Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

И. И. Пилецкий, В. Н. Козуб

СОВРЕМЕННЫЕ СРЕДСТВА ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ. SOA-АРХИТЕКТУРЫ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве пособия для специальности 1-40 01 03 «Информатика и технологии программирования»

Минск БГУИР 2017

УДК 004.415.23(076)
ББК 32.971.32-02я73
П32

Р е ц е н з е н т ы:

кафедра многопроцессорных систем и сетей факультета прикладной математики и информатики Белорусского государственного университета (протокол №3 от 31.10.2016);

заведующий лабораторией информационного обеспечения научных исследований государственного научного учреждения «Объединенный институт проблем информатики Национальной академии наук Беларуси», кандидат технических наук, доцент Р. Б. Григянец

Пилецкий, И. И.

П32 Современные средства проектирования информационных систем. SOA-архитектуры : пособие / И. И. Пилецкий, В. Н. Козуб. – Минск : БГУИР, 2017. – 64 с. : ил.

ISBN 978-985-543-319-5.

УДК 004.415.23(076)
ББК 32.971.32-02я73

ISBN 978-985-543-319-5

Содержание

1. СЕРВИС-ОРИЕНТИРОВАННАЯ АРХИТЕКТУРА

1.1. Движущие силы SOA

«Ненадежные, негибкие системы». «Дорогостоящее в обслуживании программное обеспечение». Часто можно услышать такие фразы об ИТ. «Приходится иметь дело с системами, будто отлитыми из бетона». «Можно подумать, что приложения разрабатывались в расчете на то, что ничто и никогда не изменится». Автоматизация внесения изменений в системы почти отсутствует, поскольку для этого требуется вмешательство человека и перепрограммирование со стороны отделов ИТ, а сама подгонка технологии под бизнес-цели увязает в проблемах интеграции.

В повседневной практике термин «сервис» означает, что одна сторона предоставляет нечто, запрашиваемое другой стороной согласно условиям контракта. Если обслуживание стоит слишком дорого или не соответствует требованиям компании, то она может обратиться к другому поставщику того же самого сервиса. Парадигма сервисов включает в себя и систему взаимоотношений в процессе совместного оказания услуг, когда несколько партнеров объединяют свои усилия для удовлетворения насущных нужд клиентов.

Руководители бизнес-подразделений требуют от поставщиков приложений, чтобы они лучше обеспечивали поддержку динамических взаимоотношений сервисов. Бизнес-менеджеры хотят определять сервисы, а затем в сотрудничестве с внутренними или внешними ИТ-провайдерами развертывать их согласно условиям, прописанным в соглашении SLA (Service-Level Agreement). Наиболее продвинутые из них мечтают о том, чтобы самим определять процессы и сервисы, причем делать это непринужденно, изменяя их или вводя в действие одним щелчком мыши. Но пока это действительно лишь мечты, поскольку каждый, кто ясно представляет себе реальное положение дел, знает о том, сколько сил уходит даже на простую интеграцию процессов или согласование форматов данных с поставщиками и партнерами.

Ответом ИТ-отрасли на все эти вызовы времени стала сервис-ориентированная архитектура (SOA). Быстро развивающиеся стандарты XML- и web-сервисов революционным образом меняют то, как разработчики компонуют свои системы и интегрируют их в распределенные сети. Программистам больше не нужно иметь дело с жесткими фирменными языками и объектными моделями, царившими в эпоху архитектур CORBA и DCOM (Common Object Request Broker Architecture и Distributed Component Object Model). С появлением SOA в среду разработки приходит новый подход, характерный для распределенной обработки данных на базе web. Распространение архитектуры SOA

приведет к революционным изменениям средств BPM, EAI и связующего ПО на базе обмена сообщениями.

Основные проблемы ИТ:

– размножение систем ведет к беспрецедентной сложности;
– паутина технологий и поставщиков;
– 50 % бюджета направлено на сопровождение (Back Office);
– несоизмеримость затрат со значением для бизнеса;
– длительный возврат инвестиций;
– не достигаются требования бизнеса;
– запаздывание с внедрением новых услуг.

Сервис-ориентированная архитектура представляет собой стиль создания архитектуры ИТ, направленный на превращение бизнеса в ряд связанных сервисов – стандартных бизнес-задач, которые можно при необходимости вызывать через сеть.

Это может быть Интернет, локальная сеть или географически распределенная сеть, объединяющая различные технологии и сочетающая сервисы из Нью-Йорка, Лондона и Гонконга так, как если бы они были установлены на локальной машине. Для выполнения определенной бизнес-задачи можно будет объединять множество таких сервисов. Это позволит компании быстро адаптироваться под изменение условий и требований рынка.

Основные преимущества SOA приведены в табл. 1.1.

Таблица 1.1

Преимущества SOA

Преимущества	Вместе с SOA	Без SOA
1	2	3
SOA повышает способность бизнеса реагировать на потребности рынка благодаря более гибкой инфраструктуре	ИТ-инфраструктура более гибкая и быстрее реагирует на потребности бизнеса	ИТ-инфраструктура неспособна быстро реагировать на изменение рынка, запросы клиентов, партнеров и конкурентов
SOA позволяет бизнесу расти благодаря созданию ИТ-сервисов в соответствии с бизнес-процессами	Сервисы являются кирпичиками для ИТ-инфраструктуры	ИТ-инфраструктура состоит из инфраструктурных компонентов

Окончание табл. 1.1

1	2	3
SOA помогает бизнесу снижать расходы благодаря простой интеграции	Интеграция возможна «сцеплением» модулей	Интеграция возможна «жесткой» сцепкой
SOA помогает бизнесу снижать расходы благодаря повторному использованию активов	Новые услуги могут быть легко добавлены благодаря повторному использованию активов	Приложения должны быть полностью заменены
SOA снижает бизнес-риски, предлагая другое качество, простоту и управляемость	Простота и гибкость уменьшают риски	Сложность и негибкость повышают риски

Основные движущие факторы внедрения SOA приведены на рис. 1.1.

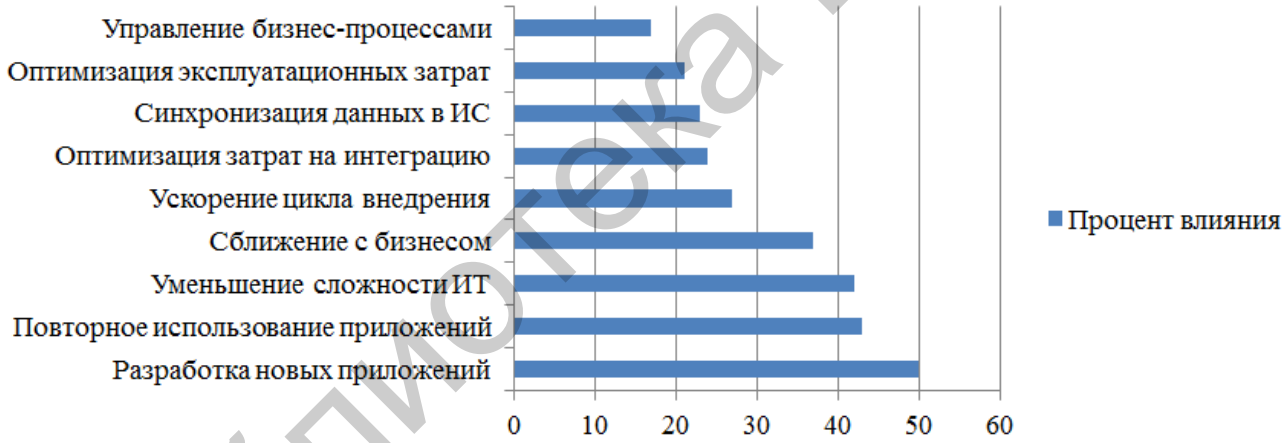

Рис. 1.1. Основные движущие факторы внедрения SOA

1.2. Основные принципы SOA

Популярность и широкое распространение при разработке информационных систем SOA-архитектура получила благодаря своим *основным принципам*:

1. Открытость исходного кода дает возможность компаниям, внедрившим у себя систему с SOA-архитектурой, обслуживать себя самим, участвовать в разработке новой функциональности и сообщать поставщику программного обеспечения о необходимости доработки SOA-платформы с целью ее дальнейшего развития. Этот принцип позволяет компании сохранить важное для бизнеса программное обеспечение, даже когда поставщик программного продукта прекратит его техническую поддержку.

2. Стандартизация производственных задач в системе управления в виде сервисов. SOA-система ориентирована на предоставление сервисов. SOA позволяет выстраивать логику процесса в системе управления отдельно от логики определенной производственной задачи (бизнес-логики), исполнение которой разрабатывается в виде стандартизованного сервиса. Весь процесс представляет собой объединение производственных задач. Так, например, процесс приемки товара на склад является объединением таких производственных задач, как сортировка прихода, идентификация товара, маркировка, палеттизация, печать актов приемки.

Работу сервиса в системе управления можно разделить на три этапа: получение данных на входе, обработка информации и формирование результата на выходе. Сформированные данные на выходе сервиса передаются на вход следующего сервиса или выводятся в том или ином виде пользователю системы (на монитор или в виде печатных отчетов). Сервис должен быть слабосвязанным и автономным.

3. Принцип повторного использования. SOA-архитектура предоставляет возможность комбинирования и многократного использования стандартизованных производственных задач системы (сервисов). Например, такие операции, как обработка серийных номеров при приемке товара, учет партии, алгоритмы перемещения, пополнения и отбора товара могут быть стандартизированы в системе управления в виде сервисов и многократно использоваться для построения процессов в системе управления складом.

4. Гибкость при разработке и внесении изменений в систему управления позволяет запустить новые процессы или модифицировать существующие, сохраняя основные процессы неизменными. Например, если необходимо добавить в процесс приемки задачу учета серийных номеров, бизнес-логика задачи реализуется в системе управления в виде сервиса (подпроцесса), и затем сервис добавляется в процесс.

5. SOA основана на принципе слабосвязанности, который ведет к уменьшению числа взаимозависимостей компонентов в процессах системы. Если в системе управления процессы имеют наименьшее количество связей, это упрощает процесс разработки новой функциональности и изменения процессов.

Однако при внедрении SOA-системы необходимо помнить, что эти преимущества обеспечивает не сама архитектура, а ее правильное использование.

