического канала интеграции сервисов в рамках сервис-ориентированной парадигмы для минимизации затрат на разработку интерфейсов межсервисного взаимодействия.

**Extensible Markup Language (XML)** — расширяемый язык разметки. Рекомендован Консорциумом Интернет. Фактически представляет собой свод общих синтаксических правил. Предназначен для хранения структурированных данных (взамен существующих файлов баз данных), для обмена информацией между программами, а также для создания на его основе более специализированных языков разметки (например, XHTML). Является упрощенным подмножеством языка SGML. Целью создания XML было обеспечение совместимости при передаче структурированных данных между разными системами обработки информации, особенно при передаче таких данных через Интернет. Языки, основанные на XML (например, RDF, RSS, MathML, XHTML, SVG), сами по себе формально описаны, что позволяет программно изменять и проверять документы на этих языках, не зная их семантики, т. е. не зная смыслового значения элементов языка. Важной особенностью XML является применение так называемых пространств имён.

**Hypertext Markup Language (HTML)** — стандартный язык разметки гипертекста, используемый при создании страниц Web. Базируется на Standard Generalised Markup Language (SGML) компании IBM.

**Hypertext Transfer Protocol (HTTP)** — базовый протокол, описывающий взаимодействие между браузерами и серверами web (точнее, серверами HTTP). При запросе страницы web пользователь в действительности посылает команду "get" серверу HTTP. Предназначен для одновременного обслуживания максимального числа пользователей. Создает соединения без запоминания адресной информации. Стандарт контролируется IETF.

**IBM Web Services Flow Language (WSFL)** — стандарт для описания базовых взаимодействий web-служб. С его помощью описывается распределение функций между web-службами, позволяющее добиться стоящих перед

потребителями целей. Ориентирован на создание бизнес-приложений объединением повторно используемых компонентов.

**Interface Definition Language (IDL)** — описательный язык, синтаксически похожий на C++; разработан OMG для описания интерфейсов распределенных объектов – названий методов и типов переменных-аргументов.

**Internet Engineering Task Force (IETF)** — независимая организация по стандартизации Интернет. Производители, отдельные организации и другие заинтересованные лица подают свои предложения, которые затем одобряются или объединяются с другими предложениями в виде новых стандартов общего пользования. Например, HTTP является стандартом IETF.

**Internet InterORB Protocol (IIOP)** — спецификация совместимости различных ORB в технологии CORBA. В теории IIOP-совместимый CORBA ORB должен взаимодействовать с любым другим IIOP-совместимым CORBA ORB.

**Internet Services Application Programming Interface (ISAPI)** — программный интерфейс Интернет-сервисов. Разработан компаниями "Microsoft" и "Process Software" для выполнения приложений на сервере. В настоящее время существует только для Windows NT Server. Использует 32-разрядные вызовы Windows вместо CGI. Конкурирует с NSAPI производства Netscape.

**Java** – объектно-ориентированный язык программирования, разрабатываемый компанией "Sun Microsystems" с 1991 г. и официально выпущенный 23 мая 1995 г. Изначально новый язык программирования назывался Oak и разрабатывался для бытовой электроники, но впоследствии был переименован в Java и стал использоваться для написания апплетов, приложений и серверного программного обеспечения. Программы на Java могут быть транслированы в особый байт-код, выполняемый на виртуальной машине (JVM) – программе, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор, но с тем отличием, что байтовый по сравнению с текстом обрабатывается значительно быстрее.

**Java Beans** — компонентная объектная модель для Java, разработанная компанией "JavaSoft". Представляет собой плаформенно-независимый набор интерфейсов API, с помощью которых объекты Java могут включаться в ActiveX (или COM), CORBA или другие объектные модели.

**JavaScript** — язык сценариев, созданный компанией "Netscape" для автоматизации web-страниц. Изначально именовавался LiveScript. Код JavaScript включается в документы на HTML. Конкурирует с VBScript. Существует польшое количество интернет-приложений, разработанных на JavaScript.

**Java Database Connectivity (JDBC)** — набор API, предназначенных для доступа к базе данных посредством Java. Опирается на спецификации X/Open. Первые драйверы JDBC использовали драйверы ODBC. Последнее время все более широкое распространение получают драйверы, не использующие ODBC.

**Multipurpose Internet Mail Extensions (MIME)** — стандарт описывающий способы пересылки по электронной почте исполняемых, графических, мультимедийных и смешаных данных. Типичные применения MIME – пересылка графических изображений, аудио, документов Word, программ текстовых файлов. Позволяет размечать письмо на части различных типов так, чтобы получатель (почтовая программа) мог определить, что делать с каждой из частей письма.

