Федеральное государственное образовательное бюджетное
учреждение высшего профессионального образования

«ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»

Кафедра бизнес-информатики

О.А. Морозова

ИНТЕГРАЦИЯ КОРПОРАТИВНЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

Учебное пособие

Москва 2014

УДК 004(075.8)
ББК 32.973я73
М80

Рецензенты:
доктор экономических наук, профессор,
декан факультета экономики и менеджмента ***Е.Д. Коршунова***
(Московский государственный технологический университет «СТАНКИН»);
доктор физико-математических наук,
профессор кафедры информатики и программирования ***Г.Б. Рубальский***
(Финансовый университет при Правительстве Российской Федерации)

Морозова О.А. Интеграция корпоративных информационных систем:
М80 учебное пособие. — М.: Финансовый университет, 2014. — 140 с.

ISBN 978-5-7942-1135-1

Рассматриваются современные подходы к разработке интеграционных решений. Обосновывается стратегическое значение интеграции информационных систем для преобразования бизнеса, приводится классификация интеграционных задач, формулируются критерии выбора интеграционного решения, проводится обзор базовых технологий и стандартов, используемых при разработке интеграционных решений. Вопросы проектирования интеграционных решений рассматриваются с использованием языка шаблонов, охватывающих все аспекты взаимодействия приложений.

Для студентов высших учебных заведений, обучающихся по направлениям подготовки 080700.68 «Бизнес-информатика» и 230700.68 «Прикладная информатика», квалификация (степень) «магистр».

УДК 004(075.8)
ББК 32.973я73

ISBN 978-5-7942-1135-1

Federal State-Funded Education Institution
of Higher Professional Education

«FINANCIAL UNIVERSITY UNDER THE GOVERNMENT
OF THE RUSSIIFN FEDERATION»

Department of Business Informatics

O.A. Morozova

ENTERPRISE INFORMATION SYSTEMS INTEGRATION

Manual

Moscow 2014

Reviewers:
Korshunova E.D., PhD, Professor, Head of the Faculty «Economics and Management», Moscow State University of Technology «STANKIN»;
Rubalsky G.B., PhD, Professor, Department of Computer Science and Programming, Financial University under the Government of the Russian Federation

Morozova O.A. Enterprise Information Systems Integration: manual. — M.: Financial University, 2014. — 140 p.

ISBN 978-5-7942-1135-1

Modern approaches of integration decisions implementation are considered. Strategic value of legacy systems integration for business transformation is shown. Classification of integration tasks is carried out, criteria of integration decision choice are formulated. Contains the review of basic technologies and the standards used at development of integration decisions. Questions of design of integration decisions are considered with use of language of the templates covering all aspects of interaction.

The manual can be recommended to students of the higher educational institutions which are training in the directions of preparation of masters of 080700.68 «Business informatics» and 230700.68 «Applied informatics».

ISBN 978-5-7942-1135-1

Введение

Информационная инфраструктура предприятия претерпевает постоянные изменения. Этот процесс может происходить как целенаправленно, в рамках определенной ИТ-стратегии, так и стихийно, под влиянием насущных потребностей бизнеса. Результатом является крайне неоднородный ИТ-ландшафт, содержащий приложения и программные компоненты от разных производителей, которые реализованы на разных платформах и зачастую дублируют отдельные функции. Ситуацию усугубляют процессы слияния и поглощения компаний, приводящие к наследованию новых информационных систем и приложений.

В современных условиях возрастает зависимость бизнеса от информационных технологий, причем для его успешного развития важен не столько набор приложений, автоматизирующих отдельные функции или бизнес-процессы, сколько интеграция и взаимосогласованность информационных систем и приложений. Следует выделить три аспекта, которые обусловливают исключительную актуальность проблемы интеграции.

Во-первых, повсеместное использование информационных технологий приводит к лавинообразному росту объемов цифровой информации. По оценкам компании IDC [5], ежегодно объем данных увеличивается более чем на 60%. Эта информация распределена по многочисленным гетерогенным приложениям, системам, настольным базам данных и мобильным устройствам. Для эффективного использования распределенных данных должны быть решены проблемы сбора, синхронизации и использования релевантной информации в масштабах предприятия. Речь, прежде всего, идет об обеспечении

качества данных, управлении данными и своевременной доставке информации потребителю.

Во-вторых, необходимо поддерживать сквозные бизнес-процессы, охватывающие различные подразделения компании и внешних контрагентов. На сегодняшний день очевидны необходимость интеграции приложений для поддержания таких стратегических направлений бизнеса, как электронная коммерция, управление цепями поставок, управление взаимоотношениями с клиентами (CRM, Customer Relationship Management), а также необходимость обмена информацией и сервисами между информационными системами в пределах предприятия.

В-третьих, требования бизнеса всегда опережают возможности информационных технологий, поэтому чрезвычайно важно уметь использовать функционал унаследованных систем для поддержки преобразований бизнеса и сохранять инвестиции в информационные технологии.

Интеграционные проекты достаточно дорогостоящи. По подсчетам компании Gartner Group, 80% ИТ-бюджетов компаний составляют расходы на сопровождение систем, из них 35% — затраты на интеграцию приложений, 60% стоимости внедрения корпоративной информационной системы составляют расходы на интеграцию.

Таким образом, интеграция разнородных приложений и систем становится все более и более значимой проблемой развития ИТ-инфраструктуры. Условие успешного развития любой компании на современном этапе — создание ИТ-инфраструктуры, в которой интегрированы все ее вычислительные, информационные и коммуникационные ресурсы.

Задача выбора интеграционной стратегии, соответствующей потребностям бизнеса, нетривиальна несмотря на то, что современный уровень развития информационных технологий позволяет успешно преодолевать технологические трудности связывания приложений.

Данное учебное пособие как раз и посвящено вопросам интеграции корпоративных информационных систем.

В *первой главе* рассматриваются общие вопросы, связанные с постановкой задачи интеграции и обоснованием ее стратегической ценности для бизнеса, вводятся понятия вертикальной и горизонтальной интеграции, формулируются критерии выбора интеграционного решения.

Во *второй главе* проводится обзор технологий и стандартов, используемых при разработке интеграционных решений; дается определение промежуточной среды и рассматриваются базовые модели взаимодействия удаленных компонентов; проводится сравнительный анализ стандартов объектно-ориентированного взаимодействия. Большое внимание уделено технологиям интеграции, базирующимся на XML, в том числе технологиям Web-сервисов. Приводится краткий, проиллюстрированный примерами, обзор спецификаций XML, DTD, XML-Schema, XSLT, WSDL, WS-BPEL, WS-CDL.

Третья глава посвящена вопросам проектирования интеграционных решений с использованием языка шаблонов, представляющих собой абстрактное описание типовых задач и способов их решения. Рассматриваются шаблоны архитектуры промежуточного слоя, шаблоны связывания приложений, а также шаблоны топологии.

Глава 1

Интеграция корпоративных информационных систем как средство развития бизнеса

1.1. Эволюция подходов к интеграции информационных систем

Концепция интеграции информационных систем далеко не нова. Необходимость интеграции стала очевидной, как только на предприятиях появилось более одной информационной системы и локальная сеть.

В процессе развития информационных технологий подходы к решению задачи интеграции менялись. В 1970–80-е гг. корпоративные приложения выполняли достаточно простые функции, сводящиеся к автоматизации отдельных операций и решению относительно несложных, легко формализуемых задач. Постепенное наращивание функциональности привело к возникновению модульной архитектуры систем. В 1990-е гг. считалось, что единственно правильным направлением комплексной автоматизации является создание универсальной информационной системы, охватывающей все области деятельности предприятия. Такая система должна быть основана на единой программно-аппаратной платформе и общей базе данных, а ее отдельные функциональные подсистемы (модули) — взаимосвязаны на основе единого технологического процесса обработки информации.

Классическим примером модульных информационных систем являются системы класса ERP (Enterprise Resource Planning), включающие модули для управления производством, планированием, договорами, материально-техническим снабжением, финансами, кадрами, сбытом, запасами и т.д. Все модули ERP-системы «интегрированы» в единую комплексную информационную систему компанией-разработчиком. Внедрение ERP-системы, как считалось, решает проблему взаимосвязи приложений и снимает необходимость вкладывать значительные средства в интеграцию. В финансовой отрасли ставка делалась на автоматизацию традиционных задач банковской деятельности (ведение бухгалтерского учета, получение обязательной отчетности, автоматизированное расчетно-кассовое обслуживание клиентов, кредитно-депозитная деятельность) и построение автоматизированной банковской системы (АБС).

АБС также имеют модульный принцип построения, позволяющий легко конфигурировать системы под конкретное предприятие с возможностью последующего наращивания функционала.

Тем не менее подход, направленный на формирование масштабной универсальной системы от одного производителя, как показала практика, не снимает проблему интеграции приложений в силу следующих обстоятельств:

- в условиях развития бизнеса высока потребность в индивидуальных информационных системах. Тиражные решения не всегда позволяют персонифицировать приложения за счет возможностей настройки, к тому же они могут оказаться слишком дорогими, поэтому предприятия используют собственные приложения или приложения других вендоров, реализующие необходимую им функциональность;
- в компаниях всегда остается несколько «устаревших» приложений, замена которых модулями комплексной системы может занять много времени. На время «переходного» периода такие приложения должны быть интегрированы;
- слияния и поглощения компаний являются источником интеграционных проблем: часто в компаниях используются ERP-системы от различных поставщиков, в банках — АБС от разных производителей;
- существует проблема взаимодействия с внешними контрагентами компании;

▪ необходимость автоматизации новых направлений деятельности (электронные карты, электронные торговые площадки, мобильная связь, облачные сервисы и др.).

Одновременно с развитием универсальных информационных систем возник рынок специализированного программного обеспечения, конкурирующего с отдельными модулями универсальных систем. Конкурентное преимущество такого программного обеспечения — специализация, основанная на глубоком знании отдельных процессов и технологий. Постепенно специализированные программы стали неотъемлемой частью ИТ-инфраструктуры большинства предприятий наряду с многочисленными вспомогательными системами, являющимися зачастую продуктами собственной разработки. Возникло множество связей между центральной и вспомогательной системами, а также прямых связей между вспомогательными системами (рис. 1.1).

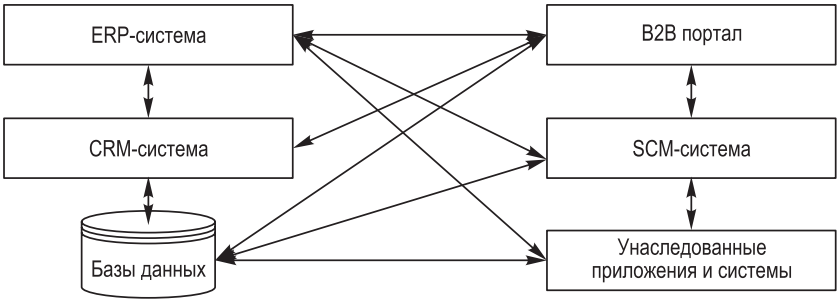

Рис. 1.1. Неупорядоченная ИТ-инфраструктура

На сегодняшний день интегрированные системы постепенно теряют лидерство в ИТ-архитектуре. Этот процесс можно проследить на примере крупных и средних банков, которые отдали предпочтение архитектуре, в которой АБС играет только роль учетной системы, а все основные бизнес-задачи решаются специализированными приложениями, интегрированными между собой.

Принято противопоставлять два крайних подхода к развитию ИТ-инфраструктуры предприятия — это мульти- и моновендорная стратегии [32]. И та и другая стратегия имеет свои достоинства и недостатки.

Преимуществом *моновендорной стратегии* (внедряются продукты одного вендора) принято считать готовую методологию внедрения и проработанные вопросы интеграции и взаимодействия компонентов. Однако платой за «простоту» интеграции может быть множество ненужных функций или дублирование уже имеющейся функциональности, а также излишняя «зависимость» от одного поставщика, ограничивающая свободу выбора программных решений. К тому же продукты одного вендора зачастую создаются разными командами с использованием различных технологий, иногда продукты наследуются крупным вендором вместе с приобретаемой компанией. Для таких продуктов вопросы их интеграции не решены самим вендором.

Мультивендорная стратегия основана на подходе «best-of-bread» (лучшего в своем классе) и требует значительно бóльших затрат на встраивание инородных приложений в сложившуюся архитектуру.

Интеграция между информационными системами строится на основе так называемой трехуровневой модели, поскольку каждое бизнес-приложение можно представить тремя логическими слоями:

1) уровнем представления (уровень пользователя, решающего задачи ввода/вывода информации);

2) уровнем бизнес-логики (обработка данных, поддержка логики бизнес-процессов);

3) уровнем данных (хранение и администрирование данных).

Связь между приложениями может быть установлена на каждом из этих уровней и требует более или менее «инвазивного» вмешательства в тот или иной слой, что усложняет как сам процесс интеграции, так и совместное функционирование систем.

Более универсальный подход к интеграции приложений основан на использовании программного обеспечения класса middleware.

Системы middleware первого поколения предоставляли дополнительный уровень — уровень передачи сообщения. Под *сообщением* понимался структурированный набор параметров, описывающих определенный тип транзакции. Система middleware распознавала формат сообщения, исходящего от одного бизнес-приложения, и трансформировала его в формат принимающего бизнес-приложения. Набор параметров разных типов транзакций и правила переформатирования хранились в базе данных системы middleware.

Принцип работы систем middleware первого поколения иллюстрирует рис. 1.2.

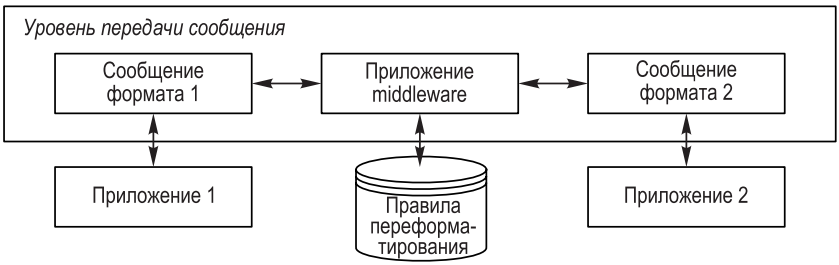

Рис. 1.2. Принцип работы систем middleware первого поколения

Такая «одномерная» модель взаимодействия приложений со временем устарела, и на смену ей пришла технология, основанная на архитектуре ESB (Enterprise Service Bus), которая обеспечивает управляемое взаимодействие между всеми приложениями, подключенными к общей шине предприятия.

Современные промышленные системы класса middleware — это сложное программное обеспечение, способное обрабатывать сообщения на базе универсальных форматов и обеспечивать многоканальную передачу сообщений между всеми бизнес-приложениями.

Основными компонентами таких систем являются:

- *интеграционный брокер*, играющий роль сервисной шины (выполняет функции переформатирования данных, маршрутизации сообщений, управления транзакциями, мониторинга и контроля взаимодействия приложений);
- *набор адаптеров*, которые позволяют различным приложениям подключаться к брокеру.

Принцип работы современных систем middleware иллюстрирует рис. 1.3.

Обзор современных программных продуктов класса middleware будет проведен во второй главе.

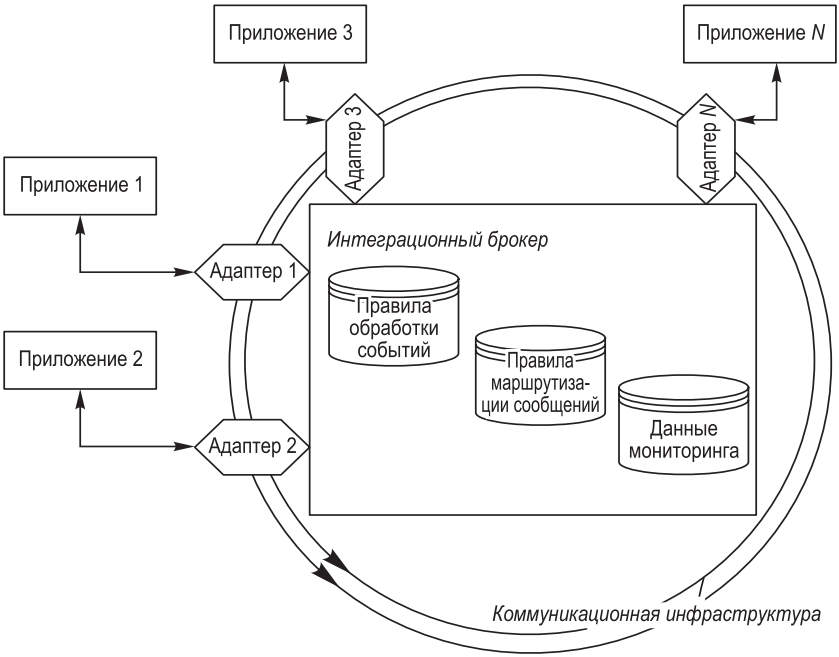

Рис. 1.3. Принцип работы современных систем middleware

1.2. Методология открытых систем и проблема интеграции

В процессе создания интеграционных решений неизбежно возникает проблема стыковки различных программных компонентов. Данная проблема проявляется при расширении систем, включении новых подсистем или новых версий компонентов, тиражировании и повторном использовании прикладных программ, организации обмена данными между приложениями.

Минимизировать затраты на интеграцию и развитие информационных систем позволяет использование методологии открытых систем, которая подразумевает выделение в системе интерфейсной части, обеспечивающей связь с другими системами или подсистемами. Для объединения систем достаточно располагать сведениями только об интерфейсных частях сопрягаемых объектов, выполненных в соответствии с определенными стандартами.

Стандарты, обеспечивающие открытость программного обеспечения, разрабатываются и поддерживаются рядом международных организаций, в частности:

- Совместным техническим комитетом ИСО и МЭК (ISO/IEC/JTC 1) — ИСО/МЭК/СТК 1 «Информационные технологии»;
- Европейской рабочей группой по открытым системам (EWOS);
- Национальным институтом стандартов и технологий США (NIST).

Методология открытых систем обеспечивает экономию инвестиций при построении информационных систем на различных аппаратных и программных платформах за счет совместимости прикладного программного обеспечения и сохранения данных.

Открытые системы — это системы, в которых реализован «исчерпывающий и согласованный набор международных стандартов информационных технологий и профилей функциональных стандартов, которые специфицируют *интерфейсы*, *сервисы* и поддерживаемые *форматы данных*, чтобы обеспечить интероперабельность и мобильность приложений, данных и персонала».

Основными нефункциональными требованиями к открытым системам являются:

- *расширяемость* (*масштабируемость*) — возможность добавлять новые или модернизировать уже имеющиеся функции без изменения остальных функциональных частей информационной системы, а также сохранять работоспособность системы при изменении значений параметров, определяющих технические и ресурсные характеристики системы и/или среды;
- *мобильность* (*переносимость*) — возможность переносить программы и данные при модернизации или замене аппаратных платформ информационной системы, а также использовать опыт людей, пользующихся данной информационной технологией, без их переучивания при изменении информационной системы;
- *интероперабельность* — обеспечение способности к взаимодействию данной информационной системы с другими системами.

Методологию открытых систем определяют следующие документы:

- Технический отчет ISO/IEC TR 10000 Framework and taxonomy of International Standardized Profiles (Основы и таксономия международных стандартизованных профилей);
- Эталонная модель окружения (среды) открытых систем (RM OSE) — ISO/IEC DTR 14252, Portable Operating System

Interface for Computer Environments — POSIX (IEEE, P1003.0, Draft Guide to the POSIX Open System Environment);

▪ Эталонная модель взаимосвязи открытых систем (RM OSI) — ISO 7498: 1996, Information processing systems — Open Systems Interconnection — Basic Reference Model (ITU-T Rec. X.200).

Сущность методологии открытых систем состоит в том, что при их построении стыковка должна обеспечиваться использованием стандартных интерфейсов между всеми компонентами систем.

Выделяют две группы стандартов:

1) стандарты прикладных программных интерфейсов (Application Program Interface (API) Standards), которые определяют взаимодействие прикладного программного обеспечения с компьютерной системой (прикладной платформой) и предназначены в основном для обеспечения переносимости приложений;

2) стандарты внешнего окружения (External Environment Interface (EEI) Standards), которые определяет взаимодействие информационной системы с ее внешним окружением и предназначены для решения проблем интероперабельности систем, повторного использования программного обеспечения и переносимости данных.

Спецификации интерфейсов взаимодействия — это строгие описания всех необходимых функций, служб и форматов определенного интерфейса. Совокупность таких описаний составляет референтную модель открытых систем.

Интерфейсы взаимодействия компонентов систем иллюстрирует рис. 1.4.

Сегодня методология открытых систем поддерживается всеми компаниями — разработчиками программного обеспечения, производителями средств вычислительной техники и средств телекоммуникаций.

Практически все современные информационные системы имеют набор хорошо документированных API для доступа к данным или функциям системы из различных сред программирования.

Рис. 1.4. Интерфейсы взаимодействия компонентов систем

1.3. Цели и задачи интеграции

Интеграция приложений — это стратегический подход к объединению информационных систем, который обеспечивает возможность обмена информацией и поддержания распределенных бизнес-процессов. Интеграция информационных систем дает предприятию такие несомненные конкурентные преимущества, как:

- ведение бизнеса в режиме реального времени с использованием событийно-управляемых сценариев;
- владение достоверной, полной и своевременно полученной информацией.

Задача интеграции — обеспечить эффективный, надежный и безопасный обмен данными между различными программными продуктами, изначально не предназначенными для совместной работы.

Как правило, требования бизнеса эволюционируют быстрее, чем способы их поддержки информационными технологиями.

Основными движущими силами интеграции являются:

- электронный бизнес — интеграция унаследованных информационных систем, поддерживающих ключевую функциональность, с Web-приложениями (Web-сервисами и порталами) с целью получения доступа к бизнес-функциям через Интернет;
- управление цепями поставок — интеграция разрозненных систем управления заказами, MRP-систем, систем календарного планирования, систем транспортного менеджмента с целью прямого обмена информацией между покупателями и поставщиками в режиме реального времени;
- управление взаимоотношениями с клиентами — получение единого консолидированного представления о клиенте путем объединения данных о нем, распределенных между несколькими изолированными приложениями (интеграция клиентских баз данных, call-центров, интернет-сервисов);
- внедрение ERP — интеграция модулей ERP-систем, поддерживающих базовую функциональность, со специализированным программным обеспечением, используемым организацией;
- электронное правительство — интеграция унаследованных back-end систем с front-end Web-приложениями, организация обмена данными между правительственными учреждениями;
- самообслуживание клиентов — возможность клиентов самостоятельно выполнять действия, традиционно являющиеся функцией

обслуживающего персонала, требует интеграции пользовательских приложений с back-end-системами;

- Business Intellegence — сбор данных из различных приложений и источников в хранилище данных с целью их обработки и анализа;
- управление знаниями — обеспечение доступа в режиме реального времени к корпоративному контенту, распределенному между многочисленными источниками, с целью управления знаниями в масштабах предприятия;
- облачные технологии — интеграция существующих бизнес-приложений с облачными приложениями и сервисами;
- аутсорсинг бизнес-процессов — интеграция с информационными системами партнеров.

Перечислим основные бизнес-выгоды, которые предприятие может получить в случае успешной реализации интеграционного проекта:

- улучшение качества поддержки и обслуживания клиентов;
- автоматизация бизнес-процессов;
- уменьшение производственного цикла;
- сокращение количества ошибок обработки данных;
- прозрачность процессов;
- уменьшение стоимости транзакций;
- оптимизация логистических процессов;
- более тесное взаимодействие с бизнес-партнерами;
- быстрое внедрение новых бизнес-сервисов;
- сохранение инвестиций в информационные технологии.

1.4. Типы интеграционных решений: горизонтальная и вертикальная интеграция

Интеграционные решения можно классифицировать разными способами. Например, в зависимости от принадлежности объединяемых приложений выделяют:

- интеграцию корпоративных приложений в пределах предприятия (Application-to-Application Integration — A2A) — автоматический событийно-управляемый обмен информацией между приложениями и системами, действующими на предприятии или в организации;
- интеграцию приложений между предприятиями (Business-to-Business Application Integration — B2B) — автоматический собы-

тийно-управляемый обмен информацией между приложениями или системами нескольких взаимодействующих предприятий или организаций.

Как известно, информационные системы обеспечивают поддержку одного из трех уровней управления: операционного, тактического и стратегического.

В данном контексте существует два варианта построения интеграционного решения:

- горизонтальная интеграция — интеграция информационных систем или приложений, относящихся к одному уровню,
- вертикальная интеграция — интеграция приложений и систем, находящихся на различных уровнях информационной пирамиды.

Типичным примером *горизонтальной интеграции* является автоматизация управления цепями поставок (различные приложения или компоненты обеспечивают полный цикл логистических операций) [12].

На рис. 1.5 показан распределенный бизнес-процесс, поддерживаемый несколькими приложениями.

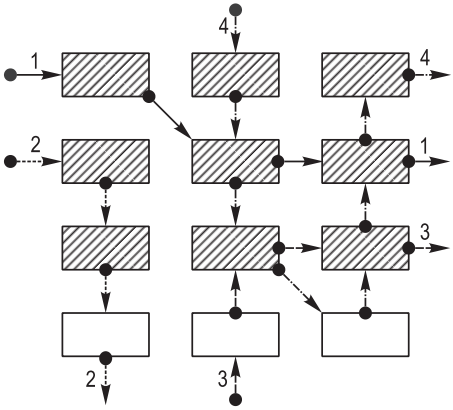

Рис. 1.5. Пример горизонтальной интеграции

Прямоугольниками обозначены приложения, выполняющие отдельные функции, штриховкой — приложения, находящиеся в организации. Процесс 1 является внутренним процессом организации. Процесс 2 начинается в организации и завершается за ее пределами. Процесс 3 начинается за пределами организации, но заканчивается

в организации. Процесс 4 выходит за пределы организации и возвращается в нее перед завершением.

Наиболее часто встречающийся пример *вертикальной интеграции* — сбор данных операционных систем в единое корпоративное хранилище данных с целью их последующего использования для анализа, управления и получения консолидированной отчетности. Рассмотрим интеграционное решение, типичное для многофилиальной территориально распределенной организации (рис. 1.6).

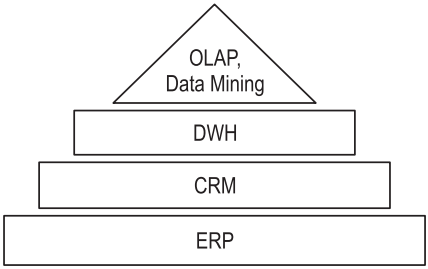

Рис. 1.6. Пример вертикальной интеграции

В хранилище данных консолидируется оперативная бизнес-информация из различных автоматизированных модулей, установленных в офисах и филиалах предприятия, включая территориально удаленные подразделения. Данные поступают в хранилище из разнообразных транзакционных систем, то есть из систем, ориентированных на управление отдельными операциями (например, систем класса CRM или ERP), разрозненных баз данных, электронных таблиц и других источников. Информация, накопленная в хранилище данных, используется приложениями, обеспечивающими технологии бизнес-планирования, управленческого учета и стратегического прогнозирования. Анализ данных выполняется с использованием различных BI-инструментов (OLAP, Data mining).