1.3. Преимущества SOA

Создание сервис-ориентированной архитектуры может помочь подготовить как ИТ, так и бизнес-процессы к быстрым изменениям. Даже на ранних стадиях внедрения SOA организация получит преимущества за счет:

- роста доходов из-за создания новых рыночных возможностей и получения новых преимуществ от существующих систем;
- предоставления гибкой бизнес-модели, позволяющей быстрее реагировать на вызовы рынка;
- снижения стоимости в связи с устранением дублирующих систем, наращиванием возможностей уже имеющихся систем и сокращением времени до выхода на рынок;
- снижения рисков и количества дефектов благодаря повышению наглядности бизнес-операций.

Подход, основанный на SOA, позволяет сократить разрыв между тем, что вы хотите от бизнеса, и инфраструктурными инструментами, которые для этого нужны:

- сокращает время на разработку и размещение благодаря использованию уже созданных сервисов, пригодных для повторного использования, в качестве строительных блоков;
- позволяет интегрировать в масштабах предприятия даже те системы, которые всегда были изолированы, и облегчает поглощение и приобретение предприятий;
- сокращает время цикла и стоимость за счет перехода от ручных к автоматизированным транзакциям;
- облегчает взаимодействие с бизнес-партнерами благодаря повышению гибкости;
- предлагает адаптируемые и масштабируемые решения сложных проблем бизнеса, основанные на передовом опыте, таком как разделение приложения на слои и слабосвязанные компоненты.

1.4. Определение SOA

Сервис-ориентированная архитектура – это подход к созданию интеграционных архитектур, основанный на такой концепции, как «служба» (service).

Приложения взаимодействуют, вызывая службы, входящие в состав других приложений, и эти службы могут объединяться в более крупные последовательности, реализуя бизнес-процессы.

SOA – это подход к интеграционной архитектуре, основанный на концепции служб.

Бизнес-функции и инфраструктурные функции, которые необходимы для построения распределенных систем, реализуются в форме служб, которые, в сочетании или по отдельности, предоставляют прикладную функциональность либо приложениям, работающим с конечным пользователем, либо другим службам.

SOA определяет, что внутри любой архитектуры должен быть единый механизм взаимодействия служб. Этот механизм должен строиться на основе свободных связей и должен поддерживать использование формальных интерфейсов.

Википедия определяет SOA как модульный подход к разработке программного обеспечения, основанный на использовании распределенных, слабо связанных (loose coupling), заменяемых компонентов, оснащенных стандартизированными интерфейсами для взаимодействия по стандартизированным протоколам.

Программные комплексы, разработанные в соответствии с сервис-ориентированной архитектурой, обычно реализуются как набор web-служб, взаимодействующих по протоколу SOAP, но существуют и другие реализации (например, CORBA на основе REST).

В сети Интернет вызов удаленной процедуры может представлять собой обычный HTTP-запрос («GET» или «POST»; такой запрос называют «REST-запрос»), а необходимые данные передаются в качестве параметров запроса.

Интерфейсы компонентов в сервис-ориентированной архитектуре инкапсулируют детали реализации (операционную систему, платформу, язык программирования) от остальных компонентов, таким образом обеспечивая комбинирование и многократное использование компонентов для построения сложных распределенных программных комплексов, обеспечивая независимость от используемых платформ и инструментов разработки, способствуя масштабируемости и управляемости создаваемых систем.

1.5. Понятие службы

Поскольку мы определили SOA как подход к созданию интеграционных архитектур, основанных на службах, важно определить, что же представляет собой служба в данном контексте, чтобы можно было полностью описать SOA и понять, чего можно добиться с ее помощью.

На рис. 1.2 приведена схема связи служб с бизнес-задачами, или функциями.

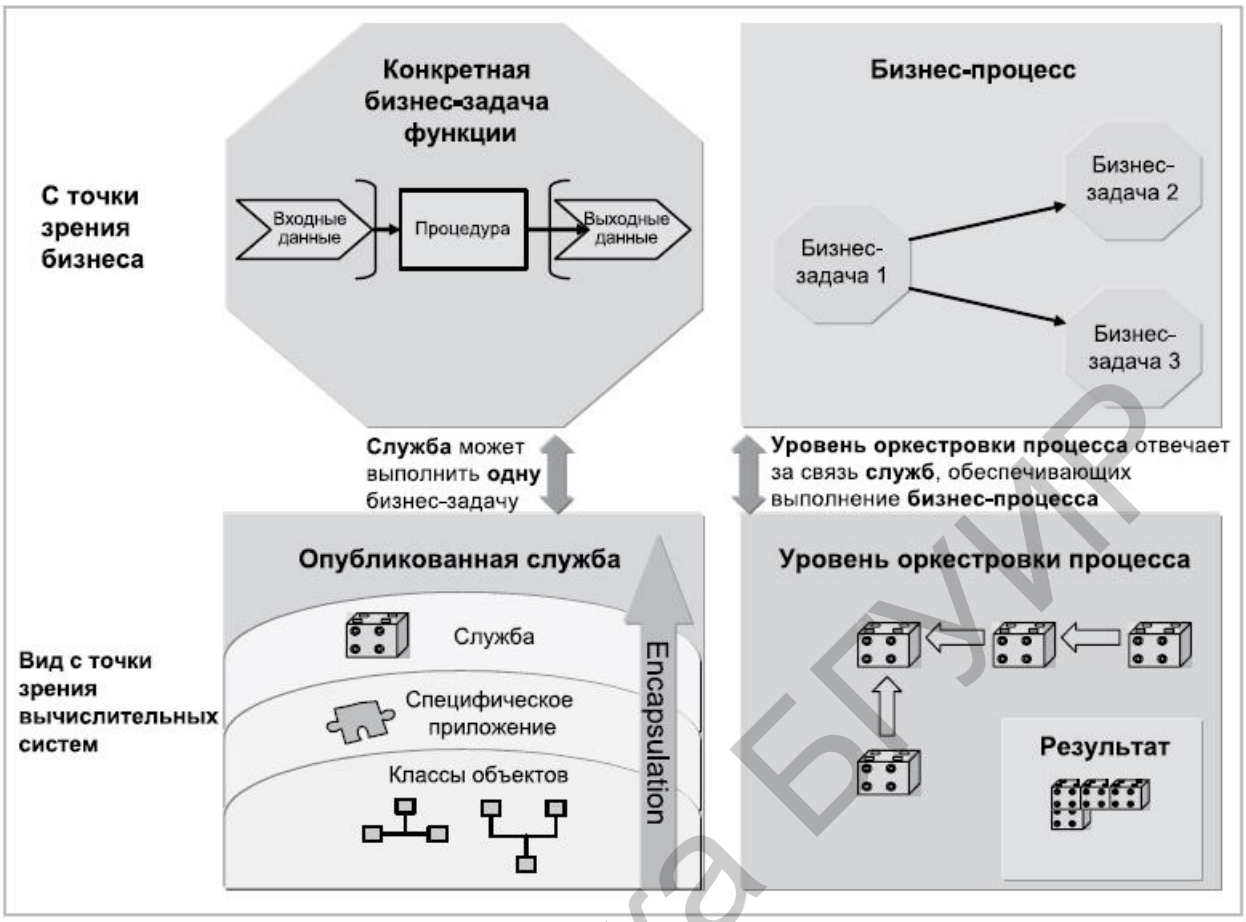

Рис. 1.2. Связь служб с бизнес-задачами, или функциями

Службой можно назвать любую дискретную функцию, которая может быть предложена внешнему потребителю. Это может быть отдельная бизнес-функция или набор функций, которые формируют процесс.

Общепринятыми являются положения, согласно которым службы:

- включают в себя многократно используемые бизнес-функции;
- определены с использованием формальных, не зависящих от реализации, интерфейсов;
- вызываются при помощи протоколов связи, обеспечивающих прозрачность местонахождения и возможность взаимодействия.

1.6. Фундамент SOA (IBM SOA Foundation)

Фундамент SOA – *это интегрированный открытый набор программного обеспечения,* лучшие методы и шаблоны, которые предоставляют вам все, что необходимо для того, чтобы приступить к созданию SOA.

Фундамент SOA обеспечивает полную поддержку жизненного цикла SOA с помощью интегрированного набора инструментов и компонентов времени

выполнения, которые позволяют вам повысить квалификацию и отдачу от вложений в общую инфраструктуру времени выполнения, инструментального оснащения и управления.

Компоненты построены по модульному принципу, что позволяет вам выбрать именно то, что вам необходимо для получения немедленного эффекта, и позволяет быть уверенным в том, что то, что вы выбрали, будет работать с тем, что вы добавите позже. Кроме того, фундамент SOA масштабируем, что позволяет вам начинать с малого и расти по мере потребностей бизнеса.

Фундамент SOA предоставляет широкую поддержку стандартов бизнеса и ИТ, облегчая обеспечение интероперабельности и переносимости приложений. Он также поможет вам использовать SOA для повышения ценности приложений и бизнес-процессов, которые используются бизнесом сегодня.

Эталонная архитектура SOA – это способ познакомиться с набором сервисов, которые входят в SOA. Эти возможности могут использоваться по мере возникновения необходимости, позволяя со временем наращивать возможности и решения уровня проекта по мере возникновения новых потребностей. Основой эталонной архитектуры является сервисная шина предприятия (enterprise service bus, ESB), которая упрощает коммуникации между приложениями и сервисами.

ESB позволяет провести объединенный взгляд на бизнес-приложения, которые находятся внутри или снаружи организации. Этот объединенный взгляд достигается за счет управления и координации потока событий (транзакции, сообщения, данные) между приложениями. Интеграция приложений, как и данных, возможна при помощи различных техник и технологий, зависящих от требований проекта, которые будут применены для разработки сервисов взаимодействия с внешними системами.

Общая информационная шина предприятия (ESB) – сервер, который управляет работой общей шины и состоит из многих компонентов (рис. 1.3).

Кроме интеграции приложений ESB позволяет реализовать интеграцию бизнес-процессов.

В настоящее время сервис-ориентированная архитектура является основой для реализации бизнес-процессов. SOA – это архитектура, а не готовое решение, она не поставляется как коробочный продукт, а создается под определенное предприятие. SOA характеризуется слабой связанностью и декомпозицией сложных задач на отдельные сервисы. Смысл SOA – в преобразовании монолитной ИТ-структуры во множество стандартизованных компонентов.