**Netscape Application Programming Interface (NSAPI)** — набор API, разработанных компанией "Netscape", с помощью которых страницы HTML могут выполнять программы на сервере. Программы NSAPI более разнообразны, чем сценарии или программы CGI. NSAPI конкурирует с ISAPI компании "Microsoft".

**Oracle Busines Models (OBM)** — модель представления любого бизнес-процесса, предложенная компанией "Oracle", для решения задач декомпозиции бизнеса при переходе к СОА.

**Remote Method Invocation (RMI)** — API для платформы Java, при помощи которого объект может удаленно (через сеть) вызывать методы другого объекта.

**RosettaNet Implementation Framework (RNIF)** — спецификация, описывающая основанные на XML протоколы обмена между серверами торговых партнеров включая транспортный уровень, маршрутизацию и упаковку, безопасность, сигналы и специальные соглашения.

**Sandbox** (с *англ.* – "ящик с песком", "песочница") — модель защиты для исполнения Интернет-приложений. запрещает доступ распределенных объектов к вызовам операционной системы или другим системным ресурсам.

**Secure Sockets Layer (SSL)** — протокол, защищающий данные, пересылаемые между web-браузерами и web-серверами. Предоставляет возможность проверки того, что данные, получаемые с узла web, приходят именно с этого узла и во время передачи они не были искажены. Чаще всего применяется для защищенного обмена данными между web-браузерами и web-серверами.

**Simple Object Access Protocol (SOAP)** — протокол обмена структурированными сообщениями в распределённой вычислительной среде. Первоначально SOAP предназначался в основном для реализации удалённого вызова процедур (RPC). Изначально его название было аббревиатурой: Simple Object Access Protocol (простой протокол доступа к объектам). Сейчас используется для обмена произвольными сообщениями в формате XML, а не только для вызова процедур. Официальная спецификация последней версии 1.2 протокола никак не расшифровывает название SOAP. SOAP является расширением языка XML-RPC. Может использоваться с любым протоколом прикладного уровня: SMTP, FTP, HTTP и др. Однако его взаимодействие с каждым из этих протоколов имеет свои особенности, которые должны быть определены отдельно. Чаще всего SOAP используется поверх HTTP для создания web-сервисов.

**Practical Extraction and Report Language (Perl)** — интерпретируемый язык общего назначения. Используется как простой язык программирования для написания простых сценариев CGI; эти сценарии вызываются страницами HTML и выполняются сервером. Ввиду медленного исполнения интерпретируемых команд сложные программы пишутся, как правило, на компилируемых языках, а не на Perl.

**Standard Generalized Markup Language (SGML)** — метаязык, на котором можно определять язык разметки для документов. Наследник разработанного в 1960 г. в IBM языка GML (Generalized Markup Language). Изначально разработан для возможности совместного использования машинно-читаемых документов в больших правительственных проектах, а также в авиационно-космических областях. Широко использовался в печатной и издательской сферах, но его сложность затруднила его широкое распространение для повседневного использования. Основные части документа SGML: SGML-декларация (SGML Declaration); объявление типа документа; содержимое SGML-документа (обязательно наличие корневого элемента). Предоставляет множество вариантов синтаксической разметки для использования различными приложениями.

**Structured Query Language (SQL)** — универсальный язык, применяемый для создания, управления и модификации реляционных баз данных, а также управления данными, хранящимися в них. SQL основывается на реляционной алгебре. Язык SQL делится на три части: операторы определения данных (Data Definition Language, DDL); операторы манипуляции данными (Data Manipulation Language, DML); операторы определения доступа к данным (Data

Control Language, DCL). Большинство серверов баз данных, существующих в настоящее время, поддерживают SQL.

**Universal Description, Discovery and Integration (UDDI)** — набор спецификаций для универсального именования ресурсов в распределенных системах и для организации поиска сервисов при построении композитных приложений.

**Web-Сервис** – программная система, идентифицируемая строкой URI, чьи публичные интерфейсы и привязки определены и описаны языком XML. Описание данной программной системы может быть найдено другими программными системами, которые могут взаимодействовать с ней согласно этому описанию посредством сообщений, основанных на XML и передаваемых с помощью Интернет-протоколов.

**Web Service Choreography Interface (WSCI)** — проект стандарта, описывающего поведение web-службы с использованием интерфейса, ориентированного на потоки сообщений. WSCI должен обеспечить разработчикам высокоуровневое представление о том, как взаимодействуют web-службы, создавая единое более крупное приложение.

**Workflow Management Coalition (Wf-XML)** — некоммерческая организация, которая ставит своей целью расширение возможностей использования технологий workflow (потоков работ) путем разработки единой терминологии и стандартов.