1.5. Проблемы интеграции

Интеграция приложений — сложная задача. Можно выделить несколько групп проблем, с которыми сталкиваются разработчики интеграционных решений [33]:

1) *технические проблемы*:

- риск задержки и потери информации, вызванный ненадежностью сетей передачи данных (обеспечивая связь между отдельными частями интеграционного решения, необходимо передавать информацию между устройствами по телефонным линиям, локаль-

ным сетям, через маршрутизаторы, общедоступные сети и каналы спутниковой связи);

▪ недостаточная скорость передачи данных (объективно время доставки данных через компьютерную сеть всегда больше времени вызова локального метода, поэтому интеграционное решение всегда работает медленнее локального приложения);

▪ необходимость учитывать все различия между приложениями (разные языки программирования, на которых написаны приложения, разные платформы, разный формат данных);

▪ ограниченный контроль над интегрируемыми приложениями, представляющими собой, например, унаследованные коммерческие приложения с закрытой структурой баз данных или плохо документированные системы, разработанные ИТ-департаментом;

▪ необходимость поддерживать интеграционное решение, то есть адаптировать его к изменениям в интегрируемых приложениях. Зачастую изменения в одной системе влекут за собой непредсказуемые изменения в других;

2) *методологические проблемы*:

▪ отсутствие корректного формата или семантического слоя для слияния двух и более несопоставимых наборов данных. Устранение семантических различий между приложениями — одна из наиболее сложных и неформализуемых задач интеграции. Действительно, одна и та же сущность (например, «Клиент») может иметь несколько различных семантик, ограничений и допущений в каждой системе. Эта проблема в англоязычной литературе получила название *семантического диссонанса*;

▪ необходимость наличия методологии для документирования таких технических аспектов интеграции, как определение записей, структур данных, интерфейсов и потоков данных в масштабах всей организации;

▪ необходимость определения правил и методов согласования данных во всех интеграционных проектах (правил устранения семантического диссонанса);

3) *организационные проблемы*:

▪ недостаток доверия к корректности информации;

▪ интеграция приложений в большинстве случаев влечет за собой пересмотр корпоративной политики компании. Объединение

информационных систем (например, CRM-системы и системы бухгалтерского учета) требует изменений в порядке взаимодействия между использующими эти системы отделами (отделом маркетинга и бухгалтерией);

- при интеграции ключевых бизнес-функций компании ее деятельность становится зависимой от функционирования интеграционного решения, сбой в работе которого может принести компании существенные убытки (ошибочные платежи, потерянные заказы и т.д.).

1.6. Критерии выбора интеграционного решения

Для обоснованного выбора способа интеграции необходимо обладать информацией о характеристиках интегрируемых приложений, а также уметь оценивать влияние способа интеграции на общую архитектуру интеграционного решения.

Следует понимать, что универсального подхода к интеграции приложений не существует, как не существует и единственного варианта интеграционного решения, но всегда можно найти оптимальный способ интеграции в рамках конкретного интеграционного сценария.

Ниже перечислены основные критерии, влияющие на выбор способа интеграции приложений:

1) *степень связывания приложений*. Зависимости между интегрируемыми приложениями должны быть сведены к минимуму (реализация принципа слабой связи между приложениями). Сильное связывание предполагает наличие большого числа допущений между приложениями. Изменение функциональности или сбой в работе одного приложения может привести к нарушению допущений и потере работоспособности интеграционного решения. Результатом сильного связывания являются неустойчивые, плохо масштабируемые и трудно поддерживаемые решения. В идеале интерфейсы объединяемых приложений должны обеспечивать необходимую функциональность, допуская возможность изменения внутренней реализации;

2) *простота поддержки интеграционного решения*. Следует стремиться к минимизации изменений, вносимых в объединяемые приложения, и объема кода, необходимого для интеграции;

3) *объем данных*. Выбор способа интеграции зависит от объема передаваемых данных. Следует учитывать, что несвоевременная доставка или потеря данных приведет к рассинхронизации приложений;

4) *стоимость решения*. Отдельные интеграционные решения требуют применения специального программного обеспечения (класса middleware), что существенно увеличивает стоимость проекта интеграции. Вместе с тем более «бюджетная» разработка проекта интеграции «с нуля» намного более рискованна и может свести усилия разработчиков к «изобретению велосипеда». В каждом проекте инструмент интеграции необходимо выбирать исходя из масштабов и сроков проекта, задач интеграции, бизнес-требований и наличия квалифицированных кадров.

Глава 2

Технологии и стандарты интеграции

2.1. Понятие промежуточной среды

Технически задача интеграция приложений решается с использованием тех же технологий, что и связывание компонентов распределенных информационных систем.

Взаимодействие удаленных систем описывается в рамках модели взаимодействия открытых систем OSI/ISO, разработанной Международной организацией по стандартам (International Standards Organization, ISO) с целью стандартизации способов связи между хостами от разных производителей.

Модель OSI/ISO разделяет процесс взаимодействия двух сторон на семь взаимосвязанных уровней и определяет функции каждого из них. Каждый уровень поддерживает интерфейсы с выше- и нижележащими уровнями (табл. 2.1).

В простейшем случае взаимодействие между двумя удаленными приложениями можно обеспечить с использованием протокола TCP/IP. Интерфейс к транспортному уровню обеспечивает операционная система, предоставляя механизм, основанный на сокетах[1].

---

[1] *Сокет* — это программный интерфейс для обеспечения обмена данными между процессами; абстрактный объект, представляющий конечную точку соединения.

Таблица 2.1

Уровни модели OSI/ISO и их характеристики

№	Название	Функции	Пример протокола	Реализация
1	Физический	Передача битов по физическим линиям связи. Определяет механические, электрические и оптические характеристики кабелей и разъемов физического интерфейса сети	Спецификация 10Base-T	Канал связи
2	Канальный	Обеспечивает взаимодействие между двумя компьютерами. Функции — проверка доступности среды передачи, реализация механизмов обнаружения и коррекции ошибок. В протоколах канального уровня, используемых в локальных сетях, заложены определенная структура связей между компьютерами и способы их адресации	Ethernet, Token Ring, FDDI, 100VG-AnyLAN	Операционная система
3	Сетевой	Реализует взаимодействие узлов сети. Этот уровень служит для образования единой транспортной системы, объединяющей несколько сетей с различными принципами передачи информации между конечными узлами	Протокол межсетевого взаимодействия IP стека TCP/IP и протокол межсетевого обмена пакетами IPX стека Novell	
4	Транспортный	Обеспечивает возможность передачи более длинных сообщений между хостами	Протоколы TCP и UDP стека TCP/IP и протокол SPX стека Novell	
5	Сеансовый	Устанавливает и поддерживает соединение между компонентами распределенной системы. Использует механизм соединений транспортного уровня		Промежуточная среда
6	Представительский	Устраняет различия в представлении данных. Этот уровень гарантирует то, что информация, передаваемая прикладным уровнем, будет понятна прикладному уровню в другой системе. При необходимости уровень преобразовывает данные в некоторый общий формат представления или выполняет обратное преобразование	Secure Socket Layer (SSL)	
7	Прикладной	Обеспечивает функционирование приложения. Набор протоколов, с помощью которых пользователи сети получают доступ к разделяемым ресурсам	Протокол электронной почты	Компонент информационной системы

Сокеты обеспечивают примитивы низкого уровня для непосредственного обмена потоком байт между двумя процессами и могут быть использованы для организации простейшего взаимодействия удаленных приложений.

Приведем пример. Пусть клиенту банка необходимо обеспечить возможность перевода денег на свой счет из другого банка, используя Web-приложение (задача интеграции Web-приложения с финансовой системой). Простейшее решение может быть получено с использованием протокола TCP/IP.

Для определенности представим, что необходимо передать только имя клиента и сумму денежного перевода. Приведенный ниже пример кода (C#) демонстрирует вариант реализации конечной точки Web-приложения, в котором создается сокет с адресом my.bank.com и по сети пересылается сумма и имя клиента.

```
String hostName = "my.bank.com";
//Задание DNS-имени удаленного компьютера
int port = 80;
IPHostEntry hostInfo = Dns.GetHostByName (hostName);
//Преобразование имени удаленного компьютера в IP-адрес
IPAddress address = hostInfo.AddressList [0];
IPEndPoint endpoint = new IPEndPoint (address, port);
Socket socket = new Socket (address.AddressFamily, SocketType.
Stream, ProtocolType.Tcp);
socket.Connect (endpoint);
byte [] amount = BitConverter.GetBytes (1000);
//Преобразование данных в поток байтов
byte [] name = Encoding.ASCII.GetBytes ("Joe");
int bytesSent = socket.Send (amount);
bytesSent += socket.Send (name);
socket.Close ();
```

На первый взгляд простой метод межплатформенной интеграции имеет ряд ограничений:

- связываемые системы должны иметь одинаковое внутреннее представление целых чисел (32 или 64 бит) и одинаковый порядок следования байтов (прямой или обратный). В противном случае передаваемые данные будут неверно интерпретированы приложением-получателем;

▪ компьютер должен иметь неизменяемый адрес. Перемещение приложения получателя на другой компьютер потребует изменения программного кода;

▪ интегрируемые приложения должны быть доступны в одно и то же время, поскольку протокол TCP/IP является протоколом с установкой соединения. Если одно из взаимодействующих приложений в момент установки соединения недоступно, то соединение не будет установлено;

▪ использование соглашения о формате данных, предаваемых между интегрируемыми приложениями. Любое изменение в формате потребует изменений в программном коде как на стороне отправителя, так и на стороне получателя.

Таким образом, использование механизма сокетов приводит к получению плохо масштабируемого, неустойчивого, трудно поддерживаемого интеграционного решения, обеспечивающего «жесткую» связь между приложениями.

В середине 1990-х гг. появилось понятие промежуточной среды [20, 18] как некоторой дополнительной, не зависящей от операционной системы и приложений «прослойки», реализующей сервисы высокого уровня для инкапсуляции удаленного взаимодействия между программными компонентами. Промежуточная среда (middleware — связующее программное обеспечение) выполняет функции сеансового и представительского уровней модели OSI/ISO. Наличие промежуточной среды позволяет создать открытые, масштабируемые и отказоустойчивые распределенные системы.

Промежуточная среда предоставляет:

▪ единый, независимый от операционной системы механизм использования одними программными компонентами сервисов других компонентов;

▪ безопасность — аутентификация и авторизация всех пользователей сервисов компонента и защита передаваемой между компонентами информации от искажения и чтения третьей стороной;

▪ целостность данных — управление транзакциями, распределенными между удаленными компонентами системы;

▪ балансировку нагрузки на серверы с программными компонентами;

▪ обнаружение удаленных компонентов.

В зависимости от модели взаимодействия компонентов можно выделить следующие типы промежуточных сред:

- *ориентированные на посылку сообщений* — поддерживают связь между компонентами распределенной системы посредством обмена сообщениями (Microsoft Message Queuing, IBM MQSeries, Sun Java System Message Queue);
- *объектно-ориентированные* — построены на модели удаленного обращения к методу (J2EE, .NET, CORBA);
- *транзакционно-ориентированные* — ориентированы на поддержку транзакций в системах распределенных баз данных (мониторы транзакций, все современные промышленные СУБД, например MS SQL SERVER).

Одно интеграционное решение может использовать несколько видов промежуточных сред.

2.2. Модели взаимодействия приложений

Взаимодействие информационных систем может быть описано в рамках модели «клиент-сервер». В данном контексте под *клиентом* будем понимать приложение, которое отправляет запрос на обработку. Приложение, отвечающее на запрос, будем называть *сервером*. В большинстве случаев одно и то же приложение в зависимости от ситуации может выступать в роли как клиента, так и сервера.

Взаимодействие между приложениями может быть синхронным или асинхронным. *Синхронным* называется такое взаимодействие, при котором приложение, выступающее в роли клиента, отослав запрос, блокируется и может продолжать работу только после получения ответа от приложения, выступающего в роли сервера. Иногда такой вид взаимодействия называют *блокирующим*.

Схема синхронного взаимодействия представлена на рис. 2.1.

В случае *асинхронного взаимодействия* приложение-клиент после отправки запроса приложению-серверу может продолжать работу, даже если ответ на запрос еще не пришел. Иногда такой вид взаимодействия называют *неблокирующим*. Схема асинхронного взаимодействия представлена на рис. 2.2.

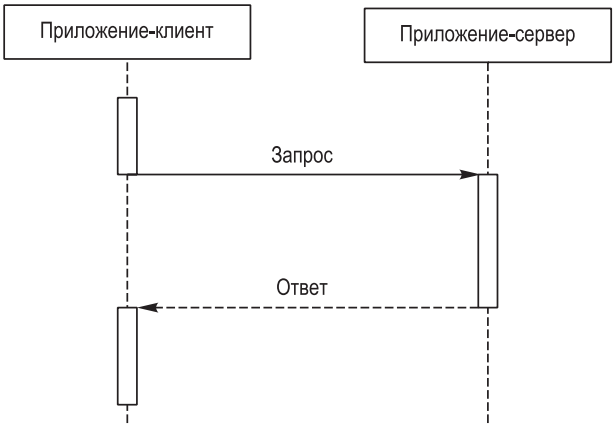

Рис. 2.1. Синхронное взаимодействие

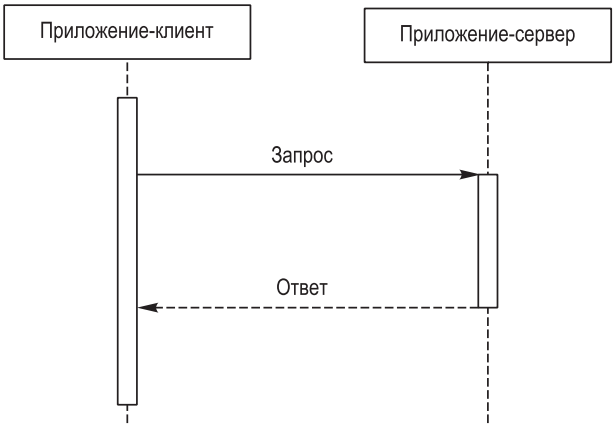

Рис. 2.2. Асинхронное взаимодействие

Технически синхронное взаимодействие проще реализовать, однако оно приводит к неоправданным затратам времени на ожидание ответа. Работа приложения, играющего роль клиента, приостанавливается, хотя оно могло бы выполнять другие действия, не зависящие от ожидаемого ответа.

Асинхронное взаимодействие позволяет достичь более высокой производительности систем, поскольку время между отправкой запроса и получением ответа на него может быть использовано

для выполнения других задач приложением-клиентом. Еще одним преимуществом асинхронного взаимодействия является меньшая зависимость приложения-клиента от приложения-сервера и возможность продолжать работу, даже если машина, на которой находится сервер, стала недоступной.

Сложности реализации асинхронного взаимодействия обусловлены следующими обстоятельствами:

- возможность использования нескольких потоков исполнения, увеличивающая производительность решения, одновременно затрудняет его отладку;
- приложение-клиент должно быть готово принять ответ в любой момент, даже в процессе выполнения другой задачи;
- поскольку асинхронные процессы могут выполняться в любом порядке, приложение-клиент должно уметь обрабатывать результат запроса с учетом времени получения.

2.3. Стандарты объектно-ориентированного взаимодействия

В данную группу входят стандарты, основанные на концепции удаленного вызова метода или процедуры. К ним относят стандарты COM (DCOM, COM+), RMI, CORBA [2, 22, 29].

В основу перечисленных стандартов положен механизм виртуализации. Идея заключается в создании виртуального компонента, являющегося прокси-объектом (локальным представителем) удаленного метода или процедуры. Приложение-клиент обращается к прокси-объекту, который и передает сообщение к удаленному объекту.

В качестве примера рассмотрим технологии удаленного вызова процедуры (remote procedure call, RPC) и удаленного вызова метода (remote method invocation, RMI).

Удаленный вызов процедуры впервые был реализован в начале 1980-х гг. компанией Sun Microsystems и явился прообразом объектно-ориентированного промежуточного слоя применительно к структурной парадигме программирования. Суть технологии RPC заключается в том, что вызов функции на удаленном компьютере осуществляется с помощью промежуточного программного обеспечения так же, как и на локальном [22].

Для того чтобы организовать удаленный вызов процедуры необходимо:

- описать интерфейс процедуры, которая будет использоваться для удаленного вызова при помощи языка определения интерфейсов (Interface Definition Language, IDL);
- скомпилировать определение процедуры (при этом будут созданы описание этой процедуры на том языке программирования, на котором будут разрабатываться клиент и сервер, и два дополнительных компонента — клиентский и серверный переходники). Клиентский переходник представляет собой процедуру-заглушку (stub), которая будет вызываться клиентским приложением. Клиентский переходник размещается на той же машине, где находится компонент-клиент, а серверный переходник — на той же машине, где находится компонент-сервер.

При обработке вызова процедуры выполняются следующие действия:

- клиентский переходник выполняет привязку к серверу (определяет физическое местонахождение (адрес) сервера);
- клиентский переходник выполняет маршалинг[1] (упаковывает вызов процедуры и ее аргументы в сообщение в некотором стандартном для данной системы формате);
- клиентский переходник выполняет сериализацию[2] сообщения (преобразует полученное сообщение в поток байтов) и отсылает с помощью какого-либо протокола (транспортного или более высокого уровня) на машину, на которой помещен серверный компонент;
- серверный переходник принимает сообщение, содержащее параметры вызова процедуры, распаковывает эти параметры при помощи десериализации[3] и демаршалинга[4], вызывает локально соответствующую функцию серверного компонента, получает ее результат, упаковывает его и посылает по сети на клиентскую машину;
- после получения ответа от сервера выполняется процедура десериализации.

---

[1] Процесс преобразования параметров для передачи их между процессами при удаленном вызове называется *маршализацией* (*marshaling*).

[2] Процесс преобразования экземпляра какого-либо типа данных в набор байт называется *сериализацией*.

[3] Процедура, обратная сериализации.

[4] Процедура, обратная маршалингу.

Сущность технологии удаленного вызова процедуры иллюстрирует рис. 2.3.

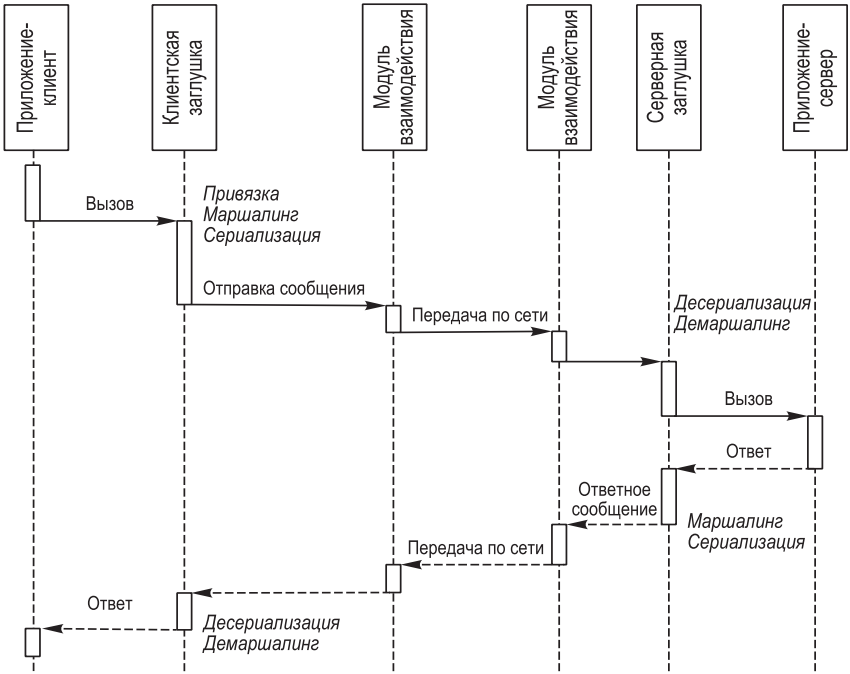

Рис. 2.3. Удаленный вызов процедуры

Удаленный вызов метода является модификацией метода удаленного вызова процедуры для объектной парадигмы программирования.

В момент вызова удаленного метода на стороне клиента создается клиентская заглушка, называемая *посредником* (*proxy*). Клиентская заглушка в своем интерфейсе имеет все методы объекта-сервера, выполняет маршалинг и сериализацию параметров вызываемых методов и передает их по сети. Поскольку один объект сервера может предоставлять несколько методов, информация о том, какой именно метод вызывается, также упаковывается вместе с аргументами вызова.

Промежуточная среда на стороне сервера десериализует параметры и передает их заглушке, называемой *каркасом*, или *скелетоном* (*skeleton*), на стороне сервера.

В табл. 2.2 приводится краткая характеристика стандартов объектно-ориентированного взаимодействия.

Таблица 2.2

Стандарты объектно-ориентированного взаимодействия

Стандарт	Назначение	Достоинства	Ограничения
Com	Средство взаимодействия приложений, функционирующих на одном компьютере	Высокий уровень абстракции Простота использования Очень высокая производительность	COM-приложения способны функционировать только под управлением Windows (двоичная структура объектов компонентной модели Microsoft) Распределенные решения плохо масштабируются (жесткая связь между клиентом и сервером) Невысокая устойчивость к сбоям COM-систем (жесткая привязка сервера к клиенту, двухуровневая архитектура)
Com+	Промежуточная среда для создания распределенных систем, действующих в локальной сети	Поддержка синхронного и асинхронного взаимодействий программных компонентов Динамическая балансировка нагрузки Поддержка логической целостности данных за счет работы с координатором распределенных транзакций Возможность ограничения доступа к компоненту в разрезе ролей, связанных с учетными записями пользователей	Использование ограничено взаимодействием компонентов внутри локальной или виртуальной частной сети, построенной на базе Microsoft Windows, в пределах одного предприятия

Окончание таблицы 2.2

Стандарт	Назначение	Достоинства	Ограничения
DCOM	Промежуточная среда для создания распределенных систем, действующих в локальной сети и сети Интернет	Независимость от языка программирования Простота Поддержка распределенных транзакций	Использование ограничено платформами Windows
CORBA	Набор спецификаций для промежуточного программного обеспечения объектного типа. Создавалась консорциумом OMG как универсальная методология создания распределенных систем в гетерогенных средах	Кроссплатформенность Высокий уровень устойчивости к внутренним сбоям за счет большей изоляции клиентов и серверов (трехуровневая архитектура) Передача данных между компонентами происходит при помощи протокола UDP (обеспечивает экономию системных ресурсов) Набор сервисов (управление транзакциями, безопасностью, жизненным циклом объектов, событиями и т.д.)	Сложность технологии
RMI	Архитектура (Remote Method Invocation, то есть вызов удаленного метода), которая интегрирована с JDK1.1 и реализует распределенную модель вычислений	Самый простой и быстрый способ создания распределенных систем Распределенное автоматическое управление объектами	Поддержка одного языка программирования Java Собственный протокол взаимодействия Трудность интегрирования с существующими приложениями

2.4. Технологии, базирующиеся на XML

Расширяемый язык разметки XML (eXtensible Markup Language) приобрел известность в конце 1990-х гг., когда он начал широко использоваться для переноса данных между различными информационными системами и описания бизнес-транзакций.

XML — это язык разметки документов, предназначенный для хранения структурированных данных, обмена информацией между программами, а также для создания на его основе специализированных производных языков [17, 15, 35].

В начале 2000-х гг., когда на первый план вышли проблемы интеграции разнородных приложений, появилась концепция Web-служб и основанная на ней парадигма сервис-ориентированной архитектуры, начали развиваться технологии управления бизнес-процессами (BPM). Крупнейшие поставщики программного обеспечения активно разрабатывали внутрикорпоративные стандарты описания, реализации и исполнения бизнес-процессов для своих программных систем. Появились многочисленные спецификации языков описания бизнес-процессов (JPDL, XLang, WSFL, WSCL, BPSS, WSCI), опубликованные конкурирующими вендорами [16]. На сегодняшний день общепризнанными стандартами процессного управления являются основанные на XML языки: WSDL (Web Service Definition Language), WS-BPEL (Web Services Business Prosecc Execution Language) и WS-CDL (Web Services Choreography Discription Language).

2.4.1. Целесообразность применения XML в интеграционных задачах

Целесообразность использования языка XML и основанных на нем технологий для решения интеграционных задач обусловлена следующими преимуществами:

1) XML является самоописывающим языком. XML-документ содержит элементы и атрибуты, использование которых подчинено строгим правилам, однако имена элементов и атрибутов имеют содержательный характер. Такой документ может быть легко прочитан прикладной программой (синтаксическим анализатором) и досту-

пен для прочтения человеку. Имеется возможность задавать имена в соответствии с принятыми в конкретной прикладной области стандартами, а не на основе жесткого синтаксиса;

2) XML — самодокументируемый формат, предназначенный не только для описания данных, но и для определения метаданных, в том числе для описания дополнительных правил, ограничивающих данные в соответствии с информационной моделью предметной области. Например, XSD-схема позволяет задать следующие типы ограничений:

- базовые и производные типы данных (строки, даты, логический и др.);
- расширенные типы данных (произвольные пользовательские типы данных);
- фасеты (дополнительные ограничения, такие как длина, дробные числа и т.д.);
- границы значений (наибольшее и наименьшее значения);
- перечисление (набор допустимых значений для атрибута);
- условия кардинальности (вхождение повторяющегося элемента данных);
- шаблоны;

3) XML — универсальный язык, подходящий для описания практически любых типов документов. В формате XML могут быть описаны основные структуры данных, такие как записи, списки и деревья;

4) XML — расширяемый язык. Любую XML-структуру можно разрабатывать в расчете на оперативное сокращение или расширение. Расширение возможно как за счет включения в структуру новых элементов и/или атрибутов, так и путем использования механизмов пространств имен;

5) XML обеспечивает межплатформенное взаимодействие. Поскольку XML — это текстовый формат, требования для организации взаимодействия между отправляющей и принимающей платформами минимальны и сводятся к двум моментам:

- возможность читать и создавать текстовые файлы в кодировке Юникод (UTF-8 или ASCII);
- наличие синтаксического анализатора для обработки XML-документов;

6) безусловным преимуществом является наличие международных стандартов (поддерживаются консорциумом W3C — www.w3.org).

Язык имеет строго определенные синтаксис и требования к анализу, что позволяет ему оставаться простым, эффективным и непротиворечивым. Актуальными спецификациями являются XML 1.0, XML Information Set, Namespaces in XML, XML Schema. В настоящее время XML широко используется для хранения и обработки документов и поддерживается всеми ведущими производителями программного обеспечения;

7) XML ориентирован на повторное использование, что позволяет уменьшить стоимость разработки интеграционного решения, снизить эксплуатационные затраты, а также способствует продвижению корпоративных и отраслевых стандартов. Например, XML-схемы могут быть спроектированы как модульные внешние компоненты. Нестандартные элементы и типы данных целесообразно определять в отдельных XML-схемах, а затем многократно использовать посредством их включения в документы или с помощью ссылки на них.