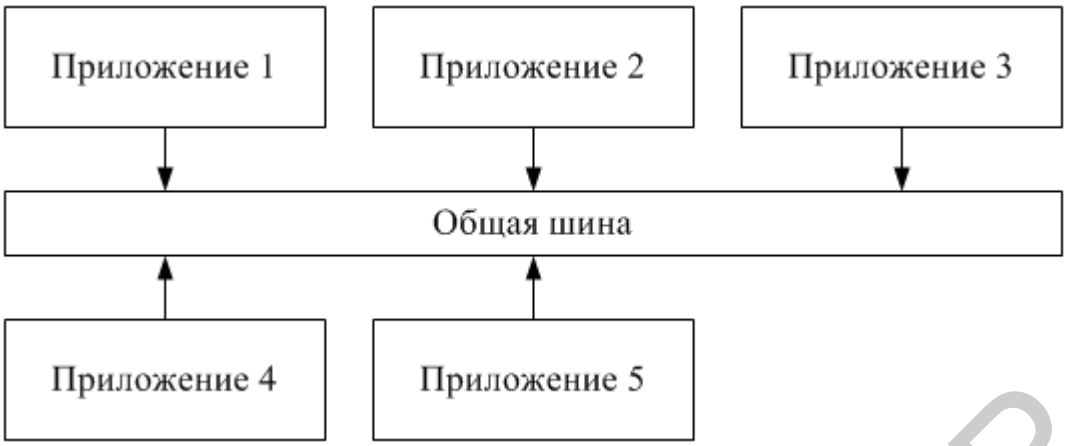

Рис. 1.3. Интеграция приложений

Сервисная ориентация позволяет строить бизнес-процессы в форме набора сервисов, а корпоративную систему представить как композитное решение, использующее сервисную шину (ESB), позволяющую минимизировать количество интерфейсов.

ESB обеспечивает: маршрутизацию сообщений между службами, преобразование протоколов службы и клиента, трансформацию форматов сообщений, обработку бизнес-событий из различных источников (рис. 1.4).

ESB является брокером, поддерживающим вызов сервиса в синхронном и асинхронном режимах. Она разрешает также передачу данных и уведомлений о событиях между приложениями. Она помогает клиентам найти провайдеры и управляет деталями взаимодействия между ними.

Сервис-ориентированная архитектура обеспечивает модульную, масштабируемую, переносимую среду для поддержки различных аспектов области бизнеса. Архитектура показывает тесную интеграцию с другими критическими аспектами ИТ, такими как безопасность, мониторинг, виртуализация и управление рабочей нагрузкой.

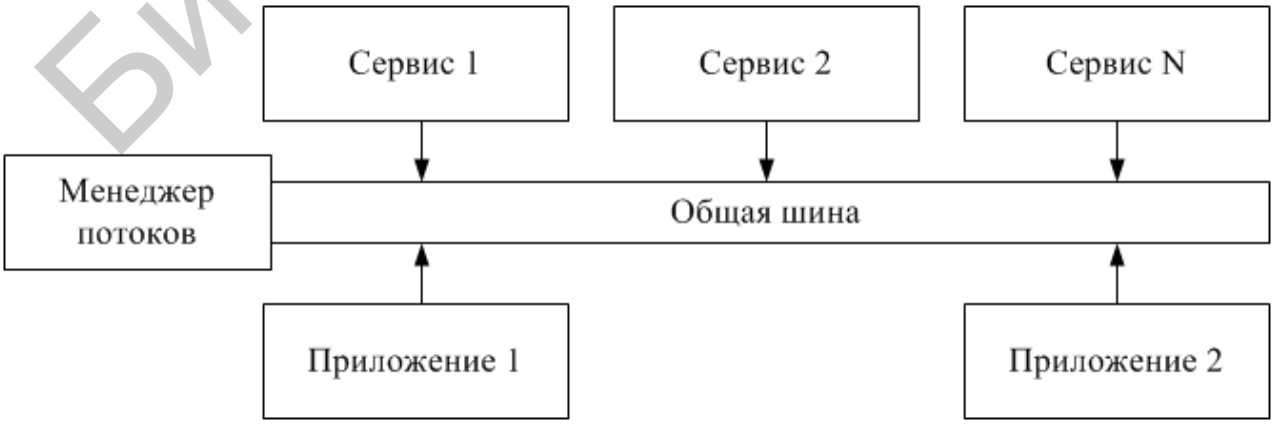

Рис. 1.4. Интеграция бизнес-процессов

2. WEB-СЕРВИСЫ

По мнению Джеймса Гослинга вице-президента компании Sun Microsystems, одного из создателей языка Java, *web-сервисы обеспечивают однородное представление неоднородной среды*.

2.1. Распределенная система

Распределенная система – программная система, компоненты которой функционируют на различных физических устройствах в сети, таких как персональные компьютеры, мейнфреймы, телефоны, планшеты и т. д.

Примеры распределенных систем:

- геторогенные приложения, состоящие из web-приложений (интернет-сайты), серверных приложений, баз данных, хранилищ, обслуживающего ПО, функционирующего на различной аппаратуре;
- информационные системы крупных предприятий.

Основные проблемы, существующие в распределенных системах:

- сложность;
- трудность интеграции неоднородных компонентов;
- обеспечение безопасности;
- управление ресурсами (выделение, организация одновременного доступа, поддержка пулов ресурсов);
- масштабируемость;
- поддержка транзакций;
- управление удаленными объектами (активация, создание, удаление, обращение по имени).

В индустрии программного обеспечения настали времена, когда задачи интеграции программных приложений, функционирующих на различных операционных системах, языках программирования и аппаратных платформах, не могут быть решены путем использования какого-либо частного решения. Традиционно проблемы были в жесткой связке приложений, когда при удаленном сетевом вызове одно приложение жестко привязано к другому названием функции с ее параметрами. Сложность представляет то, как сделать такие приложения слабосвязанными.

2.2. Понятие web-сервиса

Web-сервисы (web-службы) – это технология, которая позволяет приложениям взаимодействовать друг с другом независимо от платформы, на кото-

рой они развернуты, а также от языка программирования, на котором они написаны.

Web-сервис – это программный интерфейс, описывающий набор операций, которые могут быть вызваны удаленно по сети посредством стандартизированных XML-сообщений.

Web-службы – это самостоятельные модульные приложения, которые могут быть описаны, опубликованы, размещены и вызваны по сети.

Web-службы инкапсулируют бизнес-функции от простых, типа «запрос – ответ», до полномасштабных взаимодействий бизнес-процессов.

Службы могут создаваться заново или строиться на основе существующих приложений.

Web-сервисы используют XML. XML позволяет описать любые данные независимым от платформы способом, что, в свою очередь, приводит к слабосвязным приложениям. Кроме того, web-сервисы могут функционировать на более высоком уровне абстракции, анализируя, модифицируя или обрабатывая типы данных динамическим образом по требованию. Значит, с технической точки зрения web-сервисы могут обрабатывать данные значительно легче, предоставляя возможность программному обеспечению взаимодействовать более открыто (рис. 2.1).

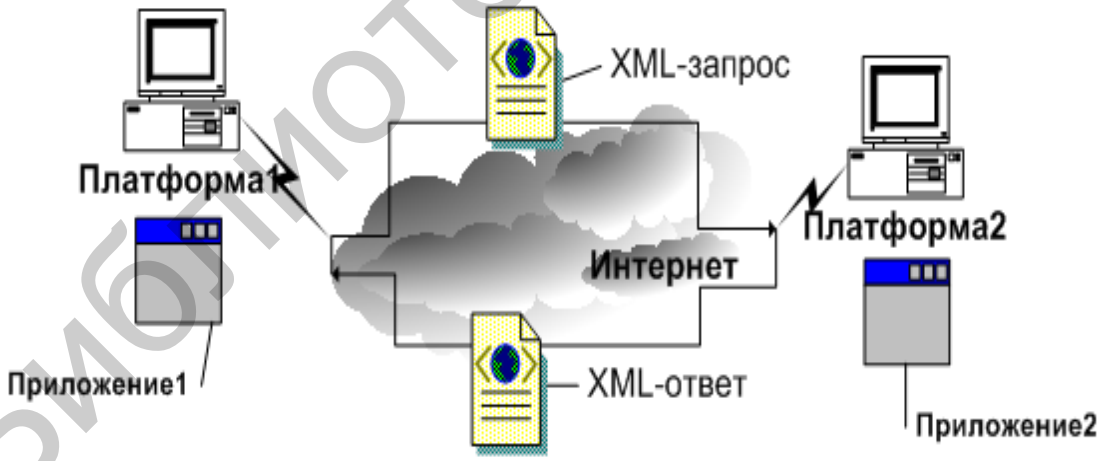

Рис. 2.1. Взаимодействие web-сервисов

На высоком концептуальном уровне мы можем рассматривать web-сервисы как единицы приложения, каждая из которых занимается выполнением определенной функциональной задачи. Если подняться на уровень выше, то эти задачи можно объединить в бизнес-ориентированные задачи для выполнения определенных бизнес-операций. Таким образом, после того как технические

специалисты разработали web-сервисы, архитекторы бизнес-процессов могут объединить их для решения конкретных бизнес-задач. Если взять за аналогию автомобиль, то при сборке кузова, двигателя, трансмиссии и других составляющих архитектор бизнес-процессов может брать двигатель целиком как законченный узел, не вдаваясь в подробности тех составляющих, из которых собран каждый двигатель. Кроме того, динамическая платформа означает, что двигатель может работать с трансмиссией или другими компонентами автомобиля от других производителей.

Из последнего рассмотренного аспекта вытекает, что web-сервисы помогают в рамках организации преодолеть разрыв между техническими специалистами и людьми, далекими от тонкостей технологий. Web-сервисы упрощают процесс понимания технических операций. Последние могут описывать события и различные типы деятельности, а технические специалисты могут ассоциировать их с соответствующими сервисами.

При использовании универсально описанных интерфейсов в хорошо спроектированных задачах web-сервисы становится значительно легче использовать повторно, а, следовательно, и приложения, которые они представляют. Повторное использование означает лучший возврат инвестиций в программное обеспечение, поскольку, используя те же ресурсы, вы можете получить больше. Это позволяет руководителям рассматривать новые возможности использования существующих приложений или предложить их партнерам для использования в новых задачах, повышая тем самым степень взаимодействия между партнерами.

Следовательно, основной проблемой, для которой web-сервисы могут служить решением, является интеграция данных и приложений, а также преобразование технических функций в бизнес-ориентированные задачи. Эти два аспекта позволяют различным компаниям взаимодействовать на уровне процессов или приложений с их партнерами, оставляя при этом возможность динамически адаптироваться к новым ситуациям или работать с различными партнерами по требованию.

Таким образом, благодаря своей простоте и широким возможностям, web-сервисы позволяют решать широкий круг задач, связывая вместе различные части программного обеспечения в рамках единой платформы. Этим обусловлена их популярность в сфере web-разработки: и крупные корпоративные порталы, и небольшие web-сайты используют web-сервисы.