# Предметный указатель

.Net, 86

AS/2, 86, 91

B2B, 86
BPEL, 58
BPEL4WS, 48
BTS2004, 86
Business Integration, 89
Business Intelligence, 66, 89

CDR, 69
CDS, 69
CGI, 9
CICS, 91
CORBA, 10, 43
COS naming, 69
CRM, 51

Data warehouse, 62
DB2, VSAM, 91
DCE, 43, 69
DCE IDL, 69
DCOM, 10, 43

ebXML, 74
EDI, 91
EDIFACT EDI, 91
EJB, 10
Enterprise Service Bus, 51
ERP, 51
ESB, 53

GIOP, 69

HTML, SMTP, 69

Identity Management, 89
IDL, 48
IIOP, 69
IMS DB, 91
IMS TM, 91
ISO, 38
ISO 9000:2000, 38

Java, 9
JAX-RPC, .Net, 69
JCA, 64
JMS, 64
JRMP, 69

Middleware, 64, 83
MIME, 91
MOM, 82

NDR, 69
NUMA, 32

OBM, 40
ODBC, 62
ORB, 63

Paging, 32
PCB, 19
PDU, 69
PeopleSoft, 91
PIP, 91
PIP BD, 91

QoS, 52, 53

recieve, 21
RFID, 89
RMI, 43, 64
RNIF, 91
RosettaNet, 90
RPC, 63, 69
RPC CO, 69

S/MIME, 91
SAP, 91, 92
SAP XI, 93
SEQUEL, 6
Siebel, 91
SLA, 52
SOAP, 69, 74
SQL, 6, 62
SSL, 86
Swapping, 32
System R, 6

Tivoli, 59

UCCnet, 86
UDDI, 69
UMA, 32
URI, 45

Web-Сервис, 10
WSAP, 93
WSCL, 74

WSD, 72
WSDL, 48, 69
WSFL, 48, 74

X12, 91
X12-HIPAA, 91
XA, 92
XA/JTA, 92
XLANG, 48, 73
XML, 48, 69
XML Schema, 48

Алгоритм Лампорта, 23
Архитектура
  "клиент–сервер", 6
  двузвенная, 6
  многозвенная, 7
  монолитная, 4

Балансировка нагрузки, 31
БД
  распределенная, 7
Бизнес-правило, 7
Бизнес-процесс, 50

Дейкстра, 25
Децентрализация
  алгоритмов, 14
  данных, 14
  служб, 14

Контекст
  пользовательский, 20
  регистровый, 19
  системный, 19
Координатный коммутатор, 33

Логические часы, 22

Маршрутизация чередующаяся, 34
Масштабируемость, 13
Матрица доступа, 36
Менеджер
  авторизации, 9
  блокировок, 9
  журнала, 9
  коммуникационный, 9
  транзакций, 9
Метрика, 40
Механизм транзакций, 25
Монитор, 26
  Хоара, 26
Мультипрограммирование, 20

Неделимость, 24
Непротиворечивость, 24

Оркестровка, 42
Отказоустойчивость, 15
Открытость, 13
Относительное время, 22

Планирование
  бригадное, 30
  двухуровневое, 29
  потоков, 21
Позднее связывание, 47
Портер, 41
Посредник сервиса, 45
Поставщик сервиса, 45
Постоянство, 24
Поток, 20
Потребитель сервиса, 45
Примитивы транзакций, 24
Прозрачность, 13
Процесс, 17

Разделение пространства, 30

Саймон, 41
Семафор, 25
Сервис-ориентированная архитектура, 10
Сервисная шина предприятия, 58
Синхронизация, 21
Система
  распределенная, 12
Событийно-управляемая архитектура, 11
Согласование транзакций, 7

Тиражирование, 7
Транзакции, 24
Транзакция
  вложенная, 25
  плоская, 25
  распределенная, 25