2.4.2. Синтаксис XML

XML описывает определенный класс объектов, называемых XML-документами. Документ представляется в виде дерева элементов, каждый из которых может иметь набор атрибутов, а также содержать другие элементы или текст.

XML-документ описывается в терминах логической и физической структуры. Приведем краткие сведения об элементах логической и физической структуры XML-документов.

▸ Логическая структура XML-документа

Логическая структура состоит из следующих элементов:

- объявление;
- определения типа документа;
- элементы;
- комментарии;
- ссылки;
- инструкции по обработке документа.

В табл. 2.3 представлены основные требования спецификации XML 1.0, предъявляемые к синтаксису XML-документов.

Приведем пример XML-документа.

```
<?xml version="1.0" encoding="windows-1251"?>
<!--Пример XML-документа -->
     <Контрагенты>
          <Контрагент Код="Ю023">
          <Наименование>Рога и копыта</Наименование>
          <ИНН>1232345678</ИНН>
          <КПП>775003657</КПП>
          <Адрес Индекс="118200" Город="Москва" Улица="Широкая"
          Дом="100">
               </Адрес>
          <КонтактноеЛицо ФИО="Иванов Иван Иванович" >
               <Телефон>
                    <СлужебныйТелефон>74952113477</СлужебныйТелефон>
                    <МобильныйТелефон>79056784523</МобильныйТелефон>
               </Телефон>
          </КонтактноеЛицо>
     </Контрагент>
</Контрагенты>
```

Таблица 2.3

Синтаксис XML

Элемент логической структуры	Описание	Пример
1	2	3
Объявление	Размещается в начале документа. Ограничивается тегами <?xml и?>. Включает атрибуты: 1) version — номер версии спецификации XML, обязательный атрибут; 2) encoding — кодировка символов документа (по умолчанию encoding="UTF-8"), необязательный атрибут. Если имена тегов задаются на русском языке, необходимо установить encoding="windows-1251"; 3) standalone — указание на наличие внешних описаний структуры документа, по умолчанию standalone="no", необязательный атрибут. Атрибуты должны следовать в указанном выше порядке. Если атрибуты не определены, то им присваивается значение по умолчанию	<?xml version="1.0" encoding="UTF-8"?>

Продолжение таблицы 2.3

1	2	3
Определения типа документа	DTD (Document Type Declaration) заключается между символами <! DOCTYPE…> и может занимать несколько строк. В этой части объявляются теги, использованные в документе, или приводится ссылка на файл, в котором записаны такие объявления. Секция DTD должна располагаться перед корневым элементом	Пример см. в п. «Языки описания структуры»

Элементы	Элементы являются основными составляющими XML-документа; бóльшая часть данных в XML-документах содержится в элементах. Элемент представляется в XML-документе с помощью открывающего (<>) и закрывающего (</>) тэгов. Открывающий тэг записывается в формате <ИмяЭлемента>, а закрывающий тэг — в формате </ИмяЭлемента>.
	Имя элемента не может содержать пробелов.
	Содержимым элемента могут быть символьные данные (текст), другие элементы (известные как дочерние элементы), а также оно может отсутствовать (пустой элемент).
	XML-документ должен содержать обязательный корневой элемент.
	Элемент может содержать любое число атрибутов, содержащих дополнительную информацию о данных, которые представляет элемент. Атрибуты указываются в виде пар «название-значение» в открывающем тэге элемента. Значения атрибутов заключаются в кавычки.
	Названия атрибутов уникальны в рамках одного элемента (в одном элементе не может быть двух атрибутов с одинаковым именем)
		`<Книга>`
		`</Книга>`
		
		`<Книга isbn="978-5-9775-0778-3">`
		`</Книга>`

Комментарии	Ограничиваются тегами <! и !>. Используются для документирования. Могут располагаться в любом месте документа	`<!-- …текст комментария… -->`
Ссылки	Ограничиваются символами «&» и «;». Используются для подстановки вместо них символов (ссылки на символы) или различных данных (ссылки на сущности), описанных в определении DTD. Ссылки на символы позволяют вставить в текст документа некоторый символ, который, например, отсутствует в раскладке клавиатуры либо может быть неправильно истолкован анализатором.	&# код_символа_в_Unicode &#xШестнадцатеричный_код_символа имя_сущности

Окончание таблицы 2.3

1	2	3
	Ссылки на сущности позволяют включать любые строковые константы в содержание элементов или значения атрибутов. Ссылки на сущности указывают программе-анализатору подставить вместо них строку символов, заранее заданную в определении типа документа. Для включения в XML-документ символьных данных, которые не следует обрабатывать, используется секция <! [CDATA [содержание секции]] >	
Инструкции по обработке	Ограничиваются тегами <? и ?>. Предназначены для передачи информации приложению, работающему с XML-документом. За начальным вопросительным знаком записывается имя программного модуля, которому предназначена инструкция. Далее через пробел записывается инструкция, передаваемая программному модулю	`<?xml version="1.0" encoding="windows-1251"?>` Эта инструкция предназначена программе, обрабатывающей документ XML. Инструкция передает ей номер версии и кодировку, в которой записан документ

▸ Физическая структура XML-документа

Физическая структура XML-документа описывает его как набор сущностей. Документ должен содержать как минимум одну сущность — корневую сущность документа. Сущности могут включаться в XML-документ также с помощью XML-ссылок.

XML-ссылка — это ссылка на внешний объект, содержимое которого размещается в указанном месте документа. Ссылка на сущность работает как подстановка и обеспечивает модульность XML-документа, которая, как будет показано ниже, позволяет объединять данные из разных источников в единую структуру и легко собирать документы, а также их схемы из пригодных для повторного использования блоков.

2.4.3. Пространства имен

Различные приложения могут использовать сущности, имеющие одинаковые имена и содержащие различные данные. Для предотвращения конфликтов имен в XML используются пространства имен,

которые представляют собой коллекции имен. В каждой коллекции имен все имена уникальны. Каждая коллекция должна иметь уникальный идентификатор (URI-адрес). Каждое XML-имя характеризуется идентификатором пространства имен и локальным именем в пределах своего пространства имен. Таким образом, появляется возможность определить элементы, имеющие одинаковые имена, но связанные с различными URI.

Рассмотрим правила использования пространств имен на конкретном примере. Пусть в одном документе необходимо объединить данные о клиенте компании, поступающие из разных источников. Из CRM-системы поступает информация о персональных данных клиента, из системы учета заказов — данные о заказе.

```
CRM
```

```
<Client>
     <Name>Mary</Name>
     <Phone>+7.602.555.9999</Phone>
     <Fax>+7.602.555.9999</Fax>
</Client>
```

```
Система учета заказов
```

```
<Client>
     <OrderNumber>2223</OderNumber>
     <OrderData>10.10.2012</OrderData>
</Client>
```

Пространство имен объявляется с помощью зарезервированного имени xmlns. Ниже приводится пример объявления пространств имен в XML-документе.

```
<Clients xmlns:ClientInfo="http://www.mycompany.com/ClientInformation"
     xmlns:ClientOrderData="http://www.mycompany.com/ClientOrderData">
```

ClientInfo и ClientOrderData являются префиксами пространств имен и представляют сокращенные наименования идентификаторов.

После объявления пространств имен их префиксы могут использоваться в документе для определения принадлежности каждого элемента к конкретному пространству имен.

Для рассмотренного примера XML-документ, содержащий данные из двух пространств имен, будет выглядеть следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
<Clients xmlns:ClientInfo = "http://www.mycompany.com/
ClientInformation"
          xmlns:ClientOderData="http://www.mycompany.com/
          ClientOrderData" >
     <ClientInfo:Client>
          <ClientInfo:Name>Mary</ClientInfo: Name>
          <ClientInfo:Phone>+7.602.555.9999</ClientInfo:Phone>
          <ClientInfo:Fax>+7.602.555.9999</ClientInfo:Fax>
     </ClientInfo:Client>
     <ClientOrderData:Client>
          <ClientOderData:OrderNumber>2223</ClientOderData:OrderNumber>
          <ClientOderData:OrderData>10.10.2012</ClientOderData:OrderData>
     </ClientOderData:Client>
</Clients>
```

Имя элемента или атрибута с префиксом называется *уточненным именем* (qualified name или QName) и используется анализаторами XML для извлечения элементов, принадлежащих соответствующим пространствам имен в пределах глобального XML-пространства имен http://www.w3.org/XML/1998/namespace.

Если пространство имен объявлено без префикса, то оно является пространством имен по умолчанию для тех элементов XML-документа, которые не используют префикс. Каждое пространство имен имеет свою область действия в рамках XML-документа. Объявление пространства имен применяется к элементу, содержащему определение, а также ко всем его дочерним элементам, если оно не переопределяется другим пространством имен в определении элемента. Имена атрибутов также можно уточнять, используя префикс объявленного пространства имен. Для атрибутов нельзя использовать пространства имен по умолчанию. Если для атрибута не указан префикс, то он не принадлежит ни к какому пространству имен. Атрибуты элементов для связывания с пространствами имен всегда необходимо уточнять префиксами.

Приведем пример использования пространства имен http://www.mycompany.com/ClientInformation как пространства имен по умолчанию:

```
<?xml version="1.0" encoding="utf-8"?>
<Clients xmlns="http://www.mycompany.com/ClientInformation"
         xmlns:ClientOrderData="http://www.mycompany.com/
         ClientOrderData">
     <Client>
          <Name>Mary</Name>
          <Phone>+7.602.555.9999</Phone>
          <Fax>+7.602.555.9999</Fax>
     </Client>
     <ClientOrderData:Client>
          <OrderNumber>2223</OrderNumber>
          <OrderData>10.10.2012</OrderData>
     </ClientOrderData:Client>
</Clients>
```

2.4.4. Описание структуры XML-документов

Каждый XML-документ несет информацию о данных и их структуре (описание метаданных).

XML-документы могут быть двух типов:

1) документы, созданные с учетом логических и структурных правил;

2) документы, не использующие никаких правил, кроме синтаксических правил оформления XML-документов.

Проверку документов первого типа на соответствие заданным правилам осуществляет XML-процессор. Проверка документов второго типа выполняется разработчиком.

При создании документа первого типа описание его структуры может быть выполнено с использованием таких языков, как Document Type Definitions (DTD), XML Schema, RELAX NG, XML Data-Reduced и др. [24, 31]. Наибольшее распространение получили языки DTD и XML Schema.

Далее анализируются сильные и слабые стороны наиболее распространенных языков описания структуры и приводится краткое изложение их основ. Поскольку данное учебное пособие посвящено проблемам интеграции информационных систем, при рассмотрении языков описания структуры основное внимание будет уделено вопросам модульности и повторного использования схем.

▶ Язык Document Type Definitions (DTD)

Язык Document Type Definitions (DTD) не базируется на XML. Этот язык имеет ряд ограничений в описании метаданных: не является расширяемым, не поддерживает строгое типизирование данных, ограниченно поддерживает пространства имен. Описание структуры на языке DTD постепенно вытесняется технологией XML Schema, однако до настоящего времени продолжает использоваться (иногда совместно с XML Schema).

Приведем краткое изложение правил использования основных конструкций DTD.

Описание структуры на языке DTD может быть включено в XML-документ (внутреннее подмножество) или размещено в отдельном файле, также возможен вариант смешанного описания. Во всех случаях для определения DTD необходимо использовать объявление <! DOCTYPE...>. В табл. 2.4 приведены правила включения DTD-описания в XML-документ.

Таблица 2.4

Правила включения DTD-описания в XML-документ

Вариант описания структуры	Правила записи
Внутренне описание DTD	<! DOCTYPE ROOT [<!— ОБЪЯВЛЕНИЯ DTD —>] > где ROOT — имя корневого элемента; <!— ОБЪЯВЛЕНИЯ DTD —> — описание структуры на языке DTD
Внешнее описание DTD	<! DOCTYPE ROOT SYSTEM "SYSTEM_ID"> где ROOT — имя корневого элемента; SYSTEM_ID — место расположения файла внешнего DTD. Например, <! DOCTYPE Clients SYSTEM "Clients.DTD"> <! DOCTYPE ROOT PUBLIC "PUBLIC_ID" "SYSTEM_ID"> где SYSTEM_ID и PUBLIC_ID — место размещения файла внешнего DTD; PUBLIC_ID не зависит от места размещения XML-файла (например, место в локальной сети); "SYSTEM_ID" будет использовано только в том случае, если нет доступа к "PUBLIC_ID". Например, <! DOCTYPE Clients PUBLIC"-//W3C//DTD Strict/RU" "Client.dtd">

Язык DTD позволяет описать требования к элементам и атрибутам документа. При описании элементов указываются:

- модель содержимого, описывающая также наличие дочерних элементов;
- ограничения на количество повторений элемента в документе.

Для описания элемента используется конструкция следующего вида:

```
! ELEMENT Имя_элемента Модель_содержимого>.
```

Правила записи модели содержимого представлены в табл. 2.5.

Таблица 2.5

Правила записи модели содержимого

Модель содержимого	Описание	Пример
ANY	Элемент может содержать любые дочерние элементы или текст	<! ELEMENT Client ANY>
EMPTY	Элемент не может содержать дочерние элементы или текст, может иметь атрибуты	<! ELEMENT OrderID EMPTY>
(#PCDATA)	Элемент может содержать только текст	<! ELEMENT Phone (#PCDATA) >
(NAME1, NAME2)	Элемент содержит указанные дочерние элементы в указанном порядке, не может содержать текст	<! ELEMENT Client (Name, Phone) >
(NAME1|NAME2)	Элемент содержит один из указанных взаимоисключающих элементов, не может содержать текст	<! ELEMENT Client (Fax |Phone) >
Смешанная модель	Элемент может содержать текст и дочерние элементы	<! ELEMENT Client (#PCDATA, Name, Phone) >

Ограничения на количество дочерних элементов задается следующим образом (табл. 2.6).

Таблица 2.6

Ограничения на количество дочерних элементов

Оператор количества элементов	Описание	Пример
Нет	Допустимо использовать один экземпляр элемента	<! ELEMENT Client (INN) >
*	Элемент может повторяться ноль и более раз	<! ELEMENT Clients (Client*) >
+	Элемент может повторяться один и более раз	<! ELEMENT Client (Phone+) >
?	Элемент может повторяться ноль или один раз	<! ELEMENT Client (Address?) >

Атрибуты элементов объявляются для каждого элемента, если это необходимо, с помощью объявления ATTLIST.

При описании атрибутов указываются:

- тип атрибута;
- ограничения на употребление атрибута.

Для описания атрибутов элемента используется конструкция следующего вида:

```
<! ATTLIST Имя_элемента Имя_атрибута1 Тип_атрибута
           (Значение_по_умолчанию | Ключевое_слово)
           Имя_атрибута2 Тип_атрибута (Значение_по_умолчанию |
           Ключевое_слово) >
```

Правила описания допустимых типов атрибутов и ключевых слов приведены в табл. 2.7.

Таблица 2.7

Правила описания допустимых типов атрибутов и ключевых слов

Тип атрибута	Описание
CDATA	Строка символов
ID	Уникальное в рамках документа значение (аналог первичного ключа в базе данных), элемент не может иметь больше одного атрибута типа ID

Окончание таблицы 2.7

Тип атрибута	Описание
IDREF	Ссылка на элемент, обладающий атрибутом ID с тем же самым значением, что и значение заданного атрибута IDREF. Используется для создания связей и перекрестных ссылок в документе. Аналог отношения «один-к-одному» в реляционной базе данных
IDREFS	Последовательность ссылок IDREF, разделенных пробелами. Позволяет смоделировать отношение «один-ко-многим»
ENTITY	Определяет имя внутренней или внешней сущности, предназначенной для повторного использования. В том числе используется для определения имени примитива, игнорируемого анализатором. Позволяет ссылаться на данные, структура которых нарушает разметку по правилам XML (в частности, использовать в XML-документах ссылки на двоичные файлы)
ENTITIES	Перечень значений ENTITY, разделенных пробелами
NMTOKEN	Имя, содержащее только символы, применяемые в именах (строка, состоящая из букв, цифр и символов «.», «-», «_», «:»). Может содержать имена других элементов или атрибутов
NMTOKENS	Перечень значений NMTOKEN, разделенных пробелами

Ограничения на использование атрибутов задаются с помощью следующих ключевых слов (табл. 2.8).

Таблица 2.8

Ограничения на использование атрибутов

Ключевое слово	Описание	Пример
#REQUIRED	Обязательный атрибут	ClientID ID #REQUIRED
#IMPLIED	Необязательный атрибут	Address Region #IMPLIED
«Значение по умолчанию»	Если атрибут отсутствует, то анализатор принимает в качестве его значения значение, заданное по умолчанию. Если атрибут имеется, то он может принимать любое значение	
#FIXED «Значение»	Атрибут является необязательным, но если значение указано, то оно задается по умолчанию	Клиент Тип #FIXED «ФЛ»

В качестве примера рассмотрим XML-документ и соответствующее ему DTD-описание.

```
<Clients>
     <Client ClientID="0001" ClientType="ФЛ" Class="VIP"
     System="CRM">
          <Name>Mary</Name>
          <Phone>
               <Home>+7.677.444.1111</Home>
               <Mobile>+7.555.444.3333</Mobile>
               <Mobile>+7.666.333.2222</Mobile>
          </Phone>
          <mail>Mary@qq.com</mail>
          <Country>Russia</Country>
     </Client>
```

```
<! ELEMENT Clients (Client*) >
<! ELEMENT Client (Name,Phone,Mail,Country) >
<! ELEMENT Name (#PCDATA) >
<! ELEMENT Phone (Home?,Mobile+) >
<! ELEMENT Mail (#PCDATA) >
<! ELEMENT Country (#PCDATA) >
<! ELEMENT Home (#PCDATA) >
<! ELEMENT Mobile (#PCDATA) >
<! ATTLIST Client
     ClientID ID #REQUIRED
     ClientType CDATA #REQUIRED
     Class CDATA #IMPLIED
     System NMTOKEN #FIXED "CRM">
```

Для описания логического компонента, многократно используемого разными XML-документами, применяются примитивы, которые задаются путем указания типа атрибута ENTITY. Примитивы могут быть внутренними (логический компонент повторно используется в одном документе) и внешними (повторно используется логический элемент из внешнего файла).

В зависимости от содержимого примитивы можно разделить на анализируемые примитивы (ссылаются на правильно сформированное содержимое XML) и примитивы, игнорируемые анализатором (ссылаются на не-XML-данные). Поскольку XML-анализатор не способен обрабатывать данные в двоичных и других не-XML-фор-

матах, каждому примитиву, игнорируемому анализатором, должна соответствовать нотация. Нотации применяются для того, чтобы связать формат внешних данных, используемых в XML-документе, с внешней программой-обработчиком. Анализатор отошлет не-XML-данные на обработку указанной программе.

XML-документ, использующий нотации, играет роль унифицирующего документа, объединяющего разнородные данные.

Форматы описания логических компонентов XML приведены в табл. 2.9.

Таблица 2.9

Форматы описания логических компонентов XML

Тип логического компонента	Формат описания (пример)
Внутренний примитив	<! ENTITY Имя_примитива "Значение"> ***Пример*** Строка DTD <! ENTITY Система "1С:Предприятие"> В DTD используется примитив **Система**, имеющий значение **1С:Предприятие**. Строка XML <Клиент ИсточникДанных="&Система;"> В XML-документе атрибуту **Источник данных** тега **Клиент** будет присвоено значение **1С:Предприятие**
Внешний примитив, обрабатываемый анализатором	<! ENTITY Имя_примитива SYSTEM "SYS_ID"> или <! ENTITY Имя_примитива PUBLIC "PUBL_ID" "SYS_ID">, где "SYS_ID" — ссылка на реально существующий внешний файл (URL); "PUBL_ID" — идентификатор ресурса (URI), не обязательно реально существующий. ***Пример*** Строка DTD <! ENTITY Описание SYSTEM "C:/Discription.xml"> В DTD используется внешний примитив **Описание**, который определяет внешний файл C:/Discription.xml, содержащий описание клиента. Строка XML <Клиент>&Описание;</Клиент> В тег **Клиент** будет вставлено содержимое файла C:/Discription.xml

Окончание таблицы 2.9

Тип логического компонента	Формат описания (пример)
Внешний примитив, игнорируемый анализатором	<! ENTITY Имя_примитива SYSTEM "SYSTEM_ID" NDATA Имя нотации>, где "SYSTEM_ID" — ссылка на реально существующий внешний файл (URL); Имя_нотации — имя нотации, содержащей информацию о программе-обработчике не-XML-данных. Для описания нотации используется следующий формат: <! NOTATION Тип_данных SYSTEM "SYSTEM_ID">, где Тип_данных — имя формата данных; SYSTEM_ID — внешняя программа-обработчик. ***Пример*** Фрагмент DTD <! ELEMENT Book EMPTY> <! ATTLIST Book image ENTITY #REQUIRED> <! NOTATION gif SYSTEM "gif-viewer.exe"> <! NOTATION jpg SYSTEM "jpg-viewer.exe"> <! ENTITY news SYSTEM "images/news.gif" NDATA gif> <! ENTITY products SYSTEM "images/products.jpg" NDATA jpg> <! ENTITY support SYSTEM "images/support.gif" NDATA gif> Фрагмент XML <Book image="news" /> <Book image="products" /> < Book image="support" /> В XML-документ будут вставлены соответствующие графические файлы

▶ Язык XML Schema Definition (XSD)

Язык XML Schema Definition (XSD) основан на XML и обладает более широкими возможностями описания структуры документа, чем DTD. Он поддерживает типизацию данных, пространства имен, регулярные выражения.

XML Schema содержит описание элементов и атрибутов XML-документа, правила наследования элементов, включая порядок и количество потомков, тип содержимого элементов, типы данных элементов и атрибутов, значения элементов и атрибутов и дополнительные ограничения на значения. Кроме того, использова-

ние XML Schema обеспечивает трансформацию XML-документа в иерархию объектов определенных типов, доступ к которым может быть осуществлен программным способом с помощью интерфейса (функциональность PSV1).

Основным преимуществом языка XML Schema является поддержка строго типизированных данных. При обмене данными между различными приложениями и базами данных задача согласования типов данных всегда остается актуальной, поскольку в разных системах определения типов данных могут отличаться. К таким отличиям относятся: максимальное и минимальное возможные значения, наибольшая длина, поддержка дробных чисел, внутренняя кодировка и внешний формат (например, для даты и времени). Таким образом, несмотря на возможное совпадение названий типов данных, их реализация в различных продуктах может отличаться. Применение типов данных в схемах позволяет проводить необходимую верификацию данных документа при обмене или совместном использовании данных несколькими системами.

Данные пособие не является подробным руководством по языку XML Schema, поэтому здесь мы ограничимся только базовыми сведениями о языке XSD, которые необходимы для понимания последующего материала.

XML Schema всегда создается в отдельном файле, имеющем расширение xsd. Файл XML связывается с соответствующей схемой с помощью атрибута schemaLocation пространства имен схемы. Для того чтобы использовать атрибут schemaLocation, необходимо определить пространство имен схемы. Все эти определения указываются в корневом элементе документа XML.

```
<КорневойЭлемент
     xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
     xs:schemaLocation="Схема.xsd">
```

Рассмотрим основные элементы структуры XML Schema.

Корневым элементом всегда является элемент <schema>. Описание атрибутов элемента <schema> приводится в табл. 2.10.

Таблица 2.10

Описание атрибутов элемента <schema>

Атрибут	Обязательный	Описание
elementFormDefault	Нет	Принимает значения «qualified» и «unqualified» (по умолчанию). Значение «qualified» указывет на то, что элементы документа должны уточняться префиксом пространства имен
attributeFormDefault	Нет	Принимает значения «qualified» и «unqualified» (по умолчанию). Значение «qualified» указывет на то, что атрибуты элементов документа должны уточняться префиксом пространства имен
xmlns: xs	Да	Всегда принимает значение xmlns: xs="http://www.w3.org/2001/XMLSchema" Указывает на пространство имен языка XMLSchema для элементов и типов данных схемы. При наличии префикса (в данном примере — xs) все элементы и типы данных схемы должны уточняться этим префиксом (xs: schema). Если префикс отсутствует, то атрибут xmlns указывает для схемы пространство имен по умолчанию. Элемент <schema> может содержать несколько атрибутов xmlns с разными префиксами, указывающими на используемые в документе пространства имен
targetNamespace	Нет	Пространство имен для элементов XML-документа, определенных данной схемой
version	Нет	Версия схемы
xml: lang	Нет	Язык для всех комментариев схемы

Корневой элемент <schema> может содержать следующие дочерние элементы:

- <element> — используется для определения элементов XML-документа;
- <attribute> — используется для определения атрибутов XML-документа;
- <group> — необходим для определения группы элементов, предназначенной для повторного использования в рамках схемы по ссылке на имя группы;

▪ <attributeGroup> — используется для определения атрибутов группы элементов;

▪ <annotation> — позволяет включать в XML-документ документацию;

▪ <import> — позволяет использовать компоненты указанной внешней схемы в основной схеме (обеспечивает модульность схем);

▪ <include> — добавляет все компоненты указанной внешней схемы в основную схему (обеспечивает модульность схем);

▪ < notation> — содержит определение нотации, описывающей формат не-XML-данных в XML-документе;

▪ <redefine> — переопределяет компоненты внешней схемы, имеющей то же пространство имен, что и основная схема;

▪ <simpleType> — объявляет простой тип содержимого элемента. Элементы с простым типом данных могут содержать только символьные данные и не могут включать атрибуты и дочерние элементы;

▪ <complexType> — объявляет сложный тип содержимого элемента, который может включать атрибуты и другие элементы.

XML Schema поддерживает три основные категории типов данных:

1) *предопределенные примитивные типы* — фундаментальные типы данных, на которые можно ссылаться и применять их к элементам и атрибутам. Примерами примитивных типов данных являются String, Float, Double, Time, Date, Decimal, AnyURI;

2) *предопределенные производные типы* — встроенные типы, полученные на основании примитивных типов. Примерами производных типов данных являются Integer, Long, Byte, Short, nonPositiveInteger, nonNegativeInteger, ID и др.;

3) *нестандартные типы* — определяемые пользователем типы данных, которые создаются на основании примитивных или производных типов путем введения дополнительных ограничений. Поддержка нестандартных типов данных исключительно полезна для верификации данных с учетом бизнес-логики.

Для описания элементов и атрибутов, имеющих предопределенные (примитивные и производные) типы данных[1], в XML Schema используются следующие синтаксические конструкции:

```
<xs:element name="ИмяЭлемента" type="ТипДанных" />
<xs:attribute name="ИмяАтрибута" type="ТипДанных"/>
```

[1] А также заранее определенные нестандартные типы данных (см. ниже).

Дополнительно для элементов и атрибутов можно указать атрибуты fixed или default для задания фиксированных значений элементов/атрибутов или значений по умолчанию.

```
<xs:element name="Пример" type="xs:string" default="Пример описания
элемента"/>
```

Если необходимо описать нестандартный тип данных для элемента или атрибута, то это следует делать с помощью тега <sympleType>, описав в нем новый тип данных.