2.3. Основные технологии web-сервисов

Рассмотрим основные технологии, используемые для построения web-сервисов:

- XML (расширяемый язык разметки, Extensible Markup Language) – это язык разметки, который лежит в основе большинства спецификаций, используемых в web-сервисах. XML – это исходный язык, с помощью которого можно описать любые данные в структурированном виде, независимо от их представления в конкретном устройстве;
- SOAP (простой протокол доступа к объектам, Simple Object Access Protocol) – это сетевой, транспортный и программный язык, а также не зависимый от платформы протокол, позволяющий клиенту вызвать удаленный сервис. Сообщения имеют формат XML;
- WSDL (язык описания web-сервиса, Web Services Description Language) – это интерфейс, основанный на XML, а также язык описания реализации. Поставщик сервиса использует WSDL-документ для описания операций, выполняемых web-сервисом, а также параметров и типов данных, которые он использует. WSDL-документ также содержит информацию для доступа к сервису;
- UDDI (универсальное описание, поиск и взаимодействие, Universal Description, Discovery and Integration) – это клиентский API, а также серверная реализация на основе SOAP, которые можно использовать для хранения и получения информации о поставщиках сервисов и web-сервисах;
- WSIL (язык обследования web-сервисов, Web Services Inspection Language) – это спецификация на основе XML, которая позволяет находить web-сервис без использования UDDI. Тем не менее, WSIL также можно использовать в сочетании с UDDI, т. е. он является независимым от UDDI и не подменяет его.

2.4. Использование web-сервисов

Web-сервис позволяет компьютерным программам стандартизировано общаться между собой.

Web-сервис является абстрактным компонентом – равно как абстрактна концепция диалога между людьми. В диалоге участвуют двое и более людей, разговаривающих на известном им языке.

Основные строительные блоки SOA, основанные на web-сервисах, приведены на рис. 2.2.

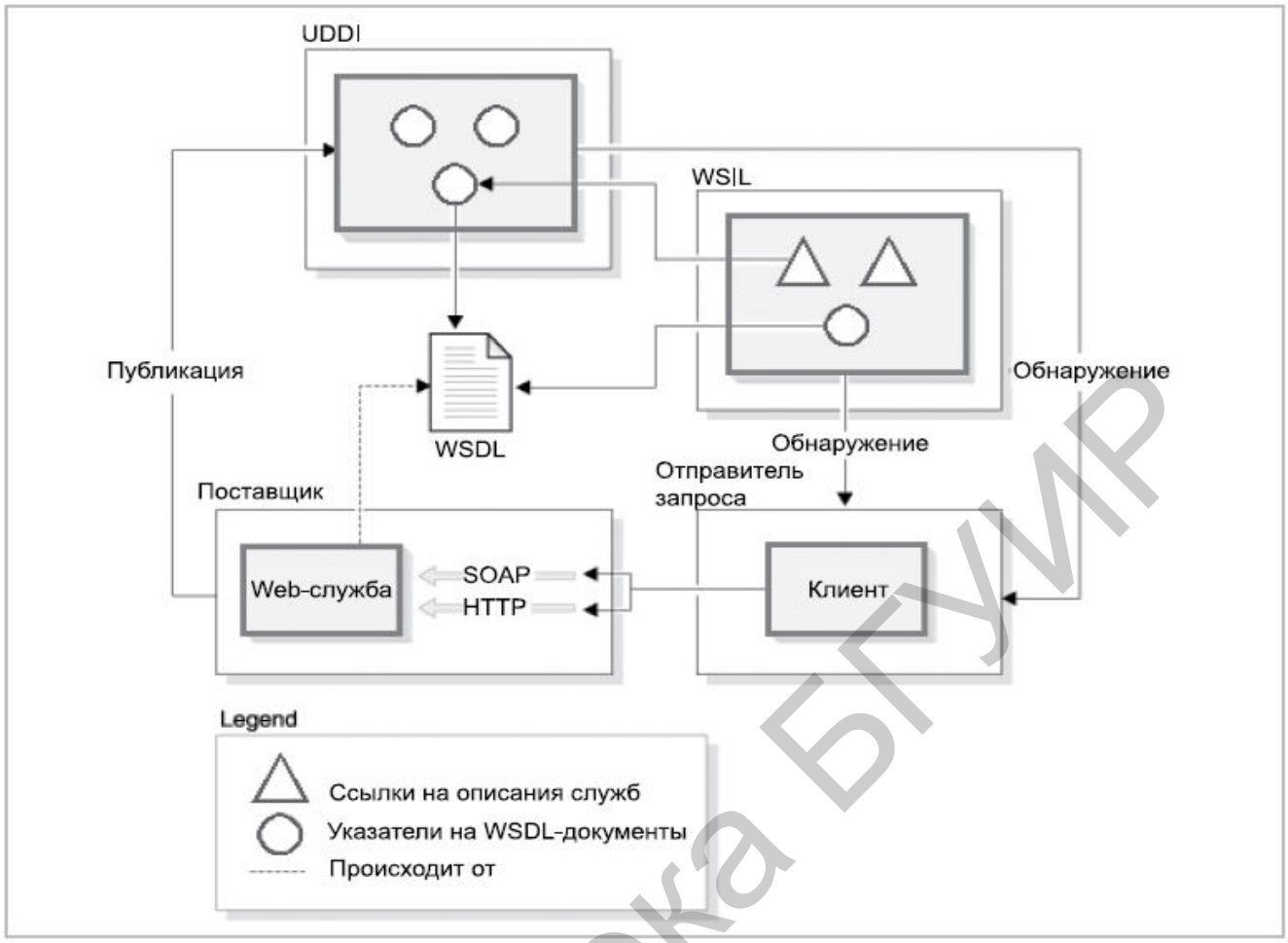

Рис. 2.2. Основные строительные блоки SOA

Во время обычного «диалога» программы общаются, используя знакомый им язык, на одном компьютере, иногда – по сети. Благодаря web-сервисам программы могут как находиться на одном компьютере, так и размещаться на разных машинах в разных точках земного шара, соединенных через Интернет. Программы и компьютеры не обязательно должны быть одинаковыми: это могут быть как две программы Microsoft.NET на одном ноутбуке, так и программа Java на сервере iSeries, программа C++ на компьютере с ОС Linux.

Как было сказано ранее, в коммуникациях посредством web-сервисов используются следующие стандартные технологии (рис. 2.3 и 2.4):

- язык XML;
- протокол SOAP;
- библиотека описаний web-сервисов (WSDL);
- универсальное описание, поиск и взаимодействие сервисов UDDI.

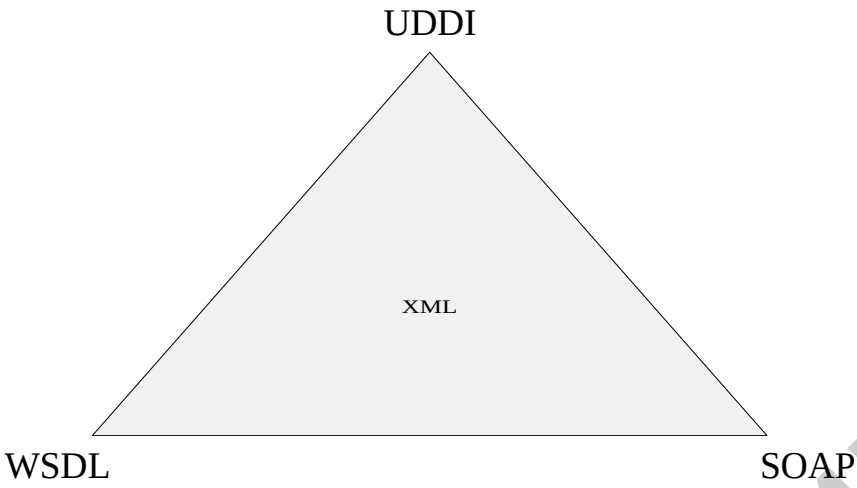

Рис. 2.3. Технологии взаимодействия

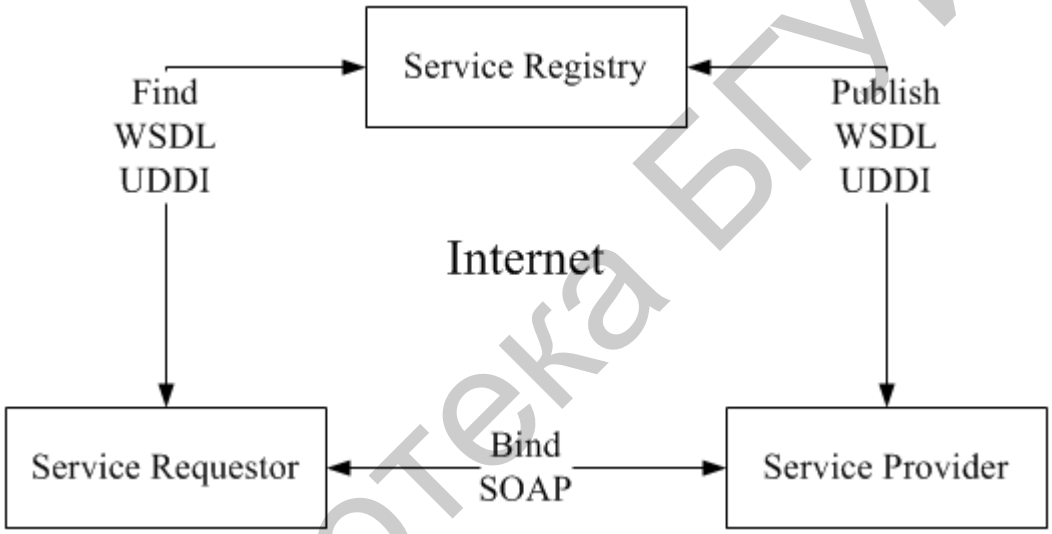

Рис. 2.4. Взаимодействие на основе web-сервисов

2.5. Язык XML

Язык XML используется для общения между компонентами web-сервисов. Сообщения, пересылаемые между приложениями, равно как определяющие web-сервис файлы, имеют формат XML. Ниже на примере показана структура простого файла XML.

Пример структуры простого файла XML:

```
<?xml version=”1.0” encoding=”UTF-8”?>
    <person xmlns=”um:mydomain:people” job=”developer”>
        <firstname>Петр</firstname>
        <lastname>Петров</lastname>
        < birthday >1980-01-01</birthday>
    </person>
```

Как можно видеть, некоторая информация в файле (такая, как имя, фамилия) окружена тегами, заключенными в треугольные скобки. Имя Петр показано как <firstname>Петр</firstname>. Также есть элементы, в которые вложены другие элементы, например, в элемент <person> вложены элементы <firstname>, <lastname> и <birthday>.