Упорядочиваемость, 24
Уровни адаптации СОА, 56

Физические часы, 22

Хаммер, 41

# Оглавление


ВВЕДЕНИЕ . . . . . 3
1. ТЕНДЕНЦИИ РАЗВИТИЯ РАСПРЕДЕЛЕННЫХ СИСТЕМ . . . . . 4
1.1. Монолитные системы . . . . . 4
1.2. Архитектуры "клиент–сервер" . . . . . 6
1.2.1. Двузвенная архитектура . . . . . 6
1.2.2. Многозвенные архитектуры . . . . . 7
1.3. Системы, ориентированные на сервисы . . . . . 10
1.3.1. Web-Сервисы . . . . . 10
1.3.2. Сервис-ориентированная архитектура . . . . . 10
1.3.3. Событийно-управляемая архитектура . . . . . 11
1.4. Контрольные вопросы . . . . . 12
2. РАСПРЕДЕЛЕННЫЕ СИСТЕМЫ. ПРИНЦИПЫ ОРГАНИЗАЦИИ . . . . . 12
2.1. Свойства распределенных систем . . . . . 12
2.1.1. Прозрачность . . . . . 13
2.1.2. Открытость . . . . . 13
2.1.3. Масштабируемость . . . . . 13
2.1.4. Безопасность . . . . . 15
2.1.5. Аспекты разработки распределенных систем . . . . . 16
2.2. Асинхронная обработка информации . . . . . 17
2.2.1. Процессы . . . . . 17
2.2.2. Потоки . . . . . 20
2.2.3. Методы синхронизации . . . . . 21
2.2.4. Синхронизация логических часов . . . . . 22
2.2.5. Транзакции . . . . . 24
2.2.6. Механизмы синхронизации . . . . . 25
2.2.7. Методы диспетчеризации . . . . . 28
2.2.8. Двухуровневое планирование . . . . . 29
2.2.9. Разделение пространства . . . . . 30
2.2.10. Бригадное планирование . . . . . 30
2.2.11. Планирование многомашинных систем . . . . . 31
2.3. Память в распределенных системах . . . . . 32
2.4. Механизмы защиты . . . . . 35
2.5. Контрольные вопросы . . . . . 37
3. СЕРВИС-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА . . . . . 38
3.1. Модели, основанные на процессном подходе . . . . . 38
3.1.1. Oracle Business Models . . . . . 40
3.1.2. Универсальные 13- и 8-процессные модели . . . . . 41
3.1.3. Модель Портера . . . . . 41
3.2. Архитектура, ориентированная на сервисы . . . . . 42
3.2.1. Определения СОА . . . . . 43
3.2.2. Базовая модель взаимодействия . . . . . 45
3.2.3. Вертикальная архитектура СОА-приложений . . . . . 50
3.3. Сервис-ориентированное моделирование и анализ . . . . . 53
3.3.1. Идентификация сервиса . . . . . 53
3.3.2. Классификация сервиса . . . . . 54
3.3.3. Анализ подсистемы . . . . . 54

3.3.4. Спецификация компонента . . . . . 55
3.3.5. Размещение сервиса . . . . . 55
3.3.6. Реализация сервиса . . . . . 56
3.4. Уровни адаптации СОА . . . . . 56
3.5. Контрольные вопросы . . . . . 60
4. ИНТЕГРАЦИЯ СЕРВИСОВ И ПРИЛОЖЕНИЙ . . . . . 60
4.1. История подходов к интеграции . . . . . 60
4.1.1. Ранние подходы . . . . . 61
4.1.2. Интеграция на уровне данных . . . . . 62
4.1.3. Интеграция на уровне корпоративных приложений . . . . . 63
4.2. Web-Услуги . . . . . 64
4.3. Контрольные вопросы . . . . . 67
5. СТАНДАРТЫ СОА . . . . . 68
5.1. Классификация соглашений и стандартов . . . . . 69
5.2. Интерфейс управления хореографией . . . . . 77
5.3. Язык исполнения бизнес-процессов BPEL4WS . . . . . 79
5.4. Контрольные вопросы . . . . . 81
6. ПРОГРАММНЫЕ СИСТЕМЫ И СРЕДЫ . . . . . 81
6.1. BEA WebLogic . . . . . 81
6.2. IBM WebSphere . . . . . 83
6.2.1. Компоновки IBM WebSphere . . . . . 85
6.3. Microsoft .NET/BizTalk . . . . . 86
6.4. Oracle 10g . . . . . 88
6.5. SAP NetWeaver . . . . . 92
6.6. ИВК "Юпитер" . . . . . 93
6.7. Контрольные вопросы . . . . . 95
СПИСОК ЛИТЕРАТУРЫ . . . . . 95
ТЕРМИНОЛОГИЧЕСКИЙ СЛОВАРЬ . . . . . 96
ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ . . . . . 103

Гладцын Вадим Андреевич
Кринкин Кирилл Владимирович
Яновский Владислав Васильевич

**СЕРВИС-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА**
**Стандарты, алгоритмы, протоколы**

Учебное пособие

Редактор И. Б. Синишева

Подписано в печать 06.07.06. Формат 60x84 1/16. Бумага офсетная.
Печать офсетная. Гарнитура "Times". Печ. л. 6,75.
Тираж 350 экз. Заказ

Издательство СПбГЭТУ "ЛЭТИ"
197376, С.-Петербург, ул. Проф. Попова, 5