```
<xs:element name="ИмяЭлемента">
     <xs:simpleType>
          Описание нестандартного типа данных
     </xs:simpleType>
</xs:element>
```

Новые нестандартные простые типы данных получают путем:

- сужения (restriction) встроенного или ранее определенного простого типа с помощью задания дополнительных ограничений;
- объединения (union) простых типов;
- использования списка (list) простых типов.

Пример использования нового простого типа данных, полученного путем сужения предопределенного типа (на базовый тип String накладываются ограничения на максимально и минимально допустимую длину строки):

```
<xs:simpleType>
              <xs:restriction base="xs:string">
                   <xs:maxLength value="20"/>
                   <xs:minLength value="10"/>
              </xs:restriction>
</xs:simpleType>
```

Пример использования нового простого типа данных, полученного путем объединения базовых типов (элемент или атрибут могут принимать неотрицательные или неположительные целые значения):

```
<xs:simpleType>
<xs:union memberTypes="xs:nonNegativeInteger
xs:nonPositiveInteger"/>
</xs:simpleType>
```

Пример использования списка простых типов (атрибут shoeSizes объявляется в качестве списка, содержащего десятичные значения 10.5, 9, 8 и 11):

```
<xs:simpleType name="Sizes">
         <xs:restriction base="xs:decimal">
              <xs:enumeration value="10.5"/>
              <xs:enumeration value="9"/>
              <xs:enumeration value="8"/>
              <xs:enumeration value="11"/>
         </xs:restriction>
</xs:simpleType>

     <xs:attribute name="shoeSizes">
          <xs:simpleType>
               <xs:list itemType="Sizes"/>
          </xs:simpleType>
     </xs:attribute>
```

Язык XML Schema использует различные типы ограничений на данные (см. табл. 2.8):

- ограничения длины (количество символов);
- границы значений (наибольшее и наименьшее значения как диапазон или порог);
- ограничения количества цифр десятичного числа (общее количество цифр или количество цифр после запятой);
- список допустимых значений;
- шаблоны;
- обработка символов пробела.

Примеры использования различных ограничений приведены в табл. 2.11.

Таблица 2.11

Примеры использования ограничений

Тип ограничения	Теги, задающие ограничения
Ограничения длины	<length> — фиксированное количество символов: <xs:simpleType> <xs:restriction base="xs:string"> <xs:length value="4"/> </xs:restriction> </xs:simpleType>

Продолжение таблицы 2.11

Тип ограничения	Теги, задающие ограничения

	<maxLength> — наибольшая длина значений определяемого типа:
	`<xs:simpleType>`
	`  <xs:restriction base="xs:string">`
	`  <xs:mfxLength value="20"/>`
	`  </xs:restriction>`
	`</xs:simpleType>`
	
	<minLength> — наименьшая длина значений определяемого типа:
	`<xs:simpleType>`
	`  <xs:restriction base="xs:string">`
	`  <xs:minLength value="2"/>`
	`  </xs:restriction>`
	`</xs:simpleType>`

Границы значений	<maxInclusive> — максимально допустимое значение,
	<minInclusive> — минимально допустимое значение
	(в примере допустимы целые значения в диапазоне от 5 до 10, включая 5 и 10):
	`<xs:simpleType>`
	`  <xs:restriction base="xs:integer">`
	`  <xs:maxInclusive value="10"/>`
	`  <xs:minInclusive value="5"/>`
	`  </xs:restriction>`
	`</xs:simpleType>`
	
	<maxExclusive> — максимально допустимое значение,
	<minExclusive> — минимально допустимое значение
	(в примере допустимы целые значения в диапазоне от 5 до 10, исключая 5 и 10):
	`<xs:simpleType>`
	`  <xs:restriction base="xs:integer">`
	`  <xs:maxExclusive value="10"/>`
	`  <xs:minExclusive value="5"/>`
	`  </xs:restriction>`
	`</xs:simpleType>`

Ограничения количества и типа цифр	<totalDigits> — общее количество цифр в определяемом числовом типе — сужении типа decimal,
	<fractionDigits> — количество цифр в дробной части числа:
	`<xs:simpleType>`
	`  <xs:restriction base="xs:decimal">`
	`  <xs:totalDigits value="8"/>`
	`  <xs:fractionDigits value="3"/>`
	`  </xs:restriction>`
	`</xs:simpleType>`

Окончание таблицы 2.11

Тип ограничения	Теги, задающие ограничения

Список допустимых значений	<enumeration> — элемент списка допустимых значений:
	`<xs:simpleType>`
	`  <xs:restriction base="xs:string">`
	`  <xs:enumeration value="один"/>`
	`  <xs:enumeration value="два"/>`
	`  <xs:enumeration value="три"/>`
	`  </xs:restriction>`
	`</xs:simpleType>`

Шаблоны	<pattern> — регулярное выражение, используемое для ограничения внешнего вида или формы значений данных:
	`<xs:simpleType>`
	`  <xs:restriction base="xs:string"`
	`  <xs:pattern value="[Ю|Ф][0-9]{3}"/>`
	`  </xs:restriction>`
	`</xs:simpleType`

Обработка символов пробела	<whitespace> — применяется при сужении типа string и определяет способ преобразования пробельных символов (пробел, табуляция, перевод строки).
	Используется в трех вариантах:
	1) все пробельные символы сохраняются:
	`<xs:simpleType>`
	`  <xs:restriction base="xs:string">`
	`  <xs:whiteSpace value="preserve"/>`
	`  </xs:restriction>`
	`</xs:simpleType>`
	2) XML-процессор заменяет все пробельные символы на пробелы:
	`<xs:simpleType>`
	`  <xs:restriction base="xs:string">`
	`  <xs:whiteSpace value="replace"/>`
	`  </xs:restriction>`
	`</xs:simpleType>`
	3) XML-процессор заменяет пробельные символы пробелами, затем убирает начальные и конечные пробелы, а из нескольких подряд идущих пробелов оставляет только один:
	`<xs:simpleType>`
	`  <xs:restriction base="xs:string">`
	`  <xs:whiteSpace value="collapse"/>`
	`  </xs:restriction>`
	`</xs:simpleType>`

Элементы, имеющие простой тип или предопределенные стандартные типы, могут содержать только данные (не могут содержать атрибутов и дочерних элементов).

Любой простой тип данных может содержать произвольный набор ограничений, который определяется бизнес-логикой приложения, работающего с данными.

Если простому типу данных присвоено имя, то ссылка на новый нестандартный тип данных может быть использована многократно в пределах данной схемы (аналогично ссылке на предопределенные типы данных).

```
<xs:simpleType name="Код">
     <xs:restriction base="xs:string">
          <xs:length fixed="true" value="4"/>
          <xs:pattern value="[Ю|Ф][0-9]{3}"/>
     </xs:restriction>
</xs:simpleType>
<xs:element name="Код1" type="Код"/>
<xs:element name="Код2" type="Код"/>
```

В данном примере определен нестандартный тип данных с именем «Код», базирующийся на типе «string»: он использован как тип данных для элементов «Код1» и «Код2».

Для описания элементов XML-документа, содержащих дочерние элементы и атрибуты, в схеме используется сложный тип данных, который задается с помощью тега <complexType>.

```
<xs:element name="ИмяЭлемента">
     <xs:complexType>
          Описание сложного типа данных
     </xs:complexType >
</xs:element>
```

При описании сложного типа указываются порядок вхождения дочерних элементов (с помощью специальных тегов — индикаторов порядка, см. табл. 2.11), а также степень кардинальности повторяющихся элементов (с использованием атрибутов minOccurs и maxOccurs).

Атрибут minOccurs определяет минимальную степень кардинальности, то есть наименьшее возможное количество повторений

дочернего элемента. Значение minOccurs, равное нулю, указывает на необязательность (опциональность) элемента.

Атрибут maxOccurs определяет максимальную степень кардинальности, или наибольшее количество повторений элемента. Максимальная и минимальная степени кардинальности задаются определенными значениями. Для maxOccurs может быть указано значение unbounded (элемент встречается любое количество раз).

```
<xs:element name="Книга">
     <xs:complexType>
          <xs:sequence>
               <xs:element name="Название" type="xs:string"/>
               <xs:element name="Автор" type="xs:string" maxOccurs="4"/>
               <xs:element name="Код" type="xs:string"/>
               <xs:element name="Цена" type="xs:string"/>
          </xs:sequence>
     </xs:complexType>
</xs:element>
```

В данном примере описан сложный тип данных для элемента «Книга», содержащего дочерние элементы «Название», «Автор», «Код», «Цена». Тег <sequence> является индикатором порядка вхождения дочерних элементов (табл. 2.12), а атрибут maxOccurs показывает максимально допустимое количество повторений элемента «Автор».

```
<xs:complexType name="Цена">
     <xs:choice>
          <xs:element name="Рубли" type="xs:double"/>
          <xs:element name="Доллары" type="xs:double"/>
     </xs:choice>
</xs:complexType >
```

Таблица 2.12

Теги индикатора порядка

Тег индикатора порядка	Описание
sequence	Дочерние элементы должны встречаться в указанном порядке. Степень кардинальности определяет, может ли дочерний элемент повторяться
all	Все дочерние элементы должны присутствовать в XML-документе. Дочерние элементы могут появляться в любом порядке, но должны встречаться только один раз. Нельзя задать значение степени кардинальности maxOccurs и minOccurs, отличное от единицы
choice	Из всех указанных дочерних элементов должен встречаться только один

Индикатор порядка choice указывает, что элемент этого типа «Цена» может содержать либо элемент «Рубли», либо элемент «Доллары», но не оба.

▸ Язык RELAX NG

Язык RELAX NG также использует XML-синтаксис и является упрощенным вариантом XML Schema, сочетает в себе простоту DTD и возможности XML Schema. RELAX NG, как и XML Schema, поддерживает типизацию данных, регулярные выражения, пространства имен и возможность ссылаться на сложные определения.

2.4.5. Программная обработка XML-документов

Приложения, работающие с XML-документами, получают доступ к их содержимому и структуре путем использования специального программного компонента — XML-процессора (XML-анализатора). Следует отметить, что приложения работают не непосредственно с XML-документом, а с его информационным пространством XML Infoset, получаемым в результате разбора XML-документа XML-процессом. Таким образом, XML-процессор предназначен для анализа XML-документа, извлечения данных и передачи их на обработку в прикладную программу. XML-процессоры поддерживают механизм XML Namespace (спецификация W3C XML Namespaces 1.0) и обеспечивают проверку структурной и синтаксической корректности XML-документов.

▸ XML-процессоры

Обрабатывая XML-документ, XML-процессоры представляют его структуру в виде упорядоченной модели данных, доступ к которой осуществляется с помощью стандартных интерфейсов прикладного программирования. Существуют два основных типа XML-процессоров — объектные (DOM) и потоковые (SAX).

Объектный анализатор строит в собственном пространстве памяти иерархическую модель разбираемого документа (Document Object Model, DOM), доступ к элементам которой прикладная программа получает с помощью DOM-интерфейсов. Основным преимуществом объектного анализатора является то, что он сразу предоставляет прикладной программе всю структуру XML-документа, позволяя ее анализировать в произвольном порядке.

Потоковый процессор является событийно управляемым и анализирует документ последовательно в режиме реального времени, что позволяет существенно экономить ресурсы памяти. Для доступа к элементам структуры документа прикладная программа в этом случае использует SAX (Simple API for XML). Следует отметить, что использование потоковых анализаторов утяжеляет логику прикладных программ и усложняет навигацию по документу.

Обобщенная схема работы XML-процессора представлена на рис. 2.4.

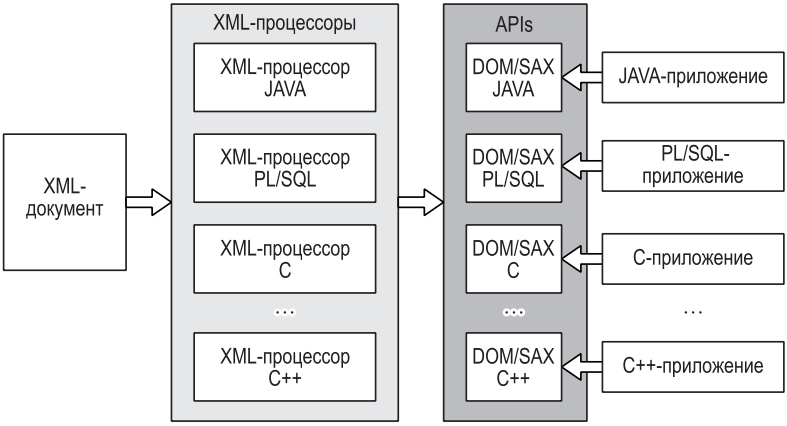

Рис. 2.4. Обобщенная схема работы XML-процессора

Программная обработка XML-документов выполняется также с использованием XSL Transformation (XSLT) процессоров, предназначенных для преобразования XML-документов в документы другой структуры и формата (XML, HTML, текстовый формат и т.д.).

XSLT-процессоры соответствуют спецификациям W3C XSL Transform Proposed Recommendation 1.0 и Xpath Proposed Recommendation 1.0, поддерживают различные языковые и программные платформы.

2.4.6. Компонентные модели структуры XML-документов

Модульность XML-документов, позволяющая объединять данные из разных источников и собирать документы и их схемы из имеющихся блоков, широко используется в интеграционных проектах.

Так, при построении модели данных SOA-проектов, как правило, создается несколько XML-схем, основанных на концепции повторного использования. Общая схема (мастер-схема) собирается из модульных подсхем, которые можно повторно использовать в разных контекстах и различных приложениях.

Например, схему Person.xsd можно использовать в контекстах имени сотрудника, имени покупателя или имени представителя поставщика. В результате достигается снижение затрат на проектирование и разработку.

Пусть схема Person.xsd имеет вид:

```
<?xml version="1.0"?>
<xsd:schema xmlns: xsd=http://www.w3.org/2001/XMLSchema
targetNamespace="http://www.company.org"
xmlns=http://www.person.org elementFormDefault="qualified">
     <xsd:complexType name="PersonType">
          <xsd:sequence>
               <xsd:element name="Name" type="xsd:string"/>
               <xsd:element name="SSN" type="xsd:string"/>
          </xsd:sequence>
     </xsd:complexType>
</xsd:schema>
```

Мастер-схема Company.xsd использует механизм include для ссылки на модульную подсхему:

```
<?xml version="1.0"?>
<xsd:schema xmlns: xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.company.org" xmlns="http://www.company.org"
elementFormDefault="qualified">
<xsd:include schemaLocation="Person.xsd"/>
     <xsd:element name="Company">
          <xsd:complexType>
               <xsd:sequence>
     <xsd:element name="Person" type="PersonType" maxOccurs="unbounded"/>
               </xsd:sequence>
          </xsd:complexType>
     </xsd:element>
</xsd:schema>
```

Синтаксис схемы позволяет определить в подсхеме элементы, группы, сложные и простые типы данных, которые можно под-

ключить и на которые можно сослаться из другой схемы. В виде повторно используемой подсхемы обычно определяют стандартные для предприятия типы данных, международные и отраслевые коды, допустимые значения, устойчивые структуры данных (типы документов, блоки адресов, структуры продуктов и др.).

Уровень вложенности ссылок на подсхемы не ограничен. Ссылка на подсхему может иметь одну из следующих форм:

- включение (include) — в этом случае включаемая подсхема становится частью пространства имен основной схемы. Данный вариант используется, если основная схема имеет единый контекст;
- импорт (import) — используется, когда основная схема относится к нескольким контекстам, и ссылка на подсхему должна сохранить уникальность пространства имен подсхемы;
- переопределение (redefine) — включает в мастер-схему подсхему и позволяет переопределить некоторые конструкции.

```
<?xml version="1.0"?>
<xsd:schema xmlns: xsd=http://www.w3.org/2001/XMLSchema
targetNamespace=http://www.company.org xmlns=http://www.company.org
elementFormDefault="unqualified" xmlns:per=http://www.person.org
xmlns: pro="http://www.product.org">
     <xsd:import namespace=http://www.person.org schemaLocation=
     "Person.xsd"/>
     <xsd:import namespace=http://www.product.org
     schemaLocation="Product.xsd"/>
     <xsd:element name="Company">
          <xsd:complexType>
               <xsd:sequence>
               <xsd:element name="Person" type="per:PersonType"
               maxOccurs="unbounded"/>
               <xsd:element name="Product" type="pro:ProductType"
               maxOccurs="unbounded"/>
               </xsd:sequence>
          </xsd:complexType>
     </xsd:element>
</xsd:schema>
```

Модульные схемы широко используются в электронной коммерции при обмене данными между информационными системами предприятий-партнеров, а также в системах автоматизации государственного управления для решения задач централизованного сбора данных и межведомственного обмена информацией. Наглядным примером

могут служить унифицированные форматы электронных банковских сообщений для безналичных расчетов, используемые для обмена с кредитными организациями и другими клиентами Банка России [30].

2.4.7. Язык запросов XSLT

XSLT (Extensible Stylesheet Language Transformations) — это расширяемый язык стилевых преобразований, предназначенный для описания способа преобразования документов XML в документы других форматов. С помощью XSLT можно трансформировать XML-документ в любой формат (HTML, TXT, RTF, PDF и т.п.), а также в XML-документ другой структуры. Последняя возможность широко используется при передаче информации между разными информационными системами, оперирующими с одной и той же информацией в различном представлении.

Язык XSLT основан на XML. Все элементы языка XSLT относятся к пространству имен http://www.w3.org/1999/XSL/Transform (обычно они имеют префикс xsl). На языке XSLT записывается инструкция по обработке XML-документа, предназначенная для XSLT-процессора. На сегодняшний день известны две спецификации языка — XSLT 1.0 и XSLT 2.0.

Инструкции по обработке, как правило, сохраняют в отдельном файле с расширением xslt, в этом случае ссылку на таблицу стилей помещают в документ XML как одну из инструкций по обработке.

Приведем пример ссылки:

```
<?xml version="1.0" encoding="windows-1251"?>
<?xml-stylesheet type="text/xsl" href="simple.xslt"?>
<Корневой элемент>
     <!-- содержание -->
</КорневойЭлемент>
```

XSLT-файл содержит описание правил преобразования исходного дерева элементов в конечное дерево. Преобразование строится путем сопоставления *образцов* и *шаблонов*. Образец сравнивается с элементами исходного дерева, а шаблон используется для создания частей конечного дерева. Структура конечного дерева может полностью отличаться от структуры исходного дерева.

XSLT-процессор выполняет преобразование, заданное в указанном XSLT-файле. XSLT совместно с входным XML-документом интерпретируется процессором, и на выходе получается документ другого формата.

Обычно XSLT используется совместно с XPath (XML Path Language) — языком запросов к элементам XML-документа. Запросы предназначены для организации доступа к узлам в XML-документе, осуществляют навигацию по DOM в XML.

XSLT-файл имеет следующую структуру:

1) обязательный корневой элемент stylesheet:

```
<xsl:stylesheet version="1.0" xmlns:xsl=http://www.w3.org/1999/XSL/
    Transform>
    <Описание преобразования>
</xsl:stylesheet>
```

2) элементы шаблона, предназначенные для поиска и преобразования отдельных элементов документа;

3) конструкции для сериализации данных в выходной документ.

Описание основных элементов языка XSLT представлено в табл. 2.13.

Таблица 2.13

Описание основных элементов языка XSLT

Элемент	Описание
xsl:output	Элемент, управляющий форматом преобразованных данных. Обычно этот элемент объявляется сразу после элемента xsl:stylesheet. Обязательный атрибут method определяет тип выходного значения для таблицы стилей XSLT и может принимать одно их трех значений: xml, html или text. Пример преобразования XML в XML: <xsl:output method="xml" indent="yes"/>
xsl:template	Основной элемент языка. Используется для записи шаблонов преобразования. Корневой элемент xsl:stylesheet должен иметь хотя бы один дочерний элемент xsl:template. В содержимом элемента xsl:template задается конструктор последовательности преобразованных узлов XML-документа. xsl:template имеет следующие атрибуты:

Продолжение таблицы 2.13

Элемент	Описание

	▪ match — обязательный атрибут, задает последовательность исходных узлов, для преобразования которых предназначен шаблон. Представляет собой выражение XPath, выделяющее узлы документа XML. Шаблон выполняется, когда обнаруживает совпадение (match);
	▪ name — необязательный атрибут, используется элементом xsl:call-template для ссылки на шаблон;
	▪ priority –необязательный атрибут, указывает порядок выполнения шаблонов, соответствующих одному и тому же выражению XPath;
	▪ mode — необязательный атрибут. С его помощью можно сопоставлять шаблоны на основе значения, указанного в атрибуте mode, и выражения, указанного в выражении match.
	`<xsl:template match="/book">`
	`  <Последовательность преобразованных узлов>`
	`</xsl:template>`
	Шаблон выполняется для каждого узла <book> в документе XML

xsl:value-of	Используется для вывода значения узла. Значение выводится в виде строки. xsl:value-of представляет собой пустой элемент и не имеет никаких дочерних элементов. Выходное значение задается в обязательном атрибуте select этого элемента как выражение XPath:
	`<xsl:template match="name">`
	`  <xsl:value-of select="."/>`
	`</xsl:template>`
	Шаблон выполняется для элемента name. В выходной документ выводится значение элемента name

xsl:apply-templates	Задается чаще всего внутри элемента xsl:template, предписывает обработать рекурсивно узлы — потомки узлов, отобранных родительским элементом xsl:template:
	`<xsl:template match="/">`
	`  <xsl:apply-templates/>`
	`</xsl:template>`
	Пример рекурсивной обработки всех потомков корневого элемента XML-документа.
	Необязательный атрибут select ограничивает обработку только указанными в нем узлами. Значение атрибута задается как выражение XPath:
	`<xsl:template match="product">`
	`<xsl:apply-templates select="name"/>`
	`</xsl:template>`
	`  <xsl:template match="name">`
	`  <xsl:value-of select="."/>`
	`</xsl:template>`

Продолжение таблицы 2.13

Элемент	Описание

xsl:if	Условный оператор, управляющий преобразованием. Конструктор преобразования задается внутри тега xsl:if и запускается только в том случае, если истинно выражение, указанное в атрибуте test:
	`<xsl:template match="name">`
	`  <xsl:if test="@country="Russia"">`
	`  <xsl:value-of select="."/>`
	`  </xsl:if>`
	`</xsl:template>`
	В выходной документ выводится значение элемента name только в том случае, если значение атрибута country равно Russia

xsl:for-each	Элемент используется для многократной обработки набора выбранных узлов. В атрибуте select задается выражение XPath, позволяющее отобрать необходимые узлы. После того как узлы выбраны, для каждого из них выполняются операторы, указанные в элементе xsl:for-each:
	`<xsl:for-each select="Контрагенты/Контрагент">`
	`  <tr>`
	`  <td><xsl:value-of select="Код"/></td>`
	`  <td>`
	`<xsl:value-of select="Наименование"/>`
	`  </td>`
	`  </tr>`
	`</xsl:for-each>`
	В выходной HTML-документ выводится таблица, содержащая столько строк, сколько элементов <Контрагент> содержит родительский тег <Контрагенты>

xsl:sort	Элемент используется для сортировки xml-тегов. Элемент должен размещаться внутри элементов xsl:apply-templates или xsl:for-each. Для задания атрибута или элемента, по которому выполняется сортировка, используется атрибут select. Для определения порядка сортировки используется атрибут order:
	`<xsl:for-each select="Контрагенты/Контрагент">`
	`<xsl:sort select="Наименование" order="ascending"/>`
	`  <tr>`
	`  <td><xsl:value-of select="Код"/></td>`
	`  <td>`
	`<xsl:value-of select="Наименование"/>`
	`  </td>`
	`  </tr>`
	`</xsl:for-each>`
	В выходной HTML-документ выводится таблица, строки которой отсортированы по наименованию контрагента (сортировка по возрастанию)

Окончание таблицы 2.13

Элемент	Описание
xsl:import	Позволяет повторно использовать шаблоны, определенные в других таблицах стилей. Записывается в корневом элементе xsl:stylesheet. Элементов xsl:import может быть несколько: `xsl:import href="URI Адрес Внешнего XSLT"/>` XSLT-процессор отыскивает внешнюю таблицу стилей и подставляет ее на место элемента xsl:import перед преобразованием. Если правила преобразования из внешних файлов XSLT конфликтуют с правилами, определенными в самой таблице стилей, то применяются те правила, которые записаны последними, поэтому важен порядок записи элементов xsl:import в таблицу стилей
xsl:include	Позволяет повторно использовать шаблоны, определенные в других таблицах стилей: `<xsl:include href="URI Адрес Внешнего XSLT"/>` Его можно записать не только в корневом элементе xsl:stylesheet, но и в любом месте таблицы стилей. Порядок записи элементов xsl:include в таблице стилей не имеет значения

В табл. 2.14 представлены выражения XPath, используемые в качестве значений атрибутов match и select.

Таблица 2.14

Выражения XPath, используемые в качестве значений атрибутов match и select

Выражение XPath	Описание
/	Корневой узел
.	Текущий узел. Может быть записано как self::node ()
..	Родительский узел текущего узла. Может быть записано как parent::node ()
Products	Узел Products
Products / Product1	Подузел Product1 узла Products
Product/*	Все потомки узла Product
/Product	Узел Product, являющийся прямым потомком корневого узла
@name	Атрибут name текущего узла
@*	Все атрибуты текущего узла
Product @ Price	Атрибут Price узла Product
Products / Product @ Price	Атрибут Price узла Product, являющегося подузлом узла Products
..@ Price	Атрибут Price родительского узла
//	Любое количество промежуточных узлов

Окончание таблицы 2.14

Выражение XPath	Описание
Products // Product	Все узлы Product, имеющие предка Products
|	Знак разделения конкретных узлов
Product | Product2	Узел Product1и узел Product2
[]	Предикатное выражение
Products [Product]	Узел Products, имеющий потомка Product
Products [Products="qq"]	Узел Products, имеющий потомка Product, значение которого равно qq
Product [@Price]	Узел Product, имеющий атрибут Price
Product [@Price ="5"]	Узел Product, имеющий атрибут Price, значение которого равно 5
count (Product/*)	Количество потомков узла Product
name ()	Имя текущего узла

Рассмотрим практический пример использования языка XSLT. Пусть необходимо связать информационную систему, выполняющую регистрацию заказов клиентов (система *A*), с системой обработки заказов (система *B*), причем внутренние форматы данных в этих системах существенно отличаются.

Простейшей технологией, обеспечивающей слабое связывание приложений и создающей основу для эффективного управления изменениями, является передача сообщений. Пусть система *A* передает сообщение о заказе клиента следующего формата:

```
<Order>
     <date>01/11/2013</date>
     <OrderNumber>4554654</OrderNumber>
     <Customer>
          <Id>123</Id>
          <Name>Smith</Name>
     </Customer>
     <OrderItems>
          <Item>
               <Quantity>10</Quantity>
               <ItemNo>444</ItemNo>
               <Discription>IPhone</Discription>
          </Item>
          <Item>
               <Quantity>20</Quantity>
               <ItemNo>555</ItemNo>
               <Discription>IPad</Discription>
          </Item>
     </OrderItems>
</Order>
```

Система *B* готова принять и обработать заказ клиента в следующем формате:

```
<?xml version="1.0" encoding="utf-8"?>
<OrderItems>
     <OrderItem>
          <date>01/11/2013</date>
          <OrderNumber>4554654</OrderNumber>
          <CustomerID>123</CustomerID>
          <Quantity>10</Quantity>
          <ItemNo>444</ItemNo>
          <Discription>IPhone</Discription>
     </OrderItem>
     <OrderItem>
          <date>01/11/2013</date>
          <OrderNumber>4554654</OrderNumber>
          <CustomerID>123</CustomerID>
          <Quantity>20</Quantity>
          <ItemNo>555</ItemNo>
          <Discription>IPad</Discription>
     </OrderItem>
</OrderItems>
```

То есть заказ клиента, содержащий несколько позиций, должен быть разбит на части, каждая из которых дублирует информацию о дате заказа, номере заказа и идентификаторе заказчика. На рис. 2.5 показана логика преобразования данных в промежуточном слое интеграционного решения.

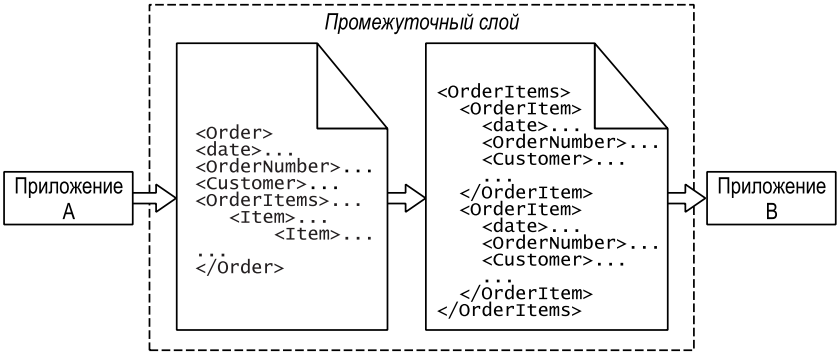

Рис. 2.5. Преобразование данных

Одним из способов подобного преобразования является использование преобразования XSLT. Следующий XSLT-код выполнит преобразование исходного XML-документа в требуемый формат:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/
XSL/Transform"
     xmlns:msxsl="urn: schemas-microsoft-com:xslt" exclude-result-
     prefixes="msxsl">
     <xsl:output method="xml" indent="yes"/>
     <xsl:template match="Order">
     <OrderItems>
          <xsl:apply-templates select="OrderItems/Item"/>
     </OrderItems>
     </xsl:template>
     <xsl:template match="Item">
          <OrderItem>
               <date>
                    <xsl:value-of select="parent::node()/
                    parent::node()/date"/>
               </date>
               <OrderNumber>
                    <xsl:value-of select="parent::node()/
                    parent::node()/OrderNumber"/>
               </OrderNumber>
               <CustomerID>
                    <xsl:value-of select="parent::node()/
                    parent::node()/Customer/Id"/>
               </CustomerID>
               <xsl:apply-templates select="*"/>
          </OrderItem>
     </xsl:template>
     <xsl:template match="*">
          <xsl:copy>
               <xsl:apply-templates select="@*|node()"/>
          </xsl:copy>
     </xsl:template>
</xsl:stylesheet>
```

Преобразование находит в исходном документе корневой элемент Order, после чего создает корневой элемент <OrderItems> и начинает обрабатывать все элементы <Item>, найденные внутри

элемента <OrderItems>. Для каждого из элементов <Item> создается новый шаблон, который копирует элементы date, OrderNumber и CustomerID из элемента Order исходного документа, находящегося на два уровня выше, и присоединяет к ним все остальные элементы, вложенные в Item.