Разработка web-сервисов на языке XML дает немалые преимущества.

Структура и грамматика XML аналогична структуре остальных языков программирования, поэтому взаимодействующим с web-сервисами программам не надо проводить структурный анализ файлов XML напрямую.

Файлы XML текстовые, и их может прочитать человек (иными словами, зная язык XML, можно открыть файл XML в текстовом редакторе и понять его содержимое). Это может помочь при отладке программ, использующих XML.

XML позволяет использовать в сообщениях любую стандартную кодировку, поэтому сообщения можно писать как на английском, так и на русском или японском языках.

XML позволяет пользоваться так называемым пространством имен, в котором можно предопределить желаемую структуру файлового элемента с определенным именем. Например, вы можете определить элемент Price, который всегда должен быть числом с плавающей точкой, или PersonName, включающий в себя два строковых подэлемента: FirstName и LastName.

Но язык XML имеет жесткую структуру, поэтому любое неправильное форматирование сообщения XML приводит к сбою анализа всего сообщения (даже если проблему легко интерпретировать или пропустить). Однако если вы используете стандартную библиотеку для генерации файлов XML (что и делается при создании web-сервисов), библиотека сама проверяет правильность форматирования.

Сообщение XML хранится в обычном текстовом файле, а потому занимает больше места, чем его эквивалент в другом формате.

2.6. Simple Object Access Protocol

Как было сказано ранее, *общение* web-сервисов ведется в формате XML, однако это решает лишь половину проблемы. Приложения могут разобрать сообщение, но им неизвестно, что делать с полученным после анализа результатом. Инструкция, описывающая правила форматирования сообщений XML для web-сервисов, известна как SOAP (Simple Object Access Protocol). В пакете SOAP определена структура сообщений, благодаря чему программы знают, как отправлять и интерпретировать данные. Базовая структура сообщения SOAP показана на рис. 2.5.

Рис. 2.5. Базовая структура сообщения SOAP

На языке XML сообщение SOAP может выглядеть так:

```
<SOAP-ENV:Body>
    <ns1:GetStockInfo xmlns:ns1="urn:thisNamespace">
        <symbol>FXX</symbol>
    </ns1:GetStockInfo>
</SOAP-ENV:Body>
```

В базовом случае пакет SOAP включает в себя тело SOAP, в котором находятся передаваемые данные. Иногда еще есть необязательный заголовок SOAP (внутри пакета перед телом), содержащий дополнительную информацию.

Следующий пример иллюстрирует запрос SOAP, называемый GetLastTradePrice, который позволяет клиенту послать запрос о последних котировках определенных акций.

```
POST/StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
    "http://schemas.xmlsoap.org/soap/envelope/"
    SOAP-ENV:encodingStyle=
        "http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Body>
        <m:GetLastTradePrice xmlns:m="Some-URI">
            <symbol>DXX</symbol>
```

```
            </m:GetLastTradePrice>
        </SOAP-ENV:Body>
    </SOAP-ENV:Envelope>
```

В первых пяти строчках (часть заголовка HTTP) указывается тип сообщения (POST), хост, тип и длина информационного наполнения, а заголовок SOAPAction определяет цель запроса SOAP. Само сообщение SOAP представляет собой документ XML, где сначала идет конверт SOAP, затем элемент XML, который указывает пространство имен SOAP и атрибуты, если таковые имеются. Конверт SOAP может включать в себя заголовок (но не в данном случае), за которым следует тело SOAP. В нашем примере в теле содержится запрос GetLastTradePrice и символ акций, для которых запрашиваются последние котировки.

Ответ на этот запрос может выглядеть следующим образом:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
        "http://schemas.xmlsoap.org/soap/envelope/"
        SOAP-ENV:encodingStyle=
            "http://schemas.xmlsoap.org/soap/encoding/"/>
        <SOAP-ENV:Body>
            <m:GetLastTradePriceResponse xmlns:m="Some-URI">
                <Price>24,5</Price>
            </m:GetLastTradePriceResponse>
        </SOAP-ENV:Body>
    </SOAP-ENV:Envelope>
```

Первые три строки – это часть заголовка HTTP. Само сообщение SOAP состоит из конверта, который содержит ответ на исходный запрос, помеченный GetLastTradePriceResponse, и включает в себя возвращаемое значение, в нашем случае – 25,5.

Примечание. Хотя формат SOAP стандартный и имеет одинаковые инструкции, необходимо помнить, что разные производители могут немного поразному воплощать эти инструкции. Например, структура именных пространств и XML в сообщении SOAP, сгенерированном Apache Axis, может сильно отличаться от структуры, сгенерированной Microsoft.NET. Однако правильно написанный клиент или сервер может обработать любое правильно написанное в соответствии с инструкциями SOAP сообщение.

На случай проблем в теле SOAP содержится информация об ошибке в форме SOAP Fault.

Fault – это структура XML, содержащая описание ошибки, например:

```
<SOAP-ENV:Fault>
    <faultcode>SOAP-ENV:Server</faultcode>
    <faultstring>Server Error</faultstring>
    <detail>Database not available</detail>
</SOAP-ENV:Fault>
```

Хотя сообщение SOAP и хранится в текстовом виде в формате XML, из-за специфичности определений пространств имен и свойств элементов его порой бывает непросто понять или интерпретировать вручную. К счастью, программ и библиотек, способных создать или интерпретировать вам сообщение SOAP, много, и они берут на себя большую часть сложностей.

2.7. Web Service Definition Language

Для того чтобы человек или компания смогли вызвать web-сервис, необходимо знать основную информацию, такую как: где сервис расположен, как сервис может быть связан и т. д. Подобная информация представляется с помощью языка описания web-сервисов WSDL.

WSDL – язык, основанный на XML, используемый для создания документов, которые содержат важную информацию о том, как web-сервисы могут быть расположены и работать. WSDL часто произносится как «Whiz-dull».

Когда говорят о процессе *публикации* web-сервиса, имеют в виду программу, *публикующую файл WSDL* и позволяющую иным программам пользоваться соответствующим сервисом. Программы, публикующие web-сервисы, называются *провайдерами*. Когда говорят об *использовании* web-сервиса, имеют в виду программу, отправляющую вызов к web-сервису на другой машине. Пользователи web-сервисов называются *клиентами*.

Для выполнения web-сервиса клиент должен определить его местонахождение, а затем загрузить эти данные в документ WSDL, в котором описаны подробности использования сервиса.

Клиент может получить файл WSDL различными способами, например:

- документ может быть расположен в доступном для поиска общественном каталоге UDDI. В этом случае любой клиент может найти web-сервис и выполнить его;
- документы WSDL могут быть найдены другими способами, например, через запросы HTTP, FTP и даже через электронную почту. Они также могут располагаться в частных каталогах UDDI.

На рис. 2.6. приведена структура файла WSDL версии 1.1 и версии 2.0.

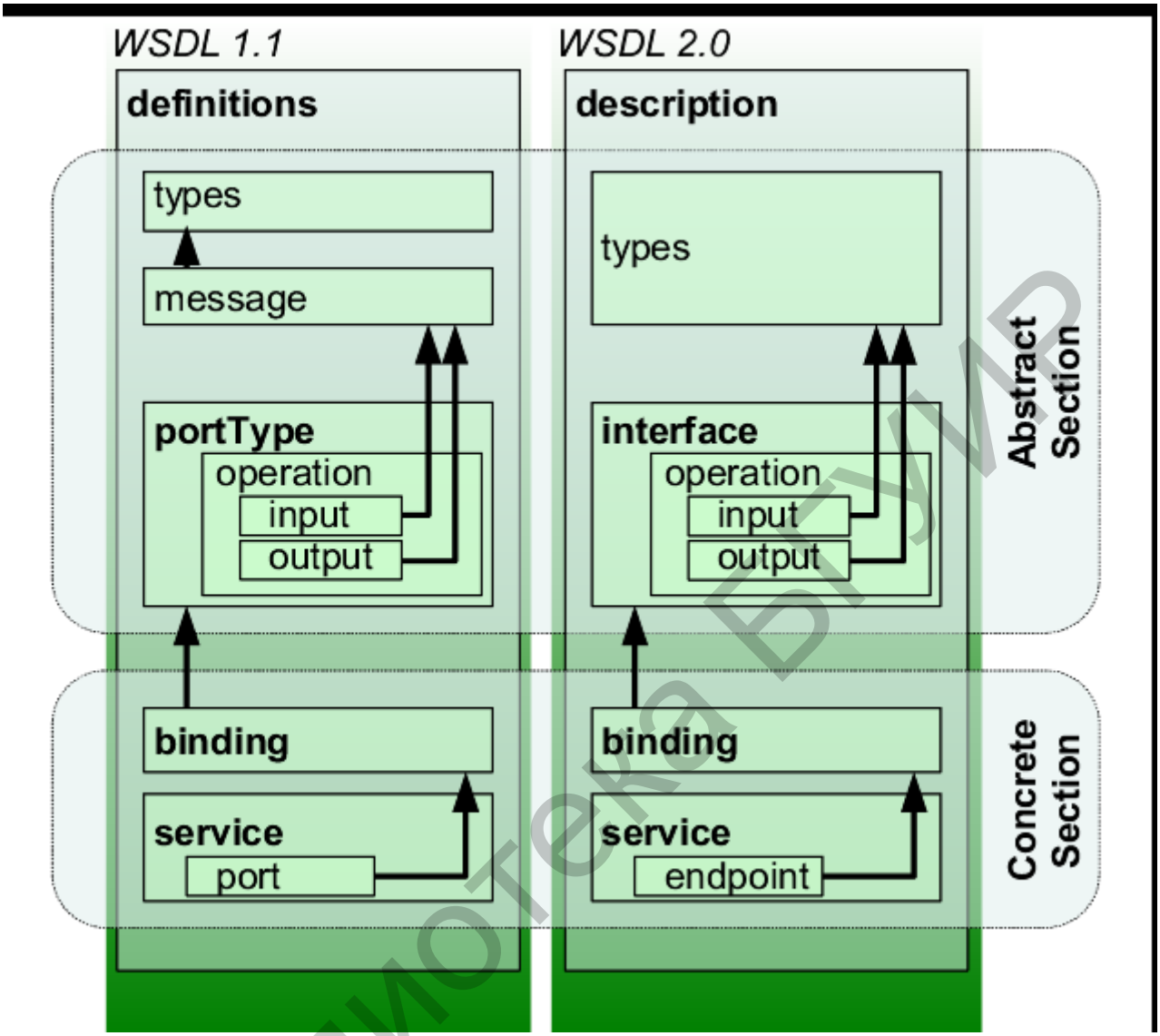

Рис. 2.6. Структура файла WSDL версии 1.1 и версии 2.0.

В файле WSDL провайдером определены методы сервиса, которыми могут пользоваться другие программы. Также в файле есть информация о таких правилах форматирования сообщений SOAP, как используемые пространства имен, порядок и структура параметров и дополнительная информация, которую необходимо включить в заголовок SOAP или HTTP.

Рассмотрим основные элементы WSDL-документа:

– <type><message>. Данные элементы описывают информацию, которая передается в web-сервис. Элемент <message> описывает непосредственно web-сервис;

– <binding>. Этот элемент детализирует то, как информация будет передаваться между клиентом и web-сервисом, также <binding> содержит сведения о протоколе и формате данных;
– <portType> описывает операции, которые будут поддерживаться web-сервисом;
– <service> содержит подробности о местоположении web-сервиса в сети.

Приведем пример web-сервиса для доступа к данным в БД по протоколу SOAP, включающего в себя следующий набор методов:

1) getAllTables:

String[] getAllTables(String user)

Этот метод позволяет получить одномерный список всех таблиц БД. Входным параметром данного метода является имя пользователя (String user) (пользовательской системы). Выходным параметром – одномерный список таблиц БД (String[]);

2) getTableInfo:

TableRow[] getTableInfo(String user, String table_name)

Этот метод позволяет получить метаинформацию об интересующей пользователя таблице БД. Входными параметрами данного метода являются имя пользователя (String user) и имя интересующей пользователя таблицы БД (String table_name), выходным параметром – двухмерный список, описывающий структуру таблицы БД (TableRow[] –содержит внутри String[]);

3) getTableData:

TableRow[] getTableData(String user, String table_name,
String[] columns, String[] sql_filtre)

Этот метод является основным методом сервиса и позволяет выгружать данные из интересующей пользователя таблицы БД. Входными параметрами данного метода являются имя пользователя (String user), имя интересующей пользователя таблицы БД (String table_name), перечень интересующих столбцов (String[] columns) – необязательный параметр и может быть равен NULL, в этом случае будут выбраны все столбцы таблицы БД – и одномерный набор SQL-фильтров для уточнения выборки (String[] sql_filtre). SQL-фильтры – это синтаксически верные части SQL-кода, которые будут подставлены под условие WHERE в запросе для выгрузки данных. Части фильтров соединяются между собой условием AND. Набор фильтров также не является обязательным параметром и может принимать значение NULL. Выходным параметром является двухмерный список данных таблицы БД.

Методы сервиса могут быть расширены или изменены. При изменении методов все зарегистрированные пользователи БД будут заранее уведомлены об изменениях и их структуре.

Работа с web-сервисами осуществляется посредством стандартного описания WSDL. Пример файла WSDL для описанного выше сервиса приведен в приложении.

Для работы с web-сервисом на стороне пользователя должен существовать клиент, работающий по протоколу SOAP, который получает доступ к описанному выше WSDL-файлу и на основе него строит соответствующие клиентские методы вызова и выгружает данные из БД.

2.8. Отправка сообщений

Сообщения обычно передаются по сети (и/или Интернет) по протоколу HTTP, почти так же, как и страницы передаются с сервера на ваш браузер. HTTP используется не всегда (его главный конкурент – SMTP, однако он далеко позади). Используемый web-сервисом протокол определен в файле WSDL.

Обычно в файле WSDL протокол, используемый для передачи сообщения SOAP, определен как HTTP. Клиент SOAP отправляет сообщения в соответствии с указанным протоколом.

2.9. Universal Description, Discovery and Integration

Universal Description, Discovery and Integration (UDDI) – это стандарт для создания каталога web-сервисов, поставляемых любым количеством программ. Это нечто вроде телефонной книги поставщиков web-сервисов. Клиенты могут искать необходимую им информацию в реестре UDDI, а реестр возвращает им необходимые данные для подключения к сервису.

Хотя UDDI довольно важный стандарт для определения web-сервисов, его значительность сильно уменьшена за счет того, что он является необязательным элементом web-сервисов, а когда есть выбор, использовать или нет, многие решают его не использовать.

Большинство организованных корпоративных сред с большим количеством внутренних web-сервисов имеют реестры UDDI. Замечательно, когда есть корпоративный сайт UDDI, содержащий информацию о доступных в вашей компании web-сервисах. Собирая вместе все сервисы, UDDI позволяет без проблем и незаметно менять их поставщиков. Если клиенты ищут сервисы через UDDI, то вызовы SOAP автоматически отправляются к новому поставщику. Однако этот компонент необязателен в архитектуре web-сервисов.

2.10. Web-сервисы и SOA

SOA представляет собой общую архитектуру интеграции приложений.

Web-сервисы представляют собой конкретный набор стандартов и спецификаций, являющихся одним из методов реализации SOA.

Между web-сервисами и SOA существует много логических связей, которые предполагают, что они могут дополнять друг друга.

Web-сервисы представляют собой модель на основе открытых стандартов, которая может быть прочитана автоматически и которая позволяет создавать формальные описания интерфейсов для web-сервисов.

Web-сервисы предоставляют механизмы коммуникации, которые обеспечивают прозрачность местоположения и возможность взаимодействия.

Web-сервисы развиваются, появляются такие технологии, как Business Process Execution Language for Web Services (WS-BPEL), SOAP в стиле документов, Web Services Definition Language (WSDL), а также WS-ResourceFramework, которые обеспечивают гибкую техническую реализацию хорошо спроектированных сервисов, включающих в себя и моделирующих многократно используемую функцию.

Работая вместе, web-сервисы и SOA имеют потенциал для решения многих технических проблем, связанных с попыткой создать среду, работающую «по требованию».

2.11. Свойства Web-сервисов

Web-сервисы являются самостоятельными. С клиентской стороны не требуется никакого дополнительного программного обеспечения. Для начала работы достаточно иметь язык программирования с поддержкой XML и HTTP. На серверной стороне требуются только HTTP-сервер и SOAP-сервер. Можно использовать web-сервисы в приложении, не написав ни единой строчки кода.

Web-сервисы описывают сами себя. Определение формата сообщения передается вместе с сообщением. Не требуется никаких внешних хранилищ метаданных или средств генерации кода.