Альтернативным вариантом является реализация программного разбора исходного XML-документа с копированием необходимых тегов в результирующий документ. Такое решение намного сложнее поддерживать в случае возможных изменений формата результирующего документа. Кроме того, практика показывает, что XSLT-преобразование выполняется намного быстрее, чем перенос элементов из исходного документа в результирующий программным путем.

2.4.8. Web-сервисы

Технология Web-сервисов разрабатывалась как основанная на открытых стандартах технология взаимодействия между приложениями. В отличие от плохо сочетаемых друг с другом стандартов удаленного вызова (DCOM, CORBA, RMI), технология Web-сервисов полностью снимает проблемы взаимодействия приложений, созданных и функционирующих на разных платформах, и позволяет строить хорошо масштабируемые слабо связанные интеграционные решения.

Web-сервисы могут выполнять два вида функций:

1) обмен данными между различными компонентами распределенной системы или несколькими взаимодействующими приложениями;

2) предоставление разнообразных сервисов (выполнение различных бизнес-функций).

▶ Понятие Web-сервиса и его характеристики

Web-сервисы — это программные компоненты, взаимодействие с которыми осуществляется по сети Интернет с использованием открытых протоколов [14, 13, 25, 34, 23].

Каждый Web-сервис характеризуется URI-адресом и набором функций, которые могут быть вызваны удаленно. Web-сервис не предназначен для обслуживания конечных пользователей, он

предоставляет услуги другим приложениям — *клиентам Web-сервисов*, поэтому у Web-сервиса нет пользовательского интерфейса — он имеет программный интерфейс, описанный в формате, пригодном для компьютерной обработки.

Интерфейс Web-сервиса определяет его абстрактную функциональность, а конкретная реализация интерфейса Web-сервиса обеспечивает отправку, обработку и получение сообщений. Обычно интерфейс Web-сервиса описывается в WSDL-формате. WSDL-описание Web-сервиса определяет формат пересылаемых сообщений, тип передаваемых данных, протокол передачи сообщений и адреса доставки сообщений.

Кроссплатформенность технологии Web-сервисов обеспечивается XML-форматом их описания и XML-форматом сообщений, отправляемых и получаемых Web-сервисами.

Взаимодействие с Web-сервисами осуществляется при помощи сообщений, передаваемых с использованием открытых протоколов, чаще всего протокола HTTP.

Таким образом, Web-сервис обладает следующими характеристиками:

- поддерживает единые методы обращения к нему;
- интерфейс Web-сервиса не зависит от платформы;
- способы внутренней реализации Web-сервиса обычно неизвестны инициатору запроса (клиенту сервиса). Web-сервис может быть реализован практически на любом языке программирования;
- Web-сервис является повторно используемым компонентом.

Несмотря на то что название «Web-сервис» подразумевает использование сервиса в сети Интернет, Web-сервисы могут работать во внутренней сети предприятия, обеспечивая связь между прикладными программами (бизнес-функция одного приложения становится доступна другим приложениям).

Web-сервисы могут обслуживать запросы на доступ к данным, преобразование данных или обмен данными. В этом случае Web-сервис обращается к источнику данных, выбирает данные, преобразует их и перемещает от источника к получателю. При перемещении данных важно правильно перевести данные из одного формата в другой, то есть отобразить метаданные источника на метаданные получателя. Для описания метаданных широко используются XML-схемы, реализованные в виде общих словарей предприятия.

Следует понимать, что построение интеграционного решения на основе Web-сервисов не всегда является оптимальным решением. Главными недостатками Web-сервисов являются меньшая производительность и больший размер сетевого трафика по сравнению с такими технологиями, как RMI, CORBA, DCOM, за счет использования текстовых XML-сообщений. Технологию Web-сервисов целесообразно применять для объединения компонентов, работающих в распределенной среде на различных платформах или в том случае, если потребитель Web-сервиса заранее неизвестен.

▶ Классификация Web-сервисов

Классификация Web-сервисов может быть выполнена на основе следующих критериев [19]:

- по типу взаимодействия с клиентом;
- по типу клиента Web-сервиса;
- по модели обработки запроса Web-сервисом.

На основе модели обработки клиентского запроса выделяют следующие типы Web-сервисов:

- метод-ориентированные;
- документ-ориентированные;
- ресурс-ориентированные.

Метод-ориентированные Web-сервисы (или RPC-Web-сервисы) — это Web-сервисы, взаимодействие с которыми производится по протоколу SOAP (Simple Object Access Protocol) с использованием XML-сообщений. Интерфейс метод-ориентированных Web-сервисов описан в формате WSDL. Такое описание интерфейса сервиса обеспечивает автоматическую генерацию кода, необходимого для связи с сервисом, на стороне клиента. SOAP-протокол может использовать различные транспортные протоколы (HTTP, FTP, SMTP). SOAP-сообщения, участвующие в обмене между клиентом и Web-сервисом, имеют строго определенную структуру и передают имя и параметры удаленно вызываемой процедуры, а также результат вызова.

Недостатком подобного подхода является достаточно тесная связь между клиентом и Web-сервисом (клиент должен знать все детали интерфейса Web-сервиса, при изменении имени или параметров вызываемой процедуры необходимо изменять клиентов).

Документ-ориентированные Web-сервисы — это XML-Web-сервисы, ориентированные на сообщения. Они обеспечивают низкоуровневую обработку XML-сообщений, содержание которых не связано с процедурами интерфейса, а полностью определяется XML-схемой. При получении сообщения от клиента Web-сервис обрабатывает полученные данные целиком и формирует ответное сообщение. XML-Web-сервисы могут принимать и передавать сообщения как в формате SOAP, так и в чистом XML-формате.

XML-Web-сервисы создают более слабую связь между клиентом и Web-сервисом, так как возможные изменения затрагивают XML-схему, а не WSDL-описание сервиса. Однако при этом увеличивается сложность реализации Web-сервиса, поскольку требуется проводить анализ структуры сообщения.

Ресурс-ориентированные Web-сервисы — это RESTful-Web-сервисы, предоставляющие доступ к удаленным ресурсам с помощью HTTP-запросов. Сервисы данного типа основаны на архитектуре REST (Representational State Transfer) и обеспечивают взаимодействие с удаленными ресурсами, передавая клиенту их представление. RESTful-Web-сервисы поддерживают четыре операции — GET, PUT, POST и DELETE, с помощью которых производятся запрашиваемые клиентом действия с ресурсами. Технология REST Web-сервисов также может использовать WSDL-описание и SOAP-протокол для передачи сообщений, но может обходиться и без них.

Архитектура REST упрощает анализ клиентского запроса, поскольку HTTP-методы запроса напрямую связываются с операциями Web-сервиса. Недостаток технологии заключается в небольшом количестве доступных операций и ограничении «один ресурс — один URL-адрес».

▸ Спецификация WSDL

Спецификация WSDL (Web Service Definition Language) определяет язык описания сервиса и обеспечивает способ описания технической информации о Web-сервисе и его интерфейсе, включая имя и адрес сервиса, способ и протокол взаимодействия с сервисом, форматы сообщений, функциональность сервиса. В настоящий момент существуют спецификации WSDL 1.1 и WSDL 2.0. Подробнее о спецификациях см. [14].

▸ Протоколы передачи данных

Для обеспечения кроссплатформенности сервиса запрос, посылаемый клиентом Web-сервису, и ответ, который сервис возвращает инициатору запроса, представляет собой сообщение XML-формата. SOAP (Simple Object Access Protocol) — это протокол обмена сообщениями в распределенной среде, то есть соглашение о структуре документов, участвующих в обмене сообщениями. SOAP-протокол может использоваться поверх разнообразных транспортных протоколов для передачи символьной и двоичной информации. В настоящий момент существуют спецификации SOAP 1.1 и SOAP 1.2. Подробнее о протоколе см. [13].

▸ Типы взаимодействия с клиентом

Web-сервисы поддерживают синхронную и асинхронную модели взаимодействия с клиентом. В случае синхронного взаимодействия после отправки запроса Web-сервису клиент блокируется до получения ответа. При асинхронном взаимодействии работа клиента не блокируется, обработка ответа производится после его получения.

В рамках синхронной и асинхронной моделей могут быть реализованы следующие сценарии взаимодействия клиента с Web-сервисом [23]:

- однонаправленный запрос — клиент посылает сообщение одному или нескольким Web-сервисам, которые его обрабатывают, но не генерируют ответ;
- синхронный запрос-ответ — синхронный обмен сообщениями между клиентом и Web-сервисом;
- асинхронный запрос-ответ — асинхронный обмен сообщениями между клиентом и Web-сервисом;
- RPC-вызов — клиент вызывает процедуру интерфейса Web-сервиса путем сериализации параметров процедуры в сообщение и отправки его Web-сервису;
- сообщение об ошибке — в случае, если выполнение процедуры интерфейса Web-сервиса завершается с ошибкой, клиент получает сообщение об ошибке;

▪ запрос с подтверждением — клиент получает от Web-сервиса сообщение об успешной или неуспешной доставке запроса;

▪ передача через посредников — клиентское сообщение передается целевому Web-сервису через посредников (промежуточные Web-сервисы);

▪ кеширование — в случае схожего клиентского запроса ответ ему посылается из кеша;

▪ дополнительная функциональность — при обмене сообщениями может быть дополнительно реализованы шифрование данных, аутентификация пользователей, поддержка транзакций и т.п.

▸ Клиенты Web-сервисов

Клиентами, пользующимися услугами Web-сервисов, могут быть приложения двух типов:

1) клиентские приложения, взаимодействие которых с Web-сервисом инициирует конечный пользователь (такие приложения могут быть как «тонким», так и «толстым» клиентом);

2) программные компоненты, автоматически совершающие запрос к Web-сервису (без участия пользователя).

▸ Репозитории Web-сервисов

Если Web-сервис предоставляет услуги для широкого круга заранее неизвестных потребителей, то он должен быть зарегистрирован в каком-либо из открытых репозиториев (реестров), чтобы потенциальные пользователи смогли его отыскать.

При использовании Web-сервисов в целях интеграции корпоративных приложений создаются специальные закрытые реестры, позволяющие осуществлять поиск нужного Web-сервиса.

Наиболее распространенными стандартами для создания корпоративных реестров являются UDDI, ebXML и DISCO.

UDDI (Universal Discription, Discovery and Integration) — стандарт для создания платформонезависимого, основанного на XML реестра Web-сервисов, позволяющего публиковать и находить их WSDL-описание. UDDI-реестр хранит информацию о поставщиках, функциональности и деталях реализации Web-сервисов.

UDDI-реестр логически разделен на три типа каталогов:

1) белые страницы — содержат информацию о поставщиках Web-сервисов (имя поставщика, описание, контактная информация);

2) желтые страницы — классифицируют Web-сервисы по сферам применения;

3) зеленые страницы — содержат информацию о деталях реализации Web-сервисов, необходимую для доступа к ним.

Данные каталогов хранятся в виде XML-файлов, структура которых определяется моделью данных UDDI. Каждый XML-файл с данными имеет уникальный UDDI-идентификатор и структуру, определяемую XML-схемами. Администрирование UDDI-реестров осуществляется с помощью специальных приложений UDDI-серверов, которые, по сути, представляют собой набор Web-сервисов для поиска, публикации, хранения и репликации данных реестра. Примером UDDI-сервера с открытым исходным кодом является http://ws.apache.org/juddi.

ebXML (Electronic Business using eXtensible Markup Language) — основанный на XML стандарт, обеспечивающий инфраструктуру для создания единого электронного рынка. Стандарт позволяет создавать бизнес-документы и описания бизнес-процессов, хранить их в едином реестре и предоставлять информацию для поиска бизнес-партнеров.

ebXML-реестр может быть создан как для отдельной корпорации, так и для предприятий отдельной отрасли. Примеры реализации технологии ebXML можно найти на официальном сайте проекта http://www.ebxml.org.

DISCO — это технология компании Microsoft, предназначенная для публикации и поиска Web-сервисов. Суть технологии заключается в том, что в определенном каталоге сервера, на котором развернут Web-сервис, создается XML-документ определенной структуры (DISCO-документ), содержащий ссылку на WSDL-описание Web-сервиса. Поиск сервисов осуществляется с помощью специальной утилиты, генерирующей файл с результатами поиска, на основании которого автоматически создаются клиентские заглушки Web-сервиса.

Обобщенная схема взаимодействия поставщика сервиса, клиента сервиса и репозитория сервисов представлена на рис. 2.6.

Рис. 2.6. Взаимодействие поставщика и клиента Web-сервиса с репозиторием сервисов

2.4.9. Оркестровка и хореография Web-сервисов

Функционирование интеграционных решений, использующих Web-сервисы (SOA-сценарий взаимодействия приложений), основано на выполнении бизнес-процессов путем последовательного взаимодействия с рядом Web-сервисов.

Существует два подхода к описанию бизнес-процессов и интеграции Web-сервисов — это оркестровка и хореография.

Оркестровка Web-сервисов — это описание корпоративного бизнес-процесса в виде набора взаимодействий внутренних и внешних Web-сервисов. Контроль над бизнес-процессом осуществляется данной компанией с помощью центрального координатора, который в общем случае также может быть реализован как Web-сервис. Центральный координатор управляет взаимодействием Web-сервисов и определяет порядок выполнения ими необходимых функций. Оркестровка представляет точку зрения на бизнес-процесс со стороны его владельца.

Хореография Web-сервисов — это описание глобального бизнес-процесса, в который вовлечены несколько участников (например,

разные компании), обменивающихся сообщениями с целью достижения общего бизнес-результата. В данном случае не требуется центральный координатор, так как каждый участник взаимодействия обладает информацией о том, когда выполнять свои операции и с каким внешним Web-сервисом требуется взаимодействовать. Хореография позволяет описать бизнес-процесс как совместное действие нескольких участников, каждый из которых знает в нем свою роль. Хореография Web-сервисов может быть реализована набором оркестровок. Различия между оркестровкой и хореографией иллюстрируют рис. 2.7, 2.8 [27].

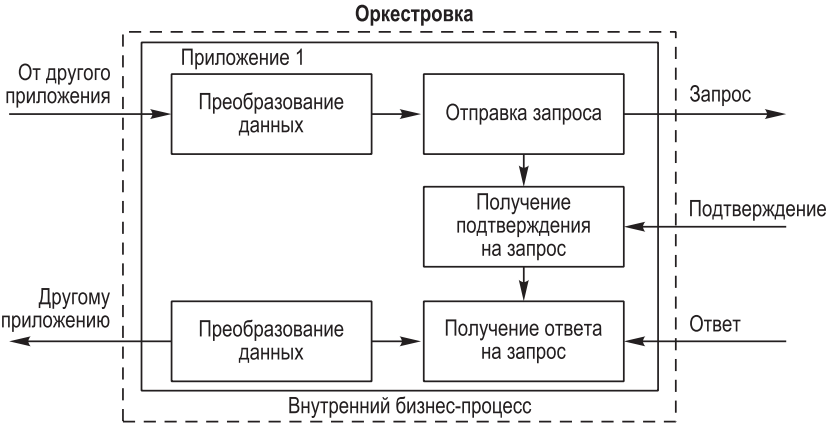

Рис. 2.7. Оркестровка сервисов

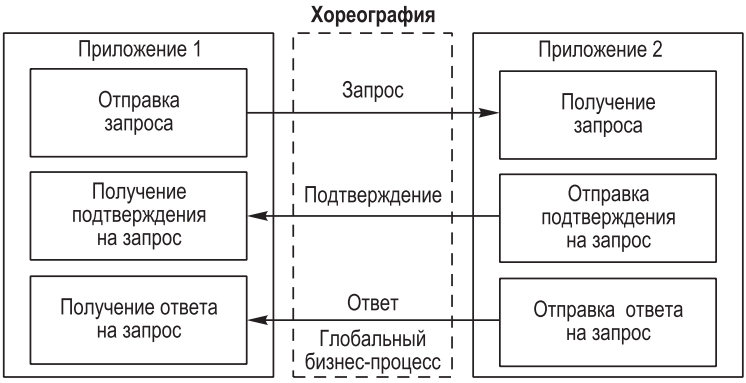

Рис. 2.8. Хореография сервисов

Для проектирования SOA-сценария взаимодействия информационной системы необходимо получить формальное описание бизнес-процессов и описание их связи с Web-сервисами. Описания бизнес-процессов в контексте технологии Web-сервисов выполняется с использованием языков WS-BPEL (Web Services Business Process Execution Language) и WS-CDL (Web Services Choreography Discription Language).

▸ Язык WS-BPEL

Язык WS-BPEL предназначен для описания бизнес-процессов, представляющих собой связанную последовательность Web-сервисов. Язык описывает оркестровку Web-сервисов при выполнении бизнес-процесса и представляет собой подмножество языка разметки XML, а также позволяет декларативно описывать связи со взаимодействующими системами и порядок обращения к сервисам, предоставляемым ими.

XML-описание бизнес-процесса, как правило, создается с помощью BPEL-визуального редактора, а затем исполняется BPEL-сервером. Язык WS-BPEL поддерживается различными серверами и средами разработки (WebLogic, Eclipse, Oracle BPEL Process Manager, Oracle Application Server, Web Sphere, Microsoft Windows Workflow Foundation и др.).

Различают абстрактное и исполняемое BPEL-описания. Абстрактное BPEL-описание используется на первоначальном этапе разработки BPEL-процесса и описывает бизнес-процесс частично, без излишней детализации. Исполняемое BPEL-описание полностью декларирует процесс и предназначено для выполнения BPEL-сервером.

Поскольку BPEL предназначен для определения последовательности вызова сервисов, он использует объявления действий. Причем BPEL-действия могут быть двух типов: структурные (содержащие другие действия и определяющие логику их выполнения) и базовые (элементарные действия).

Примеры структурных и базовых действий представлены в табл. 2.15.

С полной спецификацией языка можно ознакомиться на сайте разработчика стандарта Organization for the Advancement of Structured Information Standards (OASIS) [10].

Таблица 2.15

Примеры структурных и базовых действий

BPEL-действия	Описание
Структурные BPEL-действия	
Sequence	Содержит действия, выполняемые последовательно
If	Задает условия, для которых выполняются вложенные действия
while	Условие повторения вложенного действия
Pick	Выполняет действие при получении сообщения, связанного с действием
Flow	Обеспечивает параллельное и синхронизированное выполнение действий
For Each	Циклически повторяет действия
Базовые BPEL-действия	
Invoke	Вызов Web-сервиса
Receive	Получение сообщения от бизнес-партнера
Reply	Посылает ответ после получения сообщения от партнера
wait	Определяет задержку BPEL-процесса
exit	Завершает экземпляр BPEL-процесса

▸ Язык WS-CDL

Web Services Choreography Description Language (WS-CDL) — это XML-язык для описания взаимодействия отдельных сервисов между собой (хореографии сервисов). Язык описывает наборы правил для определения порядка обмена сообщениями между бизнес-партнерами.

WS-CDL не является исполняемым языком описания бизнес-процессов, он позволяет только строить модель взаимодействия бизнес-партнеров, которую затем можно реализовать с помощью исполняемого языка (BPEL) или любого другого языка программирования.

Полную спецификацию языка можно найти на сайте консорциума www.w3.org [11].

Глава 3

Проектирование интеграционных решений

3.1. Подход, основанный на использовании шаблонов

Связывание разнородных корпоративных приложений требует особой инфраструктуры для перемещения данных между системами. В некоторых случаях необходимо не только перемещать данные, но и поддерживать определенную бизнес-логику взаимодействия приложений, то есть автоматизировать распределенный бизнес-процесс или обеспечить единую точку доступа к информации, предоставляемой отдельными приложениями.

Не существует универсального интеграционного решения, которое могло бы подойти всем. Каждое интеграционное решение уникально и требует применения различных подходов к объединению приложений. Однако, несмотря на многообразие задач интеграции и способов связывания приложений, различными авторами предпринимались попытки выделить ряд базовых классов интеграционных решений, используя язык шаблонов.

Шаблон выполняет роль референтной модели и представляет собой абстрактное описание типовой задачи и способов ее решения.

Использование шаблонов для документирования приемов программирования началось сравнительно недавно. Развитию подхода, основанного на концепции шаблонов, способствовали работы [4, 26], посвященные проектированию архитектуры корпоративных инфор-

мационных систем. Шаблон содержит описание типового решения, которое должно быть принято на этапе проектирования, и его обоснование. Общепризнано, что использование шаблонов — один из наиболее эффективных способов документирования экспертных знаний.

Для того чтобы тот или иной шаблон можно было использовать для решения практических задач, он должен содержать: обоснование постановки задачи, описание возможных способов ее решения и объяснение преимуществ предлагаемого подхода. Язык шаблонов — это набор связанных между собой шаблонов.

Шаблоны интеграции помогают систематизировать знания в области интеграции информационных систем.

На сегодняшний день шаблоны интеграции не стандартизованы. Каждый эксперт или экспертное сообщество использует свой язык шаблонов и стиль изложения. Наиболее известными работами в данной области являются:

- Integration Patterns от компании Microsoft [7];
- Patterns: Implementing an SOA Using an Enterprise Service Bus. IBM [3].

Единственный источник на русском языке — книга Г. Хопа и Б. Вульфа «Шаблоны интеграции корпоративных приложений» (2007); в ней детально рассматриваются все аспекты только одного стиля интеграции — обмена сообщениями [6, 33].

Но, как правило, описание шаблона строится из следующих элементов:

- имя — имя шаблона, отражающее его назначение;
- пиктограмма — визуальное представление шаблона;
- контекст — описание ситуации, в которой возможно применение данного шаблона;
- возможные проблемы — описание типовых проблем, возникающих при реализации данного стиля интеграции;
- пример реализации шаблона, описание варианта технического решения.

В основу данного пособия положены шаблоны интеграции от компании Microsoft, которые представляются наиболее универсальными и охватывают такие аспекты интеграции приложений, как:

- архитектура промежуточного слоя (middleware);
- способы связывания приложений;
- топология интеграционных решений.

Общая классификация шаблонов интеграции и их взаимосвязь с основными характеристиками интеграционных решений представлены в табл. 3.1.

Таблица 3.1

Общая классификация шаблонов интеграции

Шаблоны интеграции		
Цель интеграции	⇨	***Шаблоны архитектуры промежуточного слоя***
Единое представление корпоративных данных		Агрегация сущностей
Распределенный бизнес-процесс		Интеграция процессов
Единая точка доступа к данным из нескольких источников		Интеграция, ориентированная на портал
Уровень связывания приложений	⇨	***Шаблоны связывания приложений***
База данных		Интеграция данных
Уровень бизнес-логики		Функциональная интеграция
Уровень пользовательского интерфейса		Презентационная интеграция
Способ взаимодействия приложений	⇨	***Шаблоны топологии интеграционного решения***
Взаимодействие «один-к-одному»		Точка-точка
Взаимодействие через центральный «коммутатор»		Брокер сообщений
Открытое взаимодействие		Шина сообщений
Взаимодействие «один-ко-многим»		Публикация-подписка

В дальнейшем мы более подробно рассмотрим шаблоны каждой группы.

Следует понимать, что представленная система шаблонов образует иерархию, на верхнем уровне которой находятся шаблоны, определяющие архитектуру промежуточного слоя (рис. 3.1). Для любой архитектуры слоя middleware могут быть использованы различные способы связывания приложений, равно как и для каждого способа связывания могут быть предложены различные топологии интеграционного решения.

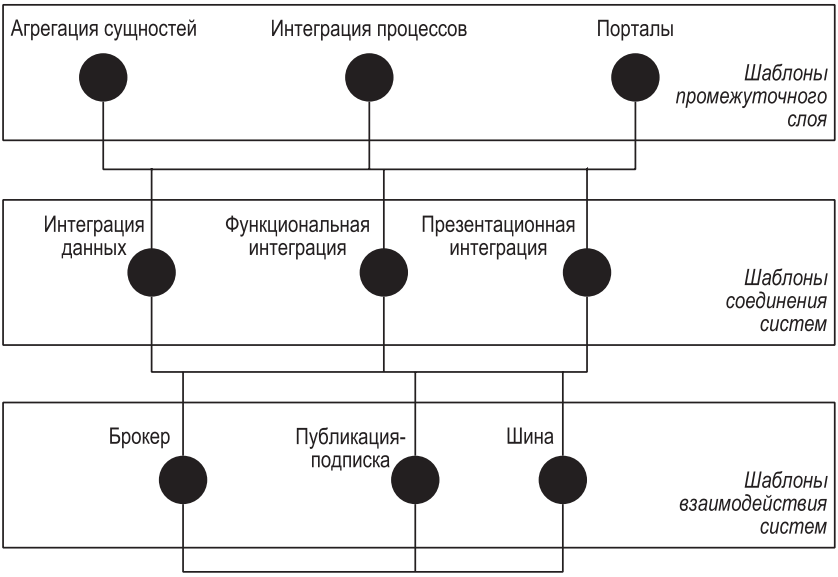

Рис. 3.1. Иерархия шаблонов

В заключение отметим, что реальные интеграционные решения, как правило, основываются на использовании нескольких шаблонов (представляют собой гибридные решения). Примеры гибридных интеграционных решений будут рассмотрены во второй части учебного пособия.

3.2. Архитектура промежуточного слоя

С точки зрения архитектуры интеграционного решения можно выделить три базовых шаблона промежуточного слоя[1] [7]:

- агрегация сущностей (Entity Aggregation);
- интеграция процессов (Process Integration);
- интеграция, ориентированная на портал (Portal Integration).

[1] Следует обратить внимание, что в данном контексте речь идет именно о шаблонах архитектуры интеграционного решения, а не о моделях взаимодействия распределенных компонентов. Использование шаблонов этого уровня позволяет абстрагироваться от деталей реализации решения.

Основные характеристики базовых шаблонов представлены в табл. 3.2.

Таблица 3.2

Основные характеристики базовых шаблонов

Шаблон промежуточного слоя	Решаемая проблема
Агрегация сущностей	Как приложениям эффективно использовать данные, распределенные между несколькими репозиториями
Интеграция процессов	Как управлять исполнением распределенного бизнес-процесса, если отдельные бизнес-функции выполняются разными приложениями
Портал-ориентированная интеграция	Как конечные пользователи могут эффективно решать задачи, требующие доступа к информации, распределенной между несколькими приложениями

3.2.1. Агрегация сущностей

Агрегация сущностей — наиболее доступный, а поэтому и наиболее распространенный вид интеграции.

Цель агрегации сущностей — обеспечить приложениям возможность эффективно использовать данные, распределенные между несколькими репозиториями. Для эффективной работы с данными приложениям необходимо иметь единое согласованное в масштабах организации (отрасли) представление ключевых сущностей (например, таких как «Клиент», «Заказ», «Счет», «Продукт» и т.п.).

Задача осложняется тем, что:

- корпоративные системы, как правило, используют разные информационные модели для описания одной и той же сущности (например, сущность «сотрудник» по-разному представлена в бухгалтерской и кадровой системах);
- зачастую существует семантический диссонанс между данными из разных систем. Один и тот же элемент данных из разных систем может представлять разную информацию (например, атрибут NumberOfDaysRemaining для задачи проекта может содержать только рабочие дни в одном репозитории или включать выходные дни в другом);
- даже при отсутствии семантических различий возможно несовпадение данных для одного и того же экземпляра сущности (напри-

мер, адрес одного и того же клиента в системе оформления заказа и адрес в CRM-системе различаются);

- репозитории могут содержать ошибочные данные;
- при проведении синхронизации данных между несколькими репозиториями может быть нарушена ссылочная целостность;
- приложение-потребитель может затребовать данные, хранящиеся в разных репозиториях.

Решением является создание промежуточного слоя, обеспечивающего единое логическое представление сущностей в масштабе организации и физически связанного как с репозиториями данных, так и с приложениями — потребителями информации (рис. 3.2).

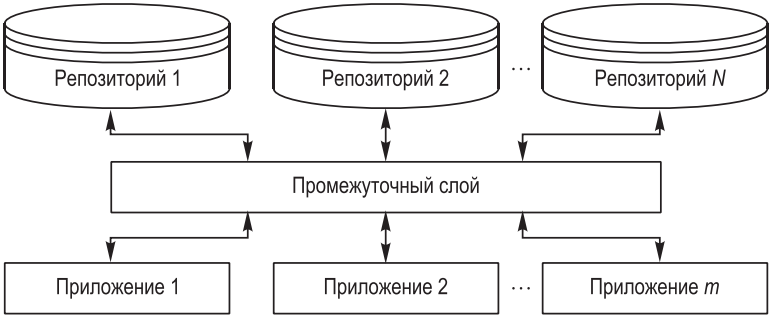

Рис. 3.2. Агрегация сущностей

Агрегация сущностей вне зависимости от конкретной реализации интеграционного решения включает два основных этапа:

1) определение единой в рамках предприятия модели данных, которая обеспечит унифицированное представление сущностей;

2) установление физических связей между компонентами интеграционного решения.

В разных репозиториях используются разные схемы данных для одной и той же сущности. Задача промежуточного слоя — гармонизировать различия между этими схемами. Промежуточный слой должен:

- удовлетворять потребностям всех интегрируемых приложений;
- определять каноническую схему данных для всех сущностей, используемых несколькими приложениями;
- поддерживать трансформации схем каждого источника данных в каноническую схему данных.

В качестве примера рассмотрим случай, когда необходимо интегрировать информацию о клиенте, распределенную по нескольким источникам данных (база данных контактов содержит информацию о способах связи с клиентом, а база финансовой системы — данные по кредитным картам клиента). В промежуточном слое определена каноническая схема данных, которая содержит все атрибуты, необходимые для описания сущности «Клиент». Кроме того, промежуточный слой содержит также маппинги (mapping) — трансформации, обеспечивающие преобразование индивидуальных схем в каноническую схему данных (и обратное преобразование).

Разработка унифицированной схемы данных — одна из наибольших трудностей, присущих данному стилю интеграции. Создание унифицированной (иногда говорят, «канонической») схемы данных (рис. 3.3), удовлетворяющей потребностям нескольких приложений, сопряжено со сложностями как технического, так и политического характера. Если применение такой схемы данных приводит к снижению производительности бизнес-критичного приложения, то руководство компании может настоять на пересмотре интеграционного решения.

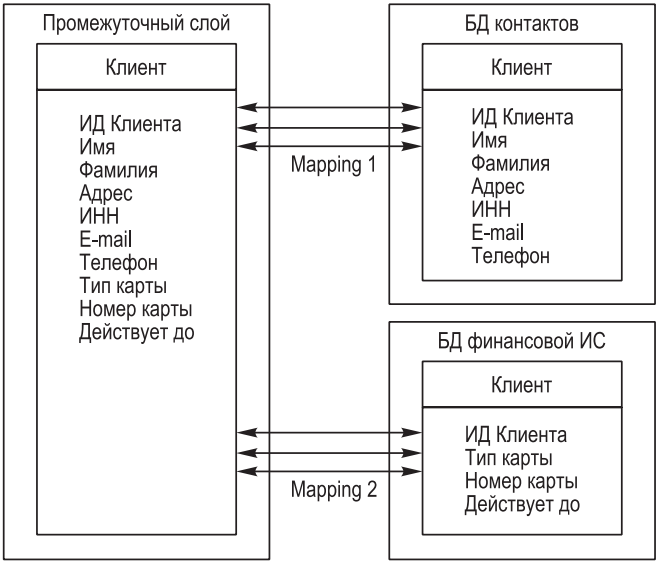

Рис. 3.3. Каноническая схема данных

Существуют два базовых архитектурных подхода к интеграции информации:

1) репликация данных;

2) федерация информации.

Рассмотрим их более подробно.

▸ Репликация данных

Под *репликацией данных* понимают процесс перемещения данных между двумя или более репозиториями (рис. 3.4). Репликация — это ориентированная на обработку наборов данных технология преобразования, предназначенная для решения задач миграции, консолидации, создания хранилищ данных. Поскольку процесс перемещения данных подразумевает их извлечение из одного источника, загрузку в целевой репозиторий, а также возможное попутное преобразование, часто используется термин ETL (Extraction Transformation Load).

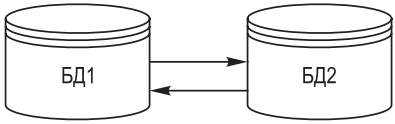

Рис. 3.4. Репликация данных

Технология репликации ориентирована на обработку очень больших объемов данных.

Основными достоинствами репликации являются:

- возможность использования разнородных данных любого качества (поскольку на этапе трансформации могут быть выполнены все необходимые процедуры проверки, очистки и обогащения данных);
- хорошая масштабируемость решений;
- наличие четкой методологии интеграции;
- асинхронный обмен данными;
- отсутствие необходимости изменений в интегрируемых информационных системах;
- эффективный доступ к интегрируемым данным за счет отделения приемника от источника;
- слабая связь между интегрируемыми системами;
- возможность повторного использования объектов и преобразований (снижает затраты на разработку и поддержку);
- относительно низкая стоимость;
- наличие доступных инструментов (программного обеспечения класса middleware).

▸ Федерация информации

Под *федерацией информации* понимают процесс извлечения данных из различных источников в режиме реального времени и представление их в едином унифицированном виде. Это технология прозрачного доступа и преобразования данных, обеспечивающая единый интерфейс доступа ко всей корпоративной информации (рис. 3.5).

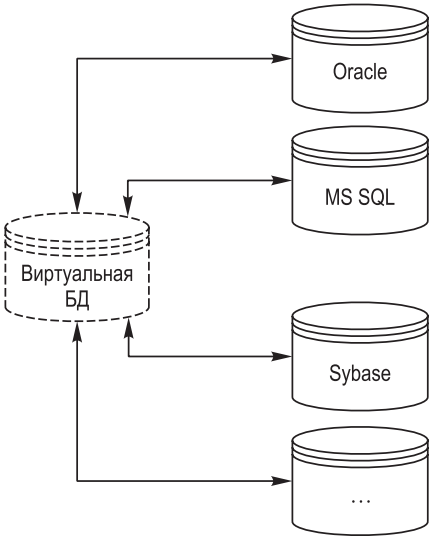

Рис. 3.5. Федерация информации

Достоинствами федерации информации являются:

- возможность интеграции структурированных и неструктурированных данных из многих источников;
- доступ к данным в режиме реального времени (по требованию);
- поддержка процессов чтения и записи данных;
- возможность преобразования данных для бизнес-анализа и обмена информацией;
- наличие доступных инструментов (программного обеспечения класса middleware).

В табл. 3.3 проводится сравнение двух подходов к интеграции данных.

Таблица 3.3

Сравнительная характеристика подходов к интеграции данных

Критерий сравнения	Репликация	Федерация
Поток данных	В одну сторону — от источника к приемнику	В обе стороны
Перемещение данных	По расписанию — пакетная загрузка; управляется процессом	В момент запроса; управляется запросом SQL
Задержка	День-неделя-месяц	Реальное время
Преобразование, очистка, использование метаданных в процессе	Лучшее; обычно высокая степень повторного использования объектов и процессов	Среднее; преобразования встроены в представления и объекты базы данных
Транспорт	FTP, прямое соединение с базой данных	Прямое соединение с базой данных
Объемы обрабатываемых данных	Очень большие (миллионы записей)	Средние (сотни тысяч удаленных записей)[1]
Сложность преобразований	Любая	Преобразования, которые можно описать на SQL
Поддержка мониторинга событий	Ограниченная; можно улучшить с помощью технологии захвата изменений в источнике	Зависит от возможностей триггеров в источнике
Контроль потоков работ	По расписанию; обработка ошибок	Нет

В заключение перечислим основные преимущества и недостатки данного стиля интеграции (табл. 3.4).

Таблица 3.4

Преимущества и недостатки агрегации сущностей

Преимущества	Недостатки
Обеспечивает унифицированное представление основных корпоративных (отраслевых) данных Облегчает доступ к информации Уменьшает семантический диссонанс между существующими приложениями	Создает дополнительный архитектурный слой Требует консенсуса между подразделениями (организациями) по вопросу унифицированного представления данных Требует подключения существующих приложений к промежуточному слою

[1] Различия, связанные с объемами данных, нивелируются по мере развития технологии Big data.

3.2.2. Интеграция процессов

Признаком того, что приложения необходимо интегрировать, является участие нескольких информационных систем в выполнении единственной бизнес-функции. Обычно вся необходимая функциональность поддерживается различными системами.

В качестве примера рассмотрим процесс обработки заказа клиента гипотетической торговой компанией.

Пусть обработка заказа включает следующие действия:

- проверка финансового досье клиента (если у клиента остались неоплаченные счета, то новый заказ отклоняется);
- проверка наличия товара на складе (если товар отсутствует, то заказ отклоняется);
- определение адреса доставки и выставление счета (выполняется, если у клиента нет неоплаченных счетов и товар есть на складе).

Диаграмма операций, выполняемых при обработке заказа, представлена на рис. 3.6.

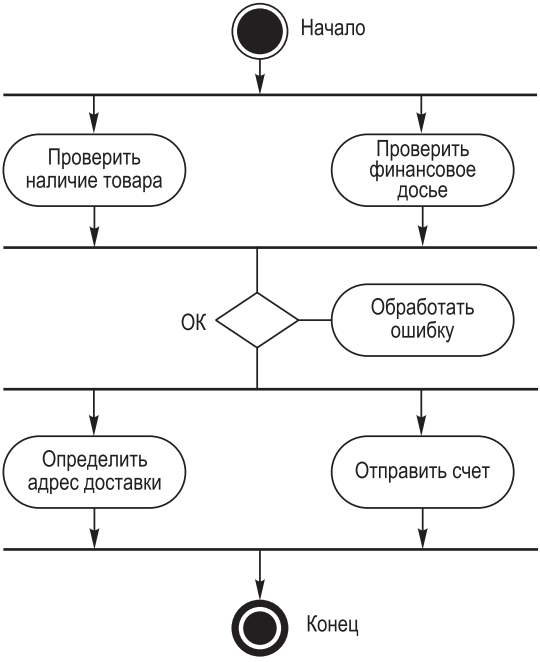

Рис. 3.6. Обработка заказа

Действия, показанные на диаграмме, выполняются независимыми системами: проверка финансового досье и выставление счета — бухгалтерской системой, проверка наличия товара на складе — системой управления запасами, определение адреса доставки — CRM-системой.

Основная задача интеграции процессов — координация бизнес-функций, поддерживаемых различными информационными системами.

Говоря об интеграции процессов, следует иметь в виду следующее:

- для реализации распределенного бизнес-процесса одна информационная система должна непосредственно обращаться к другой, что приводит к установлению зависимости между приложениями и может потребовать внесения изменений в информационные системы;
- стандартная бизнес-функциональность, выполняемая приложением, может отличаться от бизнес-функциональности, которую необходимо обеспечить в распределенном процессе, кроме того, требования к функциональности могут часто изменяться;
- время выполнения сложных бизнес-функций может составлять часы (дни, недели), в то время как большинство функций, доступных в существующих приложениях, являются синхронными, то есть запросившее бизнес-функцию приложение будет обязано ожидать ответа в течение длительного времени;
- если одно из приложений прерывает работу в процессе выполнения своей бизнес-функции, то необходимо обеспечить приведение приложений в согласованное состояние (например, откатить все взаимозависимые транзакции);
- поскольку сложные бизнес-процессы могут выполняться в течение длительного времени, новые запросы будут поступать до завершения обработки предыдущих запросов. Для оптимизации времени выполнения запросов необходимо обеспечить их параллельную обработку в отдельных потоках исполнения;
- построение модели бизнес-процесса может быть затруднено. Очень часто, принимая решения, пользователи руководствуются собственным опытом или интуицией, а не документированными бизнес-правилами.

Решение задачи интеграции бизнес-процессов требует построения модели распределенного бизнес-процесса, а также создания отдельного компонента управления им (process manager), который контролирует параллельно выполняемые экземпляры процесса и обеспечивает взаимодействие с существующими приложениями для выполнения отдельных шагов в соответствии с заданной моделью (рис. 3.7).

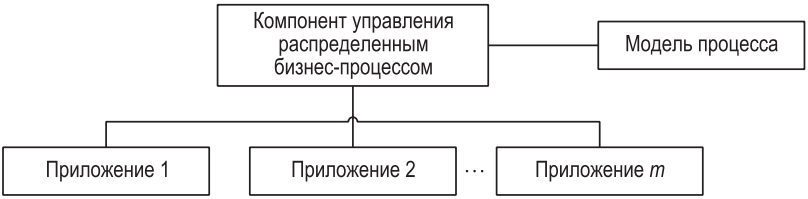

Рис. 3.7. Интеграция бизнес-процессов

По каждому запросу на выполнение бизнес-процесса Process manager создает новый экземпляр процесса на основании заданной модели.

Process manager отслеживает статус выполнения процесса и определяет очередность работы приложений, поэтому каждое приложение, участвующее в обеспечении распределенного бизнес-процесса, остается независимым и не обязано знать последовательность шагов, определенную в модели.

Process manager сохраняет статус выполнения процесса в том случае, если какое-либо из приложений становится временно недоступно, и завершает выполнение процесса, когда это становится возможным.

Концепция интеграции бизнес-процессов отделяет описание бизнес-процесса (модель процесса), исполнение бизнес-процесса (Process manager) и реализацию бизнес-функций (интегрируемые приложения). Такой подход позволяет повторно использовать отдельные функции приложений в различных бизнес-процессах.

В обобщенном виде назначение компонентов управления распределенными бизнес-процессами рассмотрено в табл. 3.5.

Таблица 3.5

Назначение компонентов управления распределенными бизнес-процессами

Компонент	Назначение
Модель процесса	Определяет последовательность шагов для выполнения бизнес-функции
Компонент управления распределенным бизнес-процессом (Process manager)	Интерпретирует модель процесса. Связывает шаги процесса, определенные в модели, с функциями приложений. Управляет экземплярами процесса, ведет мониторинг текущего состояния запущенных экземпляров процесса
Приложение	Выполняет определенную бизнес-функцию

Process manager обычно предоставляет интерфейс, с помощью которого бизнес-процесс, описанный в виде модели, может быть запущен пользователем или другим приложением. Такой интерфейс реализован либо как интерфейс пользователя, либо как API, доступное любым внешним приложениям. Причем запуск бизнес-процесса через API для внешнего приложения неотличим от вызова API отдельного приложения. Поэтому любой бизнес-процесс может быть использован как часть «родительского» бизнес-процесса, который определен моделью более высокого уровня. Таким образом, компоненты управления распределенными бизнес-процессами и отдельные приложения могут быть объединены в сколь угодно сложную иерархию (рис. 3.8).

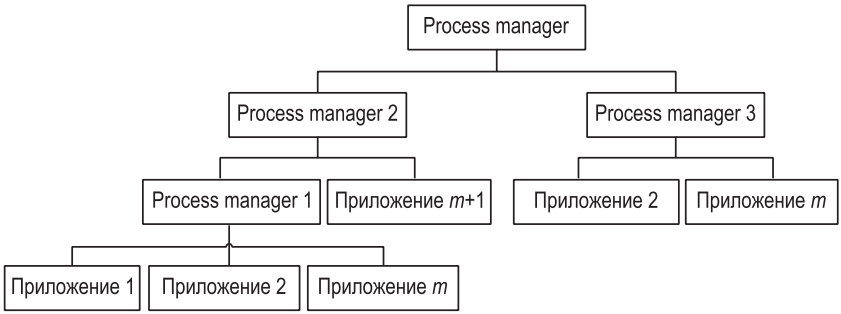

Рис. 3.8. Иерархическая интеграция бизнес-процессов

Process manager отличается от традиционного приложения, реализующего определенную бизнес-логику, тем, что использует функциональность внешних приложений. Поэтому специфическими требованиями к компоненту управления распределенным бизнес-процессом являются:

- умение соотносить сообщения, получаемые от внешних систем / передаваемые внешним системам с экземпляром бизнес-процесса, для которого они предназначены;
- поддержка распределенных во времени транзакций;
- обработка ошибок, возникающих на различных этапах выполнения бизнес-процесса, и выполнение соответствующих ответных действий, предусмотренных логикой процесса;
- обеспечение компенсационной логики для отката завершенных ранее действий в случае возникновения ошибки на одном из этапов выполнения бизнес-процесса.

Интеграцию бизнес-процессов целесообразно использовать для ускорения выполнения последовательности задач. В финансовой индустрии широкую популярность получила концепция *сквозной непрерывной обработки* (STP, Straight Through Processing), означающая процесс непрерывной, полностью автоматизированной обработки информации. Концепция возникла в конце 1990-х гг. для описания оборота ценных бумаг и сегодня является стратегическим направлением развития автоматизированной информационной системы участника финансового рынка.

В заключение перечислим основные преимущества и недостатки данного стиля интеграции (табл. 3.6).

Таблица 3.6

Преимущества и недостатки интеграции процессов

Преимущества	Недостатки
Возможность моделировать бизнес-процесс независимо от реализации бизнес-функций	Возможное падение производительности из-за слишком большого числа параллельно выполняемых процессов
Обеспечение соблюдения правил выполнения бизнес-процесса	Сложность и, как следствие, высокая стоимость решения. Поскольку реализация компонента управления распределенным бизнес-процессом — достаточно сложная задача, оптимальным вариантом является использование промышленных интеграционных платформ
Повторное использование функций, предоставляемых приложениями (существующие приложения не зависят от слоя управления бизнес-процессом)	
Быстрая адаптация к любым бизнес-требованиям	
Получение данных для анализа выполнения и оптимизации бизнес-процессов (так как управление исполнением процесса осуществляется централизованно, легко собрать статистику о времени исполнения процесса и его отдельных этапов, типичных ошибках и т.п.).	

3.2.3. Интеграция, ориентированная на порталы

Цель построения корпоративного портала — предоставление пользователям единой точки доступа ко всей корпоративной информации.

Для выполнения многих бизнес-задач пользователям требуется получать информацию из разных источников. Например, для проверки состояния заказа сотруднику может понадобиться доступ к системе

управления заказами, а также к системе обработки заказов, принятых через интернет-магазин. Основная задача корпоративных порталов — обеспечить представление информации из нескольких источников.

Конечные пользователи не могут эффективно выполнять задачи, которые требуют доступа к информации, распределенной по нескольким системам, если они привыкли вручную копировать информацию из одной системы в другую, пользуясь зачастую разными компьютерами для доступа к необходимым системам. Такой «ручной» тип интеграции очень неэффективен и рискован (большое количество ошибок ввода).

Корпоративный портал как интеграционное решение целесообразно использовать в том случае, если:

- отсутствует хорошее понимание бизнес-процесса, позволяющее построить его модель. Модель бизнес-процесса не удается построить и в тех случаях, когда пользователи принимают решения, основываясь на информации, которую они получают (интеграция бизнес-процессов не может быть реализована);
- существует непреодолимый семантический диссонанс между отдельными приложениями (интеграция данных невозможна или экономически нецелесообразна).

Построение портала — сравнительно простая форма интеграции, ее достаточно легко реализовать, она оказывает минимальное влияние на связываемые приложения. Этот подход менее эффективен, чем интеграция бизнес-процессов, но он оставляет больше простора для инициативы пользователей и его можно реализовать быстрее.

В ряде случаев построение портала может рассматриваться как временное (переходное) решение, позволяющее лучше понять бизнес-процессы и подготовиться к интеграции процессов.

Движок портала (рис. 3.9) взаимодействует с приложениями, используя разные способы соединения (подробнее о способах соединения приложений см. п. 3.3).

Рис. 3.9. Портал-ориентированная интеграция

Портал-ориентированные интеграционные решения могут быть очень разными по уровню сложности. Можно выделить следующие типы порталов (перечислены в порядке от простых к сложным):

- «только для чтения» — позволяют пользователю только просматривать данные различных приложений. Данные по каждому приложению располагаются обычно в отдельных областях экрана;
- «простое управление» — пользователю портала представляют не данные, а наглядные критерии, рассчитанные по определяемым бизнес-логикой правилам, что позволяет ему быстро принимать решения. В последнее время у различных вендоров популярны так называемые «информационные панели», представляющие индикаторы процесса в наглядной и удобной для пользователя форме;
- «интерактивные» — поддерживают ограниченное взаимодействие между областями экрана, относящимися к разным приложениям. Например, выбор элемента в списке одной области приводит к представлению определенной информации в другой. При выборе клиента в области 1 (по данным CRM-системы) в области 2 отображается его финансовое досье (по данным финансовой системы).

В заключение перечислим основные преимущества и недостатки данного стиля интеграции (табл. 3.7).

Таблица 3.7

Преимущества и недостатки интеграции, ориентированной на порталы

Преимущества	Недостатки
Оказывает минимальное влияние на интегрируемые системы	Неэффективен (по сравнению с интеграций процессов), так как предполагает участие пользователя на этапе принятия решения
Высокая скорость реализации. Интегрировать приложения возможно за несколько дней (с учетом того, что многие вендоры предлагают разнообразные платформы для построения корпоративных порталов)	Существует риск совершения ошибки пользователем
Можно использовать в тех случаях, когда бизнес-процессы не формализованы	

3.3. Способы связывания приложений

Архитектура практически любого приложения может быть представлена тремя логическими уровнями:

1) уровнем представления — это уровень пользовательского интерфейса, предназначенный для просмотра, ввода и корректировки данных, отправки на выполнение запросов конечными пользователями приложения;

2) уровнем бизнес-логики — уровень, на котором сосредоточена бизнес-логика приложения, осуществляется управление потоками данных и организуется взаимодействие частей приложения;

3) уровнем данных — уровень, отвечающий за организацию доступа к данным и работу с базой данных, это уровень серверов баз данных.

Поскольку «стыковка» с промежуточной средой может осуществляться на каждом из логических уровней, выделяют три базовых способа интеграции приложения с промежуточным слоем:

1) интеграция на уровне данных — в промежуточный слой поступают данные из базы данных;

2) функциональная интеграция — промежуточный слой интегрируется с уровнем бизнес-логики посредством API-интерфейсов и сервисов, предоставляемых приложением;

3) интеграция на уровне представлений — в промежуточный слой извлекаются данные по технологии screen scraping («прочесывание экрана»). Обеспечивает доступ к функциям приложения через пользовательский интерфейс путем моделирования ввода данных пользователем и чтения данных с экрана монитора.

3.3.1. Интеграция данных

Большинство корпоративных приложений хранит информацию в виде плоских файлов, реляционных, иерархических или объектно-ориентированных баз данных. Концепция интеграции данных предполагает, что одно приложение может использовать данные другого приложения, заимствовав их непосредственно из репозитория источника (базы данных или файла).