Web-сервисы могут публиковаться, обнаруживаться и вызываться через Интернет. Эта технология использует простые установившиеся стандарты, такие как HTTP. Используется существующая инфраструктура. Необходимы также некоторые дополнительные стандарты, такие как SOAP, WSDL и UDDI.

Web-сервисы являются модульными. Простые web-сервисы могут объединяться в более сложные либо при помощи технологий рабочих процессов (workflows), либо путем вызова низкоуровневых web-сервисов из реализации

высокоуровневых web-сервисов. Web-сервисы могут образовывать цепочки, выполняющие более высокоуровневые бизнес-функции. Это уменьшает время разработки и позволяет создавать лучшие в своем роде реализации.

Web-сервисы не зависят от языка программирования и способны взаимодействовать. Клиент и сервер могут быть реализованы в разных средах. Не нужно изменять существующий код, чтобы использовать web-сервис. По существу, вы можете использовать для реализации клиентов и серверов web-сервисов любой язык.

Web-сервисы по сути своей открыты и основаны на стандартах. Главной технической основой web-сервисов являются XML и HTTP. Существенная часть технологии web-сервисов создана с использованием проектов с открытым исходным кодом. Следовательно, независимость от производителя и возможность взаимодействия – вполне реалистичные цели.

Web-сервисы имеют слабые связи. Традиционно дизайн приложения зависит от жестких связей. Для web-сервисов требуется более простой уровень координации, который позволяет осуществлять более гибкую реконфигурацию для обеспечения интеграции нужных сервисов.

Web-сервисы являются динамическими. Динамичный электронный бизнес при использовании web-сервисов становится реальностью, поскольку с UDDI и WSDL описание web-сервисов и их обнаружение можно автоматизировать. Кроме того, web-сервисы можно реализовывать и распространять, не влияя на работу клиентов, которые их используют.

Web-сервисы обеспечивают программный доступ. Данный подход не предоставляет графического пользовательского интерфейса. Он работает на уровне кода. Потребители сервисов должны знать интерфейс web-сервисов, но не должны знать детали его реализации.

Web-сервисы могут включать в себя существующие приложения. Уже существующее самостоятельное приложение может быть интегрировано в сервис-ориентированную архитектуру путем реализации web-сервисов в качестве интерфейса этого приложения.

3. СЕРВИСНАЯ ШИНА ESB

Современные приложения редко работают изолированно, приложение не может сделать что-либо значимое без взаимодействия с другими приложениями. Сервис-ориентированная архитектура интегрирует приложения для совместной работы и ускоряет их работу, разбивая приложение на части, которые могут быть объединены друг с другом.

Модель SOA может показаться простой, но возникают две важные проблемы:

1. Как потребителю найти провайдера сервиса, который он хочет вызвать?
2. Как клиент может быстро и надежно вызвать сервис в медленной и ненадежной сети?

Оказывается, существует *прямое решение обеих проблем* – подход, называемый Enterprise Service Bus (сервисная шина предприятия). *ESB упрощает вызов сервисов как для потребителя, так и для поставщика*, управляя всеми сложными взаимодействиями между ними. ESB не только упрощает вызов сервиса приложениями (или их частями), но и помогает им передавать данные и распространять уведомления о событиях. Дизайн ESB реализует множество признанных шаблонов проектирования и спецификаций стандартов.

3.1. Что такое «сервисная шина предприятия»

SOA требуется свой собственный механизм, а именно шина ESB. Она соединяет в себе несколько технологий и позволяет SOA-системам многократно использовать бизнес-сервисы и менять процессы и взаимоотношения между приложениями за счет конфигурирования (вместо перепрограммирования).

Основными функциями шины ESB являются: обмен сообщениями, преобразование данных, маршрутизация, поддержка web-сервисов и протоколов, а также «оркестровка», или согласование сервисов.

Чтобы понять, чем именно отличаются ESB-продукты, лучше всего начать с их предшественников в сфере связующего ПО, а именно с систем EAI.

EAI (Enterprise Application Integration) – объединение программных пакетов предприятия. В эпоху, предшествующую web-сервисам, программное обеспечение EAI было альтернативой заказным коммуникациям «точка – точка» и процедурам доступа к информации, из-за которых организации часто увязали в запутанных клубках программного кода. От этой проблемы страдают сайты, пытающиеся интегрировать массу разрозненных внутренних приложений, как унаследованных, так и входящих в корпоративные системы. На рис. 3.1 показа-

но EAI со специализированным центральным узлом обработки, или концентратором, выполняющим функции преобразования данных и обеспечения безопасности.

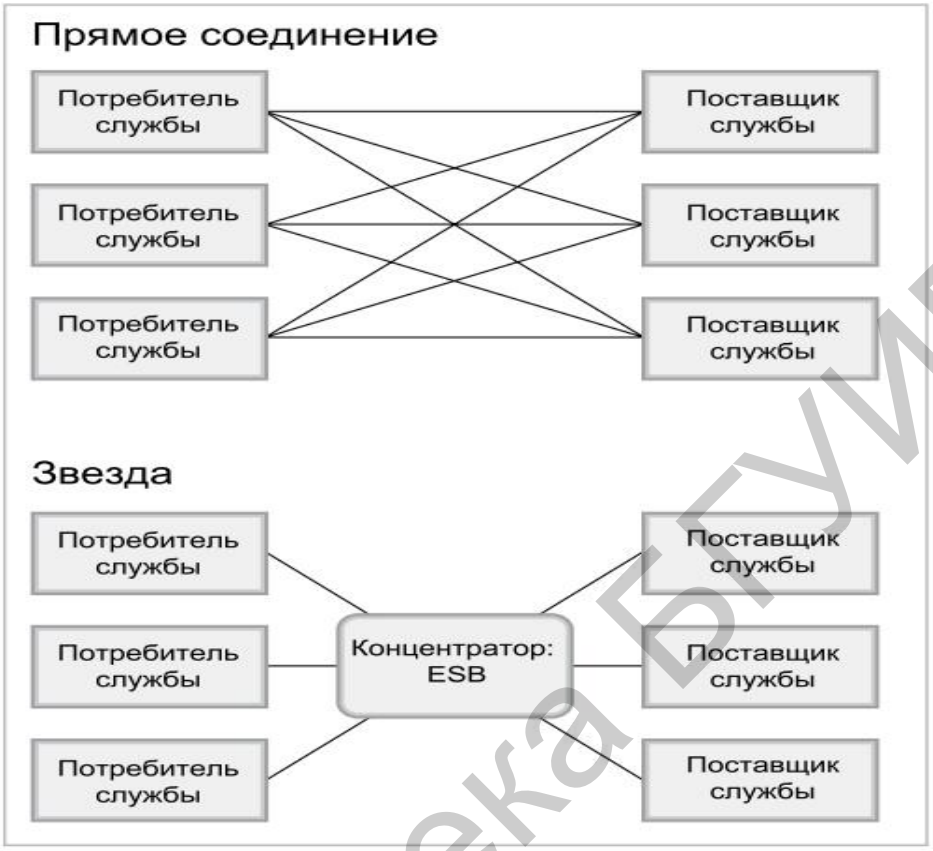

Рис. 3.1. Коммуникации «точка – точка», «звездообразная» модель

Концентратор соединен с приложениями через специальные адаптеры, благодаря чему приложения могут участвовать в коммуникациях, принимая и передавая данные, адресуемые им через концентратор другими приложениями.

Хотя средства EAI так и не справились с запутанными программами и не вытеснили конкурентов «точка – точка», зато натолкнули на идею интеграции приложений, не нуждающихся в постоянных доработках для каждого конкретного случая. Приложения поднялись на более высокий уровень абстракции, где разработчики смогли сосредоточить свои усилия на моделировании и разработке бизнес-функций.

Концепция EAI дала начало архитектуре на базе моделей. Модели в таких системах способны работать на любой платформе. Звездообразная модель обмена данными на базе концентратора означает также, что таким приложениям для совместной работы не обязательно поддерживать единую среду протоколов или других средств коммуникации. Для демонстрации развития технологий разработчики ESB часто приводят рисунки (рис. 3.2).

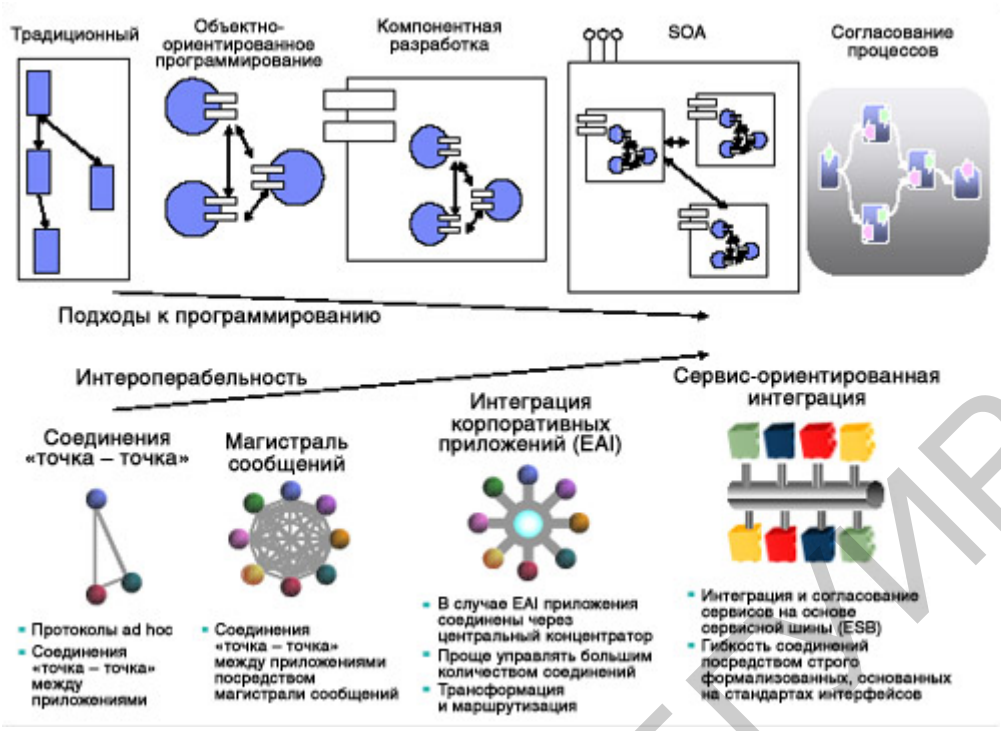

Рис. 3.2. Преемственность технологий

3.2. Вызов сервисов

Web-сервисы во многом похожи на функцию в процедурном программировании: они имеют имя, параметры и результат.

Именем является URI (Uniform Resource Identifier – унифицированный идентификатор ресурса), который *провайдер* web-сервисов использует для того, чтобы *web-сервис стал доступен как оконечная точка*. Клиент использует URI оконечной точки в качестве адреса для нахождения и вызова web-сервиса.

В *запросе* присутствуют конкретное действие и параметры, которые клиент передает web-сервису при вызове оконечной точки. После выполнения сервиса оконечная точка передает *ответ* обратно потребителю, в котором сообщается об успешном выполнении (или об ошибке) и содержится результат работы сервиса. То есть, клиент вызывает оконечную точку провайдера, передает запрос и получает назад ответ.

Клиент вызывает сервис, используя передаваемый по HTTP-сокету в HTTP-запросе SOAP-запрос. Клиент синхронно блокирует работу по HTTP-сокету, ожидая HTTP-ответ, содержащий SOAP-ответ. WSDL-контракт между потребителем и провайдером сервиса описан в API оконечной точки.

Использование ESB при реализации SOA имеет ряд преимуществ. В SOA службы должны, по определению, иметь возможности многократного исполь-

зования разными потребителями, что позволяет выиграть от уменьшения числа соединений. Кроме этого, ESB поддерживает:

– большие объемы отельных взаимодействий;
– более определенные стили интеграции, например, интеграцию, ориентированную на сообщения или на события, что расширяет возможности SOA. Шина ESB должна позволять приложениям работать с SOA либо напрямую, либо с использованием адаптеров;
– централизацию обеспечения качества обслуживания на уровне предприятия, а также возможности, связанные с централизованным управлением.

На рис. 3.3. показано высокоуровневое представление ESB.

Рис. 3.3. Корпоративная сервисная шина

3.3. Синхронный и асинхронный вызовы сервисов

Клиент может вызвать сервис синхронно либо асинхронно. С точки зрения клиента различие заключается в следующем:

– при вызове сервиса синхронно клиент использует один поток для вызова сервиса; поток передает запрос, блокируется на время выполнения сервиса и ждет ответ;
– при вызове сервиса асинхронно клиент использует два потока для вызова сервиса: один – для передачи запроса, второй – для приема ответа.

Понятия *асинхронный* и *синхронный* часто путают с понятиями *последовательный* и *параллельный*. Последние понятия относятся к порядку выполнения различных задач, в то время как *синхронный* и *асинхронный* имеют дело со способом выполнения потоком одной задачи, такой как вызов одного сервиса.

Хорошим способом понять различие между синхронным и асинхронным вызовом является рассмотрение последствий восстановления после сбоя:

- при синхронном вызове если у клиента возникает аварийная ситуация во время блокирования при работе сервиса, нельзя повторно подключиться к этому сервису после перезапуска, поэтому ответ теряется. Клиент должен повторить запрос и надеяться на отсутствие аварийной ситуации;