Интеграцию приложений на уровне данных иллюстрирует рис. 3.10.

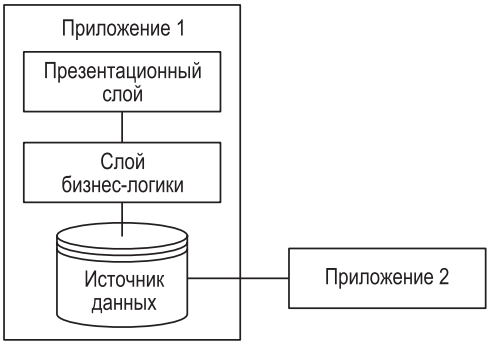

Рис. 3.10. Интеграция приложений на уровне данных (базовый шаблон)

Поскольку вся необходимая информация находится в репозитории приложения, ее можно извлекать оттуда, не вторгаясь в работу приложения. Вместе с тем получение доступа к данным приложения связано с определенными рисками (особенно опасна запись обновлений прямо в базу данных, минуя приложение). Кроме того, производители коммерческого программного обеспечения часто меняют схему базы данных в новых версиях приложений, это делает интеграционные решения, основанные на интеграции данных, чувствительными к изменениям.

Перечислим основные проблемы, возникающие при интеграции данных:

- неоднородность источников данных на физическом уровне (могут использоваться различные форматы файлов, одни источники могут быть веб-сайтами, а другие — объектными базами данных и т.д.);
- неоднородность источников данных на логическом уровне — неоднородность используемых моделей данных для различных источников;
- проблема семантического диссонанса;
- постепенно усложняющийся процесс управления данными. Необходимо обрабатывать постоянно растущие объемы критически важных для бизнеса данных. Причем эти данные становятся все более разнообразными (имеют различный формат, поступают от различных партнеров и систем). Задача управления временем

передачи данных, форматами, структурами и объемами становится все более нетривиальной;

- увеличивается количество бизнес-запросов, требующих своевременной доставки информации потребителю. Причем доставка данных может осуществляться как в пакетном режиме, так и в режиме реального времени;
- возрастают проблемы с качеством данных. Бизнесу нужны достоверные данные, а потому необходимы инструменты аудита, мониторинга и очистки данных.

Можно привести огромное количество примеров интеграции приложений на уровне данных. Пусть система оформления заказов торговой компании использует справочник продуктов, который должен быть синхронизирован со справочником продуктов ERP-системы. Если коды продуктов не меняются часто, то данные системы-источника (ERP-системы) периодически (ежедневно или еженедельно) должны синхронизироваться с данными приемника (системы оформления заказов).

С технической точки зрения интеграция на уровне данных — это самый простой тип интеграции, который может быть реализован с использованием FTP, командных файлов, различных утилит СУБД, ETL-инструментов и специальных сервисов интеграции данных.

Выбор способов связывания приложений на уровне данных определяется частотой изменения данных, сложностью их преобразований, имеющейся инфраструктурой.

Выделяют три базовых шаблона связывания приложений на уровне данных:

1) файловый обмен;
2) общая база данных;
3) копирование данных.

▸ Файловый обмен

Файловый обмен — это наиболее простой и исторически первый способ обмена данными между приложениями.

Технология основана на том, что одно приложение сохраняет (экспортирует) данные в файл, а другое — читает (импортирует)

данные из файла. Обмен данными между приложениями осуществляется в пакетном режиме. Операции экспорта и импорта данных выполняются специальными модулями экспорта или импорта соответствующих приложений. В ряде случаев задача разработки подобных моделей решается в рамках интеграционного проекта, однако большинство современных приложений имеет встроенные механизмы экспорта/импорта данных в файлы популярных форматов.

Схема файлового обмена представлена на рис. 3.11.

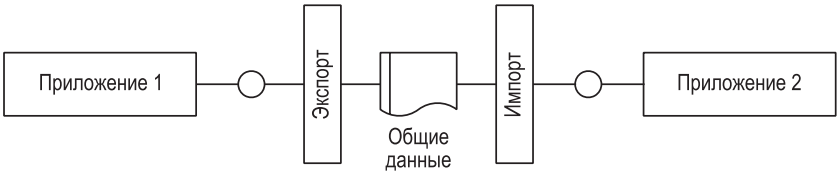

Рис. 3.11. Файловый обмен

При разработке интеграционного решения, использующего механизм файлового обмена, нужно учитывать следующее:

- необходимо выработать соглашение об общем формате файлов (xml, txt, xls, ...). Задача преобразования данных в необходимый формат возлагается на модули импорта/экспорта;
- приложения должны использовать общее соглашение о правилах именования и расположения файлов;
- приложение, создающее файл, должно обеспечить уникальность его имени;
- необходимо определить регламент записи/чтения файлов. Частота создания/чтения файлов определяется бизнес-логикой процесса и технологическими ограничениями (доступность приложения, объем трафика, скорость передачи данных);
- приложение, записывающее данные в файл, и приложение, читающее данные из файла, не могут получить к нему доступ одновременно, поэтому должен использоваться механизм блокировки доступа к файлу на время его записи;
- должен быть использован механизм удаления/архивирования старых файлов.

Преимущества и недостатки файлового обмена рассмотрены в табл. 3.8.

Таблица 3.8

Преимущества и недостатки файлового обмена

Преимущества	Недостатки
Универсальный способ для всех операционных систем, поддерживается любыми языками программирования	Возможность рассинхронизации интегрируемых систем вследствие низкой частоты обмена информацией
Не нужны сведения о внутренней реализации приложения; основная задача — преобразование форматов файлов	Нерациональное использование ресурсов при слишком частой работе с файлами
Обеспечивает слабое связывание приложений	
Нет потребности в специализированном связующем дорогостоящем программном обеспечении	

▸ Общая база данных

Общая база данных обеспечивает доступ нескольких приложений к данным одного физического репозитория (рис. 3.12).

Рис. 3.12. Общая база данных

Общая база данных обеспечивает согласованность хранящейся в ней информации за счет того, что все приложения используют общую модель данных.

Оптимальный способ реализации общей базы данных заключается в использовании реляционной СУБД с поддержкой SQL. Язык запросов SQL поддерживается практически всеми платформами и избавляет от необходимости использования новой технологии.

Наличие общей базы данных решает проблему семантического диссонанса. Все вопросы, связанные с разработкой унифицированной модели данных, решаются на этапах проектирования и реализации интеграционного решения.

Решения, использующие общую базу данных, сложно поддерживать. Изменения в одном приложении могут повлечь за собой необходимость изменения структуры базы данных, что, в свою очередь, скажется на всех остальных приложениях. Поэтому организации, использующие общую базу данных, крайне неохотно относятся к необходимости ее изменения, что может затруднить реализацию новых бизнес-требований.

Принято считать, что общая база данных — это наименее эффективный способ интеграции. Однако в ряде случаев он может с успехом использоваться (например, для ведения общекорпоративных справочников и построения корпоративных хранилищ данных).

Преимущества и недостатки общей базы данных перечислены в табл. 3.9.

Таблица 3.9

Преимущества и недостатки общей базы данных

Преимущества	Недостатки
Решается проблема семантического диссонанса для нескольких приложений	Сложно разработать единую схему базы данных. При создании унифицированной схемы базы данных возможны сложности технического (может привести к снижению производительности бизнес-критичного приложения) и политического характера
	Есть программы (коммерческое программное обеспечение), которые работают со встроенной базой данных (работает только со встроенной схемой данных)
	Возможно снижение производительности при увеличении числа обращений к общей базе данных, возрастает вероятность блокировки данных
	Доступ к базе данных по медленным линиям связи
	Изменение в приложениях могут повлечь за собой потребность в изменении структуры базы данных
	Изменение структуры базы данных требует согласования со всеми использующими ее приложениями
	Низкий уровень абстракции; приложения работают не с логическими объектами, а с конкретными полями и записями

▸ Копирование данных

Идея заключается в использовании копий базы данных для различных приложений, при этом синхронизация данных обеспечивается путем копирования данных из одной базы данных в другую (рис. 3.13).

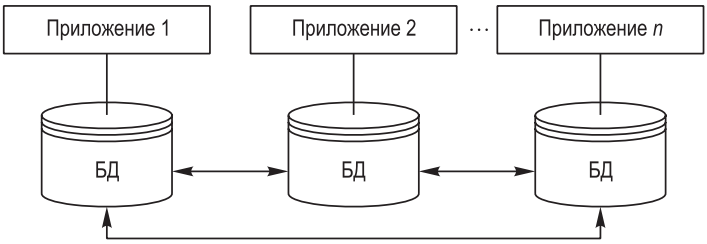

Рис. 3.13. Копирование данных

Цель интеграции — организовать такое управление копиями баз данных, которое минимизирует неизбежно возникающую рассинхронизацию данных.

Базовые сценарии копирования данных могут быть описаны с использованием таких понятий, как «издатель» (сервер СУБД, имеющий эталонную копию данных) и «подписчик» (сервер СУБД, нуждающийся в актуальной копии данных).

Возможны три основных сценария копирования данных [19]:

1) один издатель, много подписчиков — существует центральная база данных, содержащая актуальную копию данных, и несколько подписчиков (например, база данных каталога, размещенная в центральном офисе компании, и базы данных филиалов, нуждающиеся в копиях базы данных каталога);

2) много издателей, один подписчик — единственная центральная база данных, являющаяся подписчиком публикаций с нескольких серверов (например, информация о состоянии региональных складов публикуется на центральном сервере компании);

3) много издателей, много подписчиков — в этой модели каждый сервер СУБД одновременно является и издателем и подписчиком. Например, пусть существует сеть точек продаж компании. Каждая точка продаж должна владеть актуальной информацией о товарах, имеющихся в наличии в других точках. После каждой транзакции в каждой точке продаж выполняется репликация транзакции на все базы данных.

При выборе сценария копирования данных должны быть учтены следующие факторы:

- автономность — степень независимости, которой будут обладать подписчики по отношению к получаемым данным (это может быть доступная для чтения или изменения копия данных);
- допустимая задержка в обновлении данных — время, в течение которого подписчик может обходиться без свежей копии данных;
- согласованность обновления данных — подписчик может получать копии транзакций в том же порядке, в котором они выполнялись на сервере издателя, или этот порядок не важен (или достаточно выборки из журнала транзакций издателя).

Преимущества и недостатки копирования данных рассмотрены в табл. 3.10.

Таблица 3.10

Преимущества и недостатки копирования данных

Преимущества	Недостатки
Механизмы репликации поддерживаются встроенными функциями всех промышленных СУБД	Задержка обновления данных
	Доступ к базе данных по медленным линиям связи
Достаточным условием для копирования данных между СУБД различных производителей (гетерогенная репликация) является совместимость с OLE DB	Необходимость в разрешении конфликтов, возникающих при изменении одной и той же записи в разных копиях базы данных

3.3.2. Функциональная интеграция

Функциональная интеграция — это способ связывания приложений на уровне слоя бизнес-логики, при котором одно приложение использует функционал, инкапсулированный в другом приложении.

Шаблон функциональной интеграции представлен на рис. 3.14.

Большинство современных коммерческих приложений имеют хорошо документированные программные интерфейсы для доступа к базовой функциональности. Обычно производители приложений предлагают набор API, специально разработанных для целей интеграции с внешними системами. Причем это может быть как набор компонентов (объекты COM, компоненты CORBA и др.), так и прямой программный интерфейс, зависящий от языка про-

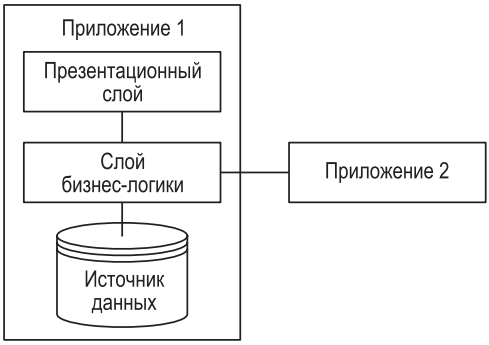

Рис. 3.14. Шаблон функциональной интеграции

граммирования (библиотеки C++, C#, Java и др.). Программные интерфейсы обычно независимы от модели данных приложения и более стабильны, чем пользовательский интерфейс. Если модель данных может претерпевать изменения с каждой новой версией программного обеспечения, то набор API для обеспечения обратной совместимости обычно предоставляется разработчиками информационной системы.

Пример функциональной интеграции — использование функции расчета кредитного рейтинга заемщика, инкапсулированной в скоринговой системе. Многие финансовые приложения нуждаются в получении кредитного рейтинга клиента (для проведения классификации клиентов или построения отчетов). Кредитные рейтинги представляют собой расчетные величины и зависят от большого количества быстро изменяющихся факторов, то есть быстро утрачивают актуальность. В данном случае применение сценария общей базы данных нецелесообразно, так как либо в ней будут сохраняться неактуальные данные, либо придется копировать в общую базу данных все первичные документы и реализовывать в каждом приложении дополнительный компонент для расчета кредитного рейтинга.

Для связывания приложений на уровне бизнес-логики необходимо выполнение двух условий:

1) бизнес-функция, которую необходимо использовать, существует в приложении-источнике. Зачастую API приложений не обеспечивает доступа к данным и функциям на том уровне детализации, который необходим для интеграции бизнес-функций;

2) возможен удаленный доступ к программному интерфейсу, инкапсулирующему нужную бизнес-функцию.

Выделяют три базовых шаблона связывания приложений на уровне бизнес-логики:

1) использование распределенных объектов;

2) интеграция, ориентированная на сообщения;

3) сервис-ориентированная интеграция.

▸ Использование распределенных объектов

Данный тип интеграции основан на стандартах объектно-ориентированного взаимодействия (см. п. 1.3) и использует такие технологии, как .NET remoting, COM+ или CORBA. Объекты одного приложения взаимодействуют с объектами другого приложения, используя механизм виртуализации. Преимущества и недостатки функциональной интеграции приведены в табл. 3.11.

Таблица 3.11

Преимущества и недостатки функциональной интеграции

Преимущества	Недостатки
Использование хорошо известных разработчикам технологий, применяемых при построении распределенных приложений Целесообразно использовать, если необходимо обеспечить интеграцию приложений в режиме реального времени	Сильное связывание приложений Сложная модель взаимодействия приложений Плохо масштабируемые решения Не все технологии обеспечивают межплатформенное взаимодействие Сложности поддержки

▸ Интеграция, ориентированная на сообщения

Данный способ связывания приложений использует промежуточное программное обеспечение (систему обмена сообщениями), ориентированное на посылку сообщений. Обмен сообщениями позволяет наладить асинхронное взаимодействие между слабо связанными приложениями и гарантирует доставку сообщений от отправителя к получателю [9, 24, 28].

Основными понятиями технологии обмена сообщениями являются:

1) *сообщение* — наименьшая единица данных, которая может быть передана от отправителя к получателю;

2) *каналы* — логические адреса в системе обмена сообщениями. Данные передаются между приложениями по каналам сообщений. Реализация каналов зависит от конкретной системы. Например, несколько логических каналов могут использовать один физический канал. Логические каналы скрывают детали конкретной реализации от приложений.

Процесс обмена сообщениями между приложениями включает следующие этапы:

1) создание — отправитель создает сообщение, содержащее полезную информацию;

2) отправка — отправитель помещает сообщение в канал;

3) доставка — система обмена сообщениями доставляет сообщение с компьютера отправителя на компьютер получателя;

4) получение — получатель извлекает сообщение из канала;

5) обработка — получатель считывает полезную информацию.

Существуют два метода передачи сообщений от одной удаленной системы к другой — непосредственный обмен сообщениями и использование очередей сообщений. Передача сообщения напрямую возможна только в том случае, если принимающая сторона готова принять сообщение в тот же момент времени.

Основная идея очереди сообщений — использование посредника (менеджера очередей сообщений), что позволяет работающим в различное время приложениям взаимодействовать в гетерогенных сетях и системах и не требует одновременного подключения к сети. Очередь сообщений — это репозиторий, в котором хранятся сообщения в процессе их пересылки. Менеджер очереди сообщений передает сообщения из источника в место назначения. Таким образом, назначение очереди — маршрутизация сообщений и обеспечение их гарантированной доставки получателю (если получатель недоступен во время отправки сообщения, то очередь хранит его до тех пор, пока сообщение не будет доставлено).

Примерами промежуточного программного обеспечения, реализующего сервисы обмена сообщениями, являются Microsoft Message Queuing, IBM MQSeries, Sun Java System Message Queue. Эти системы позволяют выполнять следующие операции по использованию очередей:

- добавить сообщение в очередь;
- взять первое сообщение из очереди;

- проверить очередь на наличие сообщений;
- установить обработчик, вызываемый при появлении сообщения.

На рис. 3.15 представлен процесс пересылки сообщения с клиента на сервер. Сообщение записывается в очередь исходящих сообщений, и начинается процесс его доставки. Как только связь с сервером установлена, система обмена сообщениями отсылает все сообщения, накопленные за время ожидания подключения к серверу (или просто за время простоя). Полученные данные сервер записывает в очередь входящих сообщений. Теперь серверное приложение может в любой момент времени просмотреть полученную информацию и удалить ее. При создании сложных интеграционных решений используется маршрутизация сообщений. В этом случае сообщения проходят через ряд промежуточных менеджеров очереди сообщений.

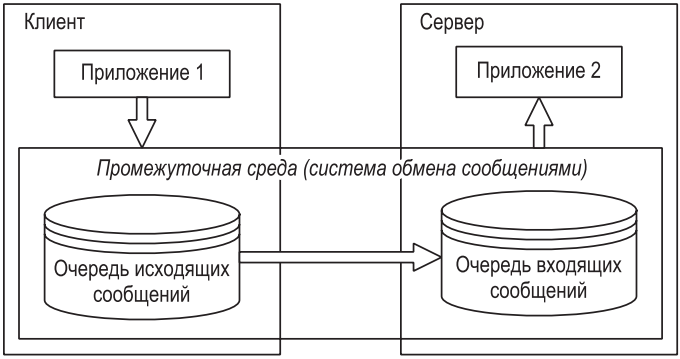

Рис. 3.15. Система обмена сообщениями

Преимущества и недостатки технологии обмена сообщениями представлены в табл. 3.12.

Таблица 3.12

Преимущества и недостатки технологии обмена сообщениями

Преимущества	Недостатки
Время функционирования сервера может быть не связано со временем работы клиентов (не требуется одновременная доступность отправителя и получателя) Гарантированная доставка сообщений от отправителя к получателю	Сложная модель программирования (событийно-управляемая модель программирования, создается множество обработчиков событий, реагирующих на входящие сообщения); сложность отладки и тестирования интеграционных решений

Окончание таблицы 3.12

Преимущества	Недостатки
Обеспечивается частный обмен небольшими порциями данных (синхронизация данных приложений) Считывать и обрабатывать заявки из очереди могут несколько независимых компонентов, что дает возможность достаточно просто создавать устойчивые и масштабируемые системы Сообщения могут быть преобразованы во время передачи без уведомления отправителя и получателя (независимость промежуточной среды от средства разработки компонентов и используемого языка программирования) Слабое связывание соединяемых приложений Возможность реализации различных способов доставки сообщений (рассылка всем получателям, рассылка по группам, маршрутизация)	Возможны задержка при доставке информации, нарушение порядка сообщений Не подходит для передачи больших объемов данных, снижает производительность Сложность реализации синхронного обмена

▸ Сервис-ориентированная интеграция

В настоящее время сервис-ориентированная интеграция (SOA-интеграция) — наиболее проработанный системный подход к решению проблем интеграции приложений. По сути, SOA-интеграция стирает грань между интеграцией приложений и созданием распределенного компонентного приложения. В стандарте Reference Model for Service Oriented Architecture 1.0 консорциума OASIS (Organization for the Advancement of Structured Information Standards) дано следующее определение SOA [8]: «SOA — это парадигма для организации и использования распределенных возможностей, которые могут находиться в различных областях собственности».

Центральные понятия SOA-интеграции — «сервис» и «процесс».

Сервис — это функция, являющаяся четко определенной, самодостаточной и не зависящей от контекста или состояния других сервисов.

Процесс (*бизнес-процесс*) определяется как независимая от реализации сервисов логика их взаимодействия.

Выделение сервисов на основе функциональности приложений имеет смысл, только если они могут использоваться неоднократно и в различных контекстах. Принято выделять поставщика, потребителя сервиса и компонент, обеспечивающий взаимодействие поставщика и потребителя (так называемый *брокер*).

Схема взаимодействия поставщика и потребителя сервиса представлена на рис. 3.16.

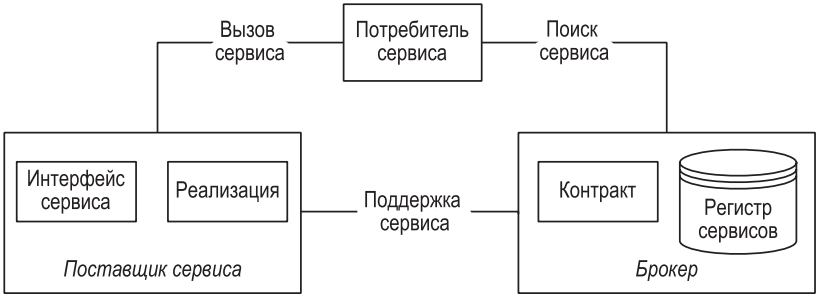

Рис. 3.16. Взаимодействие поставщика и потребителя сервиса

Сервисы имеют четко определенные интерфейсы, инкапсулирующие ключевые правила доступа к ним, и строятся без каких-либо допущений о том, кто будет использовать или потреблять их, то есть они слабо связаны с потребителями сервисов. Интерфейс сервиса используется для связи поставщика и потребителя, он позволяет идентифицировать сервис, описать входную и выходную информацию. Реализация сервиса определяет его функционал и бизнес-логику, причем детали реализации полностью скрыты от потребителя. Отметим, что сервис может быть реализован как Web-сервис, Java, .Net, CORBA.

Для взаимодействия потребителя с сервисом используется механизм виртуализации. Виртуальный сервис (прокси-объект) для реального сервиса представляет собой интерфейс, используемый потребителем сервиса. Потребители обращаются к прокси-объекту, который передает сообщения к поставщику сервиса. Взаимодействие потребителя сервиса с разными типами поставщиков схематично представлено на рис. 3.17.

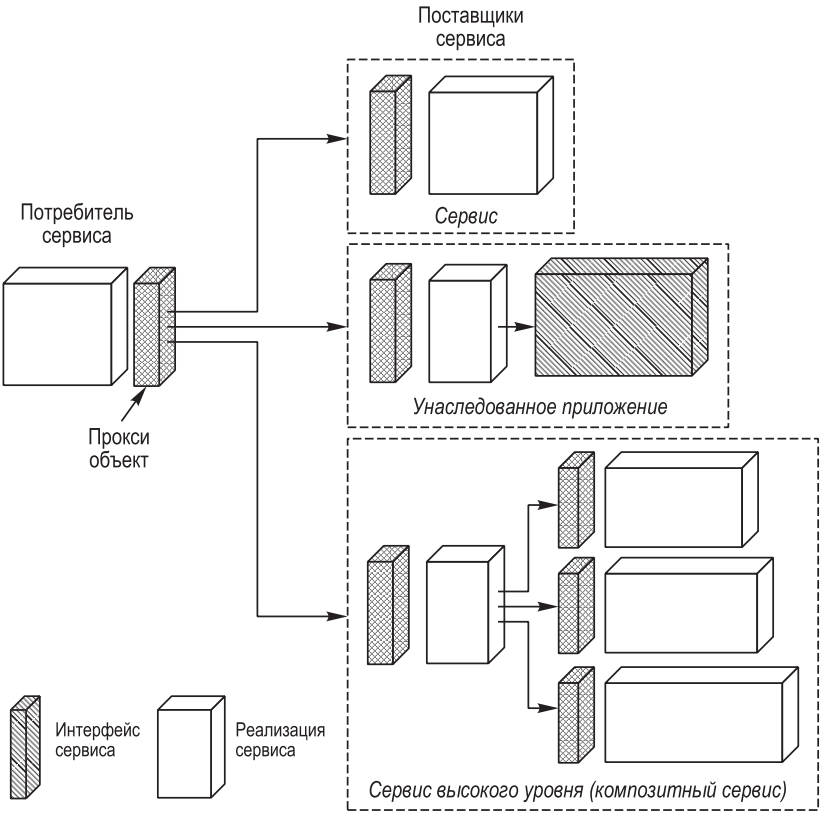

Рис. 3.17. Взаимодействие потребителя сервиса с разными типами поставщиков

Выделяют следующие функциональные типы сервисов:

- сервисы данных — содержат логику обработки данных и обеспечивают доступ к бизнес-данным, могут виртуально объединять данные из нескольких источников;
- бизнес-сервисы — реализуют элементарные бизнес-функции, которые не могут быть детализированы в рамках используемой бизнес-модели; могут комбинироваться для создания высокоуровневых сервисов;
- сервисы процессов — используются для управления бизнес-процессом, оркестровки сервисов, используемых бизнес-процессом, мониторинга состояния бизнес-процесса;

▪ сервисы взаимодействия — обеспечивают взаимодействие между приложениями и конечными пользователями;

▪ сервисы доступа — предназначены для включения унаследованных приложений в SOA-архитектуру; могут менять логику бизнес-функции существующего приложения в целях оптимизации взаимодействия сервисов или использовать несколько функций унаследованного приложения для обеспечения семантических требований к сервису.

Сервисы и процессы высокого уровня комбинируются из сервисов и процессов более низких уровней. Сервисы низких уровней реализуются внутри прикладных информационных систем организации либо являются результатом их работы. SOA-интеграция базируется на таких открытых стандартах, как SOAP, WSDL, UDDI, WS-BPEL, WS-CDL, BPMN.

На рис. 3.18 представлена референтная архитектура SOA-решения от компании IBM. Доступ к сервисам обеспечивается со стороны потребителей через ESB[1].

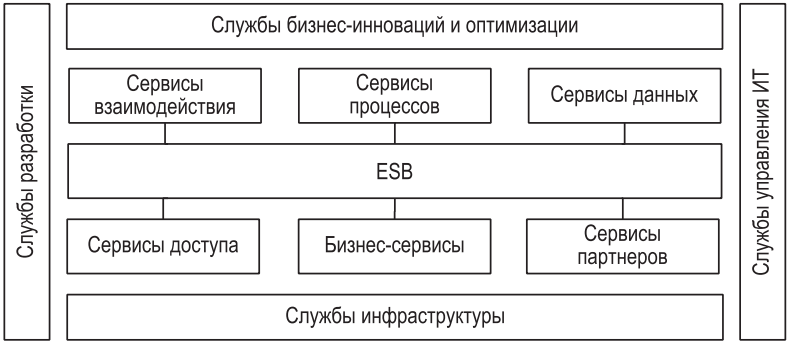

Рис. 3.18. Референтная архитектура SOA-решения

SOA-сценарий интеграции применим для:

▪ крупных территориально распределенных компаний, решающих задачу построения единого информационного пространства, объединения унаследованных приложений и вычислительных ресурсов;

▪ сервисных компаний, осуществляющих управление услугами и обеспечивающими их бизнес-процессами. Задачи вывода на рынок

[1] Подробнее о ESB см. п. 3.4.3.

новых услуг, оказания услуг потребителям с гарантированным уровнем качества, модификации услуг требуют согласованной работы нескольких информационных систем, включая системы сторонних компаний;

▪ управления процессами, требующими интенсивной обработки документов (задачи интеграции систем управления контентом, с САПР, системами управления проектами, отдельными модулями ERP).

Преимущества и недостатки сервис-ориентированной интеграции представлены в табл. 3.13.

Таблица 3.13

Преимущества и недостатки сервис-ориентированной интеграции

Преимущества	Недостатки
SOA основана на использование стандартов (WSDL, BPEL, UDDI), поддерживаемых большинством поставщиков информационных технологий, что обеспечивает ее независимость от платформ интегрируемых приложений	Нецелесообразно использовать в гомогенной информационной среде (например, задача интеграции информационной системы от одного производителя)
Исключается дублирование бизнес-функций приложений	Не подходит для эффективной работы в режиме реального времени
Обеспечивается слабое связывание между приложениями	Сложно правильно выделить сервисы для получения хорошо масштабируемого решения
Гибкая интеграция приложений позволяет быстро перестроить бизнес-процессы и адаптировать их к изменившимся внешним условиям	Проблема обеспечения целостности информации в рамках транзакций, охватывающих сервисы. Для транзакций, растянутых на множество шагов, сложно добиться целостности, особенно если транзакция затрагивает разные приложения и длится долго (дни, месяцы)
Создание хорошо масштабируемых интеграционных решений	Сложно добиться семантической согласованности информации в интегрируемых решениях
Полная автономность отдельных сервисов (сервисы независимо проектируются, версионируются и поддерживаются)	
Независимая масштабируемость отдельных сервисов. Резкий рост нагрузки на отдельный сервис не «тормозит» всю систему. В SOA-системе перегружается лишь отдельный сервис, и проблема решается увеличением его мощности	

3.3.3. Интеграция на уровне пользовательского интерфейса

Доступ к функциональности приложения осуществляется через пользовательский интерфейс путем моделирования операций ввода/чтения данных с экрана.

Данный тип интеграции называют *прочесыванием экрана* (Screen scraping), поскольку промежуточное программное обеспечение должно собрать информацию, отображаемую на экране, во время сеанса работы пользователя с системой — источником данных. Технически достаточно просто реализовать чтение данных, выводимых в интерфейс конечного пользователя. Более сложной задачей является моделирование поведения пользователя в процессе доступа к необходимым интерфейсам и данным. Моделирование поведения пользователя осуществляется специальным компонентом интеграционного решения — эмулятором терминала, который воспринимается приложением — источником данных как обычный терминал, но может управляться программно.

Шаблон интеграции на уровне пользовательского интерфейса представлен на рис. 3.19.

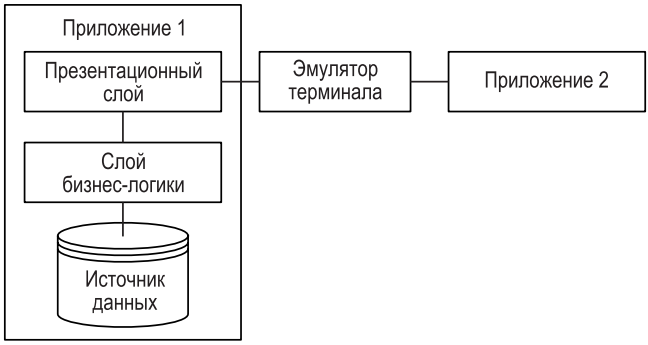

Рис. 3.19. Шаблон интеграции на уровне пользовательского интерфейса

Таким образом, интеграция на уровне пользовательского интерфейса обеспечивается взаимодействием трех компонент, характеристики которых представлены в табл. 3.14.

Таблица 3.14

Компоненты интеграционного решения

Компонент	Функции
Презентационный слой приложения-источника	Визуальное представление данных Обработка введенных пользователем данных и преобразование их в команды, выполняемые на уровне бизнес-логики
Эмулятор терминала	Моделирует сеанс работы пользователя с презентационным слоем Обеспечивает доступ к данным экрана через API Обеспечивает передачу команд приложения презентационному слою
Приложение	Использует данные приложения-источника Генерирует команды

Непроходящий интерес к данному стилю интеграции обусловлен широким использованием систем, клиентская часть которых реализована как Web-приложение. Доступ к пользовательскому интерфейсу Web-приложения легко получить через Интернет с использованием протокола HTTP. Презентационный слой Web-приложения обычно реализован как HTML-документ, из которого можно легко извлечь информацию программным путем.

Интеграция на уровне пользовательского интерфейса с успехом может применяться для решения задач, связанных со сбором данных с Web-сайтов для получения и предоставления новых типов услуг (например, широко распространенные сервисы, позволяющие сравнивать цены или предоставлять информацию о наличии товаров в интернет-магазинах).

Преимущества и недостатки интеграции пользовательского интерфейса представлены в табл. 3.15.

Таблица 3.15

Преимущества и недостатки интеграции на уровне пользовательского интерфейса

Преимущества	Недостатки
Работает с монолитными приложениями Практически исключает риск нарушения целостности данных приложения-источника (данные защищены бизнес-логикой приложения-источника) Не требует никаких изменений в приложении-источнике	Сложность поддержки, связанная с тем, что пользовательские интерфейсы изменяются чаще, чем программные интерфейсы и схемы данных Ограниченный доступ к данным (доступны только данные, отображаемые в пользовательском интерфейсе)

Окончание таблицы 3.15

Преимущества	Недостатки
	Отсутствие доступа к метаданным (теряется информация о типах данных, ограничениях, связях, логических элементах) Невысокая скорость (может потребоваться сбор данных из нескольких пользовательских форм) Сложность реализации (интеграционное решение должно моделировать поведение пользователя, например обеспечивать регулярную смену паролей в соответствии с политикой безопасности системы-источника, а это требует написания промежуточного кода большого объема)

3.4. Топология интеграционных решений

В интеграционном решении каждое приложение выступает в качестве ресурса, предоставляющего или использующего определенные сервисы, и представляет собой часть единой интеграционной архитектуры.

Существуют различные способы взаимодействия приложений. Простейшим вариантом является соединение системы с необходимым сервисом по типу «точка-точка». В более сложных случаях, когда система должна выступать в качестве событийно-управляемого поставщика или потребителя сервиса, взаимодействие между приложениями должно осуществляться через посредника (брокера или сервисную шину).

Выделяют следующие базовые шаблоны связывания приложений:

- «точка-точка»;
- брокер;
- шина сообщений.

Помимо них рассматривается шаблон «публикация/подписка» (см. п. 3.4.4), позволяющий поставлять информацию от поставщика к нескольким получателям.

3.4.1. Интеграция по типу «точка-точка»

Многие интеграционные проекты начинаются с объединения двух приложений. Простейшим способом связывания двух систем является интеграция типа «точка-точка».

Система, передающая данные, должна знать адрес получателя и уметь преобразовывать сообщение из формата системы-источника в формат системы-приемника. Интеграция осуществляется или через специальный программный интерфейс (API) или путем чтения/записи данных непосредственно в базу данных приложения. Как правило, для связи каждой пары приложений разрабатывается отдельный программный модуль, требующий поддержки и обновления в случае изменений в связанных приложениях.

В большинстве интеграционных сценариев необходимо реализовать определенную логику маршрутизации при передаче данных. При использовании связей типа «точка-точка» логику трансформации и маршрутизации приходится дублировать в каждом программном модуле, что увеличивает стоимость разработки и поддержки связующих модулей, а также создает проблемы при добавлении новых связей.

При хаотичном увеличении количества связей типа «точка-точка» между приложениями сложно обеспечивать своевременную синхронизацию данных, гарантировать качество переносимых данных и поддерживать преобразование бизнеса.

Преимущества и недостатки интеграции типа «точка-точка» представлены в табл. 3.16.

Таблица 3.16

Преимущества и недостатки интеграции «точка-точка»

Преимущества	Недостатки
Простота реализации при небольшом количестве интегрируемых приложений	Высокая стоимость поддержки интеграционного решения Плохая масштабируемость (добавление новой связи «точка-точка» требует добавления нового программного модуля) Сильная связь между приложениями Сложность управления интеграционным решением, состоящим из большого числа обособленных модулей взаимодействия Сложный алгоритм синхронизации данных

3.4.2. Брокер

Недостатки интеграции типа «точка-точка» могут быть нивелированы введением дополнительного слоя, содержащего брокер.

Брокер — это компонент, выступающий в качестве посредника и обеспечивающий связь между приложениями. В контексте объектно-ориентированной модели взаимодействия система-источник может послать брокеру маршализованный набор параметров при вызове метода системы-приемника, а система-источник — сообщение, которое вызовет сервис системы-приемника.

Брокер изолирует взаимодействующие системы друг от друга и обеспечивает взаимодействие между конечными точками приложений.

Основными функциями брокера являются:

- маршрутизация — определение местонахождения системы-приемника;
- регистрация конечных точек — механизм, который используют системы для обнаружения друг друга;
- трансформация — процесс преобразования данных из формата приложения источника в формат приложения приемника.

Маршрутизация может осуществляться в ходе прямого (прямой брокер) и непрямого (непрямой брокер) взаимодействия. Прямой брокер устанавливает соединение между конечными точками взаимодействующих систем. После установления связи две конечные точки взаимодействуют напрямую с использованием прокси-объектов на стороне клиента и сервера. Принцип действия прямого брокера представлен на рис. 3.20.

Примерами реализации прямого брокера являются Microsoft Distributed Common Object Model (DCOM), Common Object Request Broker Architecture (CORBA), Universal Description Discovery and Integration (UDDI), .NET Framework remoting.

Рассмотрим, каким образом DCOM управляет взаимодействием удаленных приложений. Когда приложение на сервере-источнике пытается создать и использовать объект на сервере-приемнике, DCOM определяет положение сервера-приемника с использованием конфигурационной информации, которая хранится в регистре. Далее создается клиентский прокси-объект на стороне системы источника и серверный прокси-объект (stub) на стороне системы приемника. Дальнейшее взаимодействие между системами осуществляется

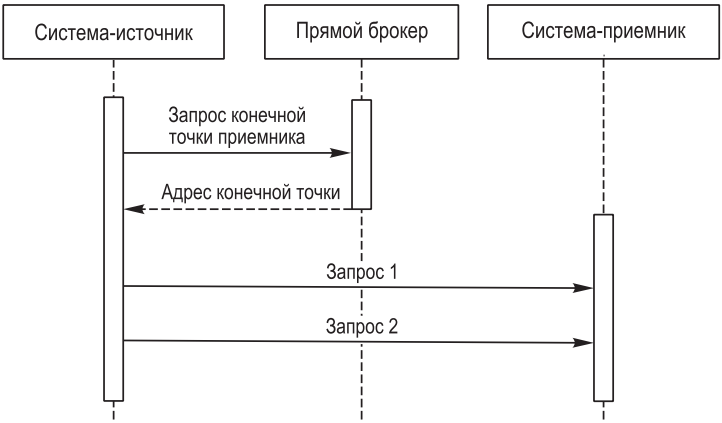

Рис. 3.20. Принцип действия прямого брокера

непосредственно между клиентским и серверным прокси-объетами с использованием соединения типа «точка-точка».

В случае непрямого брокера приложение отправляет сообщение посреднику вместе с логическим именем получателя. Брокер находит получателя и передает ему сообщение. Принцип действия непрямого брокера представлен на рис. 3.21.

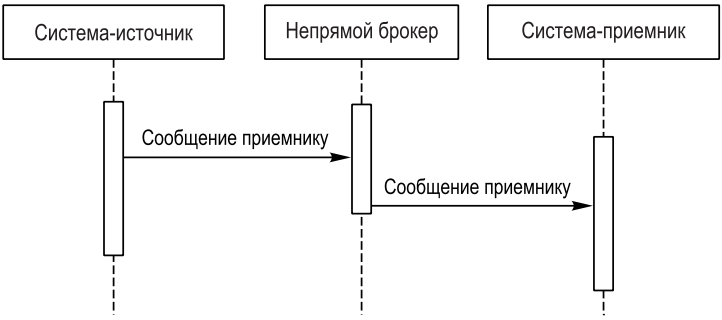

Рис. 3.21. Принцип действия непрямого брокера

Примером непрямого брокера является брокер сообщений, который может быть реализован с использованием программного обеспечения промежуточного слоя (MS BizTalk Server, TIBCO Business Integration).

3.4.3. Шина сообщений

Интеграционные решения достаточно быстро разрастаются, следуя потребностям бизнеса, при этом связи между приложениями создают жесткие зависимости между системами (получатель информации должен «узнавать» сообщения всех отправителей). Если архитектура интеграционного решения основана на связях «каждый с каждым» («точка-точка»), то количество связей растет по квадратичному закону с ростом числа взаимодействующих приложений[1]. Результатом является дорогое, плохо масштабируемое и плохо управляемое решение, не способное поддерживать преобразование бизнеса.

Решение данной проблемы состоит в том, что все приложения подключаются через специальный логический компонент, называемый *шиной сообщений*. При использовании шины сообщений приложение-источник и приложение-приемник больше не взаимодействуют напрямую. Приложение посылает сообщение в шину, которая доставляет сообщение всем приложениям-получателям через общую инфраструктуру, приложение-получатель взаимодействует только с шиной сообщений. В итоге у каждого приложения остается единственная связь.

Шина сообщений предназначена для решения следующих задач:

- распределение сообщений между приложениями;
- конвертирование транспортных протоколов между приложением-источником и приложением-приемником;
- конвертирование форматов сообщений между приложением-источником и приложением-приемником;
- управление бизнес-событиями различных источников.

Шина сообщений включает три основных элемента:

1) *набор согласованных схем сообщений*. Стандартом формата сообщений является XML с описанием метаданных при помощи XML Schema. Шина обеспечивает при необходимости трансляцию данных при помощи XSLT и XQuery;

2) *набор общих сообщений с командой*. При обмене информацией через шину необходимо иметь стандартный набор команд, понятных всем участникам;

3) *совместно используемую коммуникационную инфраструктуру*, которая играет роль межплатформенного универсального адаптера

[1] Если для полного связывания трех приложений необходимы три связи «точка-точка», то для связывания десяти приложений потребуется 45 связей.

между любыми приложениями, подключенными к шине. Для отправки сообщений обычно используется система обмена сообщениями, которая может включать функциональность маршрутизатора сообщений или поддерживать модель «публикация/подписка» (см. п. 3.4.4) для рассылки сообщений всем получателям. Шина сообщений использует один из трех вариантов модели «публикация/подписка» (рис. 3.22).

Рис. 3.22. Шаблон шины сообщений

В контексте архитектуры интеграционного решения, ориентированного на сервисы, следует рассмотреть концепцию корпоративной сервисной шины ESB (Enterprise Service Bus). Информационные системы/приложения рассматриваются здесь как поставщики и потребители сервисов. Все опубликованные в шине сервисы помещаются в единый реестр с возможностью повторного использования и управления политиками, связанными с сервисами.

Наиболее часто интеграционная шина используется для решения следующих задач:

- обмен сообщениями между приложениями;
- синхронизация справочной информации между различными приложениями;
- реализация сквозных бизнес-процессов;
- организация единой точки доступа к услугам (сервисам);
- унификация взаимодействий с партнерами;
- мониторинг бизнес-активности.

Преимущества и недостатки интеграционного решения, ориентированного на создание корпоративной сервисной шины, рассмотрены в табл. 3.17.

Таблица 3.17

Преимущества и недостатки интеграционного решения, ориентированного на создание корпоративной сервисной шины

Преимущества	Недостатки
Кроссплатформенность — обеспечение взаимодействия приложений, работающих на различных аппаратных и программных платформах	Сложность реализации Высокая стоимость
Слабая связность интегрируемых приложений обеспечивает возможность переноса/замены одного из приложений без последствий для других	
Масштабирование — возможность строить решения любого размера и нагруженности	
Гибкость и адаптивность — возможность реализовывать и изменять интеграционные сценарии по бизнес-запросу	
Использование открытых стандартов	
Уменьшение дублирования разработки приложений — использование функционала (сервисов), реализованного в одних приложениях, для целей других приложений и информационных систем вместо повторной реализации данного функционала	
Централизация управления и контроля — использование единых механизмов мониторинга, администрирования, модификации интеграционных процессов	

3.4.4. Интеграция по типу «публикация/подписка»

Данная модель взаимодействия приложений стала особенно популярной в последние годы, поскольку она обеспечивает регулярную доставку часто меняющейся информации от одного поставщика многим получателям. При этом получатели не должны конкурировать друг с другом, а доставка информации всем получателям должна быть гарантированной. Модель «публикация/подписка» часто реализуется в распределенных или мобильных архитектурах и позволяет потребителям информации получать уведомления

о появлении интересующей их информации. Примером является рассылка информации об актуальных курсах валют всем приложениям, использующим эту информацию.

В схеме «публикация/подписка» задействованы три участника (рис. 3.23):

- поставщик информации, которого принято называть *издателем* (publisher);
- потребитель информации, называемый *подписчиком* (subscriber);
- сервис событий, выполняющий роль посредника и осуществляющий управление подписками и уведомлениями.

Рис. 3.23. Взаимодействие «публикация/подписка»

Поскольку подписчик получает некоторое подмножество из набора сообщений, публикуемых издателями, существует механизм отбора сообщений, подходящих подписчику.

Выделяют три механизма фильтрации:

1) *темо-ориентированный механизм*. Издатель публикует сообщения на определенные темы (topics), организованные в виде иерархии. Подписчик подписывается на получение информации на необходимые ему темы и получает уведомления о подходящих сообщениях, опубликованных издателем. Подписка на тему в иерархии, содержащую подтемы, позволяет подписчику получать все сообщения, публикуемые в теме и ее подтемах. Идентификация тем производится по ключевым словам;

2) *содержимо-ориентированный механизм*. Сообщения доставляются подписчику, если атрибуты или содержание сообщения удовлет-

воряет ограничениям потребителя. Фильтры выбирают подходящие события из опубликованной информации с использованием языка подписки (SQL, XPath);

3) *типо-ориентированный механизм.* Оповещения делятся на типы, которые могут инкапсулировать атрибуты и методы. Получатель подписывается на определенный тип и определяет фильтр на основании атрибутов события. Фильтры могут быть применены удаленно для уменьшения загрузки сети. Подписчик определенного типа объектов получает только экземпляры классов необходимого типа и его подтипов.

Преимущества и недостатки интеграции по типу «публикация/подписка» рассмотрены в табл. 3.18.

Таблица 3.18

Преимущества и недостатки интеграции по типу «публикация/подписка»

Преимущества	Недостатки
Пространственное разделение — подписчики и издатели обычно не обладают информацией друг о друге Временно́е разделение — взаимодействующие стороны не обязательно должны быть доступны в одно и то же время. Издатель может опубликовать событие в момент, когда подписчик недоступен, и наоборот, подписчик может быть уведомлен о событии, когда издатель, опубликовавший данное событие, недоступен Асинхронное взаимодействие издателя и подписчика Хорошая масштабируемость решений за счет отсутствия явных зависимостей между взаимодействующими сторонами	Низкая производительность содержимо-ориентированных решений Невозможность подписаться на часть темы в темо-ориентированных решениях

Заключение

Для любого современного предприятия применение интеграционных технологий носит стратегический характер и обеспечивает несомненные конкурентные преимущества.

С ростом числа используемых систем интеграционные проекты все более усложняются и требуют от ИТ-специалистов умения принимать мотивированные решения по вопросам, связанным с организацией взаимодействия унаследованных приложений.

Разработка оптимальной интеграционной стратегии невозможна без знаний базовых технологий и стандартов удаленного взаимодействия систем. Применение референтных моделей (шаблонов), документирующих экспертные знания, позволяет правильно классифицировать задачу и сравнить возможные варианты ее решения.

Во второй части учебного пособия будут подробно рассмотрены примеры интеграционных проектов и дан сравнительный анализ промышленных интеграционных платформ.

Библиографический список

1. *Alexander et al.* A Pattern Language. — Oxford, 1977.

2. CORBA: [Электронный ресурс]. — URL: http://www.corba.org. Дата обращения: 05.10.2013.

3. Enterprise Connectivity Patterns: Implementing integration solutions with IBM»s Enterprise Service Bus products: [Электронный ресурс]. — URL: http://www.ibm.com/developerworks/webservices/library/ws-enterpriseconnectivitypatterns/index.html?S_TACT=105AGX99&S_CMP=CP. Дата обращения: 10.10.2013.

4. *Gamma et al.* Design Pattern — Elements of Reusable Object-Orientated Software. — Addison-Wesley, 1995.

5. IDC. The Digital Universe Decade — Are You Ready? IDC, 2010: [Электронный ресурс]. — URL: http://www.emc.com/digital_universe. Дата обращения: 21.06.2013.

6. Integration Patterns Overview: [Электронный ресурс]. — URL: http://www.enterpriseintegrationpatterns.com/eaipatterns.html. Дата обращения: 14.09.2013.

7. Integration Patterns. — Microsoft, 2004.

8. *MacKenzie*, *Laskey K.*, *McCabe F.*, *Brown P.F.*, *Metz R.* Reference Model for Service Oriented Architecture 1.0, OASIS Standard: [Электронный ресурс]. — URL: http://docs.oasisopen.org/soa-rm/v1.0. Дата обращения: 12.05.2013.

9. Microsoft Message Queuing (MSMQ) — промежуточная среда обмена сообщениями: [Электронный ресурс] // Microsoft. Microsoft

Message Queuing (MSMQ). — URL: http://www.intuit.ru/department/se/msfdev/6/1.html. Дата обращения: 13.05.2013.

10. OASIS Web Services Business Process Execution Language (WSBPEL) TC: [Электронный ресурс]. — URL: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel. Дата обращения: 15.06.2013.

11. *Takashi Kobayashi*, *Masato Tamaki*, *Norihisa Komoda*. Business process integration as a solution to the implementationof supply chain management systems//Information & Management. — 2003. — No. 40. — P. 769–780.

12. World Wide Web Consortium. SOAP Current status: [Электронный ресурс]. — URL: http://www.w3.org/standards/techs/soap#w3c_all. Дата обращения: 17.06.2013.

13. World Wide Web Consortium. Web Services Discription Language (WSDL) 1.1: [Электронный ресурс]. — URL: http://www.w3.org/TR/wsdl. Дата обращения: 17.06.2013.

14. World Wide Web Consortium. XML Technology: [Электронный ресурс]. — URL: http://www.w3.org/standards/xml. Дата обращения: 20.06.2013.

15. *Артамонов И.* Современные стандарты описания и исполнения бизнес-процессов: [Электронный ресурс]. — URL: http://ecm-journal.ru/post/Sovremennye-standarty-opisanija-i-ispolnenija-biznes-processov.aspx. Дата обращения: 13.06.2013.

16. *Бин Д.* XML для проектировщиков. Повторное использование и интеграция. — М.: КУДИНЦ-ОБРАЗ, 2004.

17. *Волкова И.А.*, *Головин И.Г.*, *Карпов Л.Е.* Системы программирования: учебное пособие. — М.: Издательский отдел факультета ВМК МГУ, 2009.

18. *Гандерлой М.*, *Джордан Д.*, *Чанц Д.* Освоение Microsoft SQL Server 2005: пер с англ. — М.: ИД «Вильямс», 2007.

19. *Горин С.В.*, *Крищенко В.А.* Поддержка разработки распределенных приложений в Microsoft .NET Framework. — М.: МГТУ им. Баумана, 2006.

20. *Зайден М.* XML для электронной коммерции. — М.: Бином: Лаборатория знаний, 2003.

21. Использование очереди сообщений MSMQ в платформе .NET: Практическое руководство: [Электронный ресурс]. — URL: http://msdn.microsoft.com/ru-ru/library/ms172497.aspx. Дата обращения: 22.05.2013.

22. *Машнин Т.С.* Web-сервисы Java. — СПб.: БХВ-Петербург, 2012.

23. Разработка Web-сервисов XML и серверных компонентов на Visual Basic .NET и Visual C# .NET: учебный курс. — М.: Русская Редакция, 2004.

24. Руководство Microsoft по проектированию архитектуры приложений // Корпорация «Майкрософт», 2009: [Электронный ресурс]. — URL: http://www.docme.ru/doc/6625/rukovodstvo-microsoft%C2%AE-po-proektirovaniyu-arhitektury-pril... Дата обращения: 29.08.2013.

25. *Самуйлов К.Е.*, *Чукарин А.В.*, *Яркина Н.В.* Бизнес-процессы и информационные технологии в управлении телекоммуникационными компаниями. — М.: Альпина Паблишерз, 2009.

26. Системы очередей сообщений: [Электронный ресурс]. — URL: http://www.intuit.ru/department/network/webspheremq/1/1.html. Дата обращения: 29.06.2013.

27. Технический обзор DCOM: [Электронный ресурс]. — URL: http://www.interface.ru/magazine/tcs/Archive/198/DCOM/dcom.htm. Дата обращения: 17.09.2013.

28. Унифицированные форматы электронных банковских сообщений: [Электронный ресурс]. — URL: http://cbr.ru/analytics/?Prtid=Formats. Дата обращения: 10.09.2013.

29. *Хабибулин И.Ш.* Самоучитель XML. — СПб.: БХВ-Петербург, 2003.

30. *Хайрук С.* О мульти- и моновендорной стратегии: [Электронный ресурс]. — URL: http://www.computerra.ru/cio/old/blog/index.php?page=post&blog=discussions&post_id=20. Дата обращения: 15.10.2013.

31. *Хоп Г.*, *Вульф Б.* Шаблоны интеграции корпоративных приложений: пер с англ. — М.: ИД «Вильямс», 2007.

32. *Хохгуртль Б.* C# и Java: межплатформенные Web-сервисы. — М.: КУДИНЦ-ОБРАЗ, 2004.

33. Школы консорциума W3C/XML: [Электронный ресурс]. — URL: http://xml.nsu.ru/xml/xml_home.xml. Дата обращения: 15.05.2013.

34. *Эммерих В.* Конструирование распределенных объектов. Методы и средства программирования интерпортабельных объектов в архитектурах OMG/CORBA, Microsoft/COM и Java/RMI: пер. с англ. — М.: Мир, 2002.

Оглавление

Table of Contents

Учебное издание

Ольга Анатольевна Морозова

ИНТЕГРАЦИЯ КОРПОРАТИВНЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

Учебное пособие

Редактор Т.А. Балашова
Оформление обложки и компьютерная верстка
Т.В. Иванниковой

Подписано в печать 02.12.13. Формат 60×90 $^{1}/_{16}$.
Бумага офсетная. Гарнитура Petersburg.
Усл.-печ. л. 8,75. Уч.-изд. л. 8,00.
Тираж 40 экз. Заказ № 880.

Финансовый университет
Ленинградский проспект, 49, Москва, ГСП-3, 125993
Отпечатано в ООП (ул. Олеко Дундича, 23)
Издательства Финансового университета

Для заметок

Для заметок