- при асинхронном вызове если у потребителя возникает аварийная ситуация во время ожидания ответа на запрос, то после перезапуска клиент может продолжать ожидать ответ, т. е. ответ не теряется.

Восстановление после сбоя – это не единственное различие между синхронным и асинхронным вызовами, но благодаря этому различию можно определить стиль конкретного используемого вызова.

Определив варианты взаимодействия при вызове сервиса, рассмотрим варианты соединения клиента с провайдером. Клиент может выбрать следующие варианты подключения:

- синхронный прямой вызов;
- синхронный вызов через посредника (брокера);
- асинхронный вызов через посредника (брокера).

3.3.1. Синхронный прямой вызов

Синхронным прямым вызовом называется метод вызова web-сервиса, аналогичный вызову функции: клиент знает адрес оконечной точки и вызывает ее напрямую. Для успешного вызова:

- web-сервис должен функционировать при вызове оконечной точки клиентом;
- web-сервис должен дать ответ до истечения времени ожидания.

Если web-сервис разворачивается в новом месте (например, другой интернет-домен), клиент должен быть уведомлен о новом URI оконечной точки. При развертывании нескольких провайдеров одного типа оконечная точка каждого провайдера должна иметь различный URI. Для выбора конкретного провайдера сервиса клиент должен знать каждый такой URI.

Например, рассмотрим простой web-сервис получения котировок акций: клиент передает символ акции и получает назад ее текущую стоимость. Такой сервис может предоставляться разными компаниями-брокерами, каждая из которых будет иметь свой URI. Поиск URI web-сервиса – это проблема «курицы и яйца». Если бы клиент знал местонахождение оконечной точки, он мог бы за-

просить адрес сервиса. Получается, чтобы запросить адрес сервера, нужно знать адрес оконечной точки.

Для решения этой проблемы существует спецификация UDDI (Universal Description, Discovery, and Integration), которая является каталогом (аналогичным телефонной книге) для поиска других web-сервисов. Идея заключается в развертывании UDDI-сервиса по известному клиенту адресу; затем клиент может использовать UDDI для поиска других web-сервисов.

В ситуации с сервисом котировок акций клиент знает адрес UDDI-сервиса, который, в свою очередь, знает адреса сервисов котировок акций (рис. 3.4).

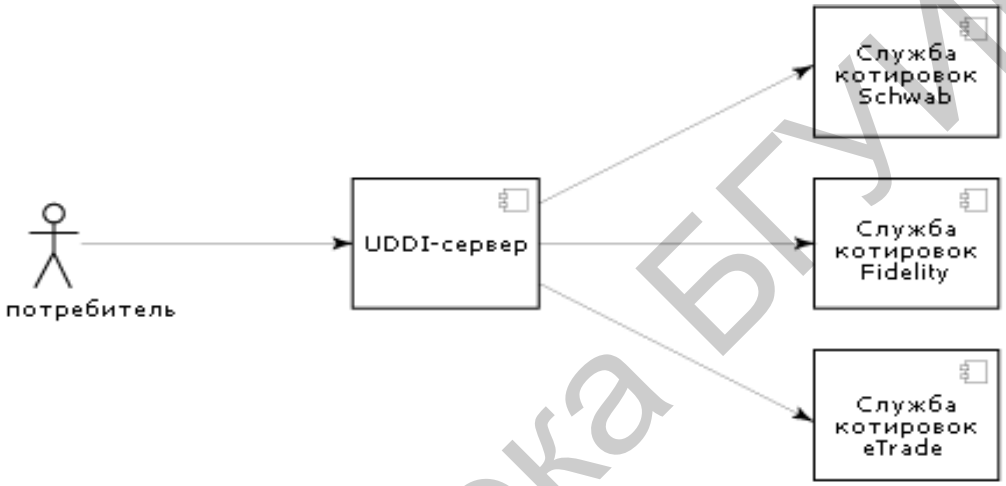

Рис. 3.4. Синхронный прямой вызов web-сервиса

На рис. 3.4 показано, как клиент использует UDDI-сервис для поиска оконечной точки провайдера сервисов котировок акций и для вызова одной из них.

Процесс вызова сервиса происходит по следующему алгоритму:

– клиент запрашивает у UDDI список провайдеров сервисов;
– клиент выбирает оконечную точку провайдера из списка, полученного из UDDI;
– клиент вызывает эту оконечную точку.

Отметим, что алгоритм выбора провайдера полностью зависит от клиента. В данном примере клиент просто выбирает первый по списку. В реальной ситуации этот выбор может быть более сложным.

Отметим также и то обстоятельство, что поскольку оконечная точка сервиса может меняться, каждый раз, когда клиент хочет вызвать сервис, он должен повторно запрашивать UDDI и анализировать, изменилась ли информация о провайдере. Необходимость запроса UDDI при каждом вызове сервиса вносит

дополнительные накладные расходы, особенно в обычной ситуации, когда информация о провайдере не меняется.

3.3.2. Синхронный вызов через посредника

Недостатком прямого вызова является то, что для вызова сервиса клиент должен знать URI оконечной точки провайдера. Он использует UDDI как каталог для поиска этого URI.

Если имеется несколько провайдеров, UDDI содержит несколько URI, и клиент должен выбрать один из них. Если провайдер меняет URI оконечной точки, он должен повторно зарегистрироваться на сервере UDDI, для того чтобы UDDI хранил новый URI. Клиент должен повторно запросить UDDI для получения нового URI.

В сущности, это означает, что каждый раз, когда клиент хочет вызвать сервис, он должен запросить в UDDI URI оконечных точек и выбрать один из них. Это ведет к затратам при периодических операциях запроса в UDDI и выбора провайдера. Этот подход также вынуждает клиента выбирать провайдера каким-либо способом, по всей видимости, из эквивалентного списка.

Одним из способов упрощения проблемы является введение *брокера*, работающего как промежуточное звено при вызове web-сервиса.

Клиент больше не вызывает сервис провайдера напрямую, а вызывает прокси-сервис брокера, который, в свою очередь, вызывает сервис провайдера. Клиент должен знать URI оконечной точки прокси-сервиса и поэтому использует UDDI для поиска адреса, но, в данном случае, UDDI возвращает только один URI, и клиент не должен делать выбор. Клиент может даже и не знать, что оконечная точка является прокси-сервисом; он знает только о том, что может использовать этот URI для вызова web-сервиса. Брокер связывает клиента с провайдерами сервисов (рис. 3.5).

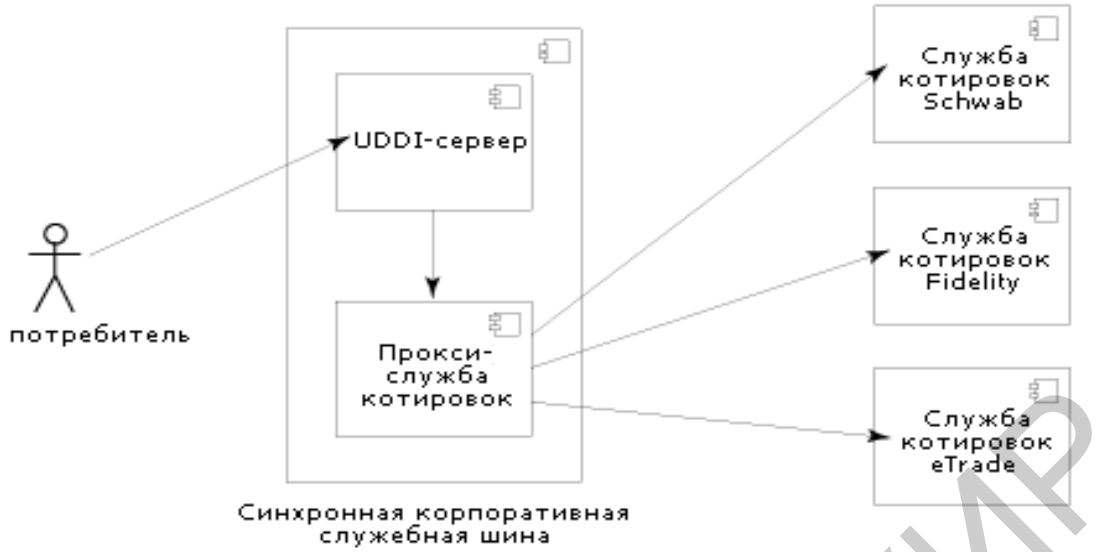

Рис. 3.5. Синхронный вызов через посредника

URI прокси-сервиса должен быть статическим. После использования UDDI для получения URI прокси-сервиса при первом вызове сервиса клиент может повторно использовать этот URI для последовательных вызовов (по крайней мере, пока прокси-сервис не прекратил работу). Тем временем прокси-сервис отслеживает провайдеров, а именно их URI (которые могут меняться между вызовами), их доступность (закончился ли неудачно последний вызов), их загрузку (как долго происходил последний вызов) и т. д.

Прокси-сервис может взять на себя выбор наилучшего провайдера для каждого вызова, освобождая от этого клиента. Клиент просто вызывает каждый раз один и тот же прокси-сервис, который координируется с различными провайдерами.

Схема использования брокера клиентом при вызове сервиса следующая:

- клиент запрашивает в UDDI список провайдеров сервиса. Возвращенный из UDDI URI фактически является прокси-сервисом. UDDI вернет только один URI, а не несколько, поскольку брокер имеет только один прокси-сервис для каждой конкретного сервиса;
- клиент вызывает сервис, используя URI прокси-сервиса;
- прокси-сервис выбирает провайдера сервиса из списка доступных провайдеров;
- прокси-сервис вызывает оконечную точку выбранного провайдера.

Забота о выборе провайдера ушла от клиента к прокси-сервису брокера. Это упрощает жизнь клиента. В конце концов, процесс выбора в прокси-сервисе может не отличаться от процесса, использовавшегося клиентом. Однако, поскольку UDDI-сервер и прокси-сервис инкапсулированы в брокере, могут

быть легко реализованы определенные функциональные возможности, такие как кэширование UDDI-информации и получение уведомлений из UDDI-сервера для прокси-сервиса о неактуальности находящейся в кэше информации.

Отметим также то, что поскольку адрес прокси является статическим, клиент не должен постоянно запрашивать UDDI при каждом вызове сервиса. То есть клиент должен только один раз запросить UDDI, затем сохранить в кэше адрес прокси-сервиса и использовать его при повторных вызовах сервиса. Это значительно снижает накладные расходы при вызове сервисов.

3.3.3. Асинхронный вызов через посредника

Недостатком синхронного подхода является то, что клиент должен ждать окончания выполнения сервиса, следовательно, поток должен быть заблокирован на время работы сервиса. Если сервис выполняется длительное время, клиент может прекратить ожидание ответа.

Когда клиент выполняет запрос (и если нет функционирующих провайдеров сервиса или они перегружены), он не может ждать. Как упоминалось выше, если у клиента возникает аварийная ситуация во время блокировки работы, даже после перезапуска ответ будет потерян и вызов нужно будет повторить.

Общим решением этой проблемы является вызов сервиса в асинхронном режиме. При таком подходе клиент использует один поток для передачи запроса и второй для получения ответа. То есть клиент не должен блокировать работу при ожидании ответа и может в это время выполнять другую работу, следовательно, клиент намного менее чувствителен к продолжительности работы сервиса.

Брокер, предоставляющий возможность клиенту вызывать web-сервис асинхронно, реализуется при помощи системы обмена сообщениями, которая использует очереди сообщений для передачи запроса и получения ответа.

Аналогично синхронному методу, для прокси-сервиса пара очередей сообщений выступает как один адрес, независимо от количества возможных прослушиваемых провайдеров (рис. 3.6).

Этот подход использует шаблон «запрос – ответ» для вызова web-сервиса. Транспортные функции могут теперь выполнять очереди сообщений. SOAP-запрос и ответ являются такими же, как и описанные выше, но они заключены в сообщения системы обмена сообщениями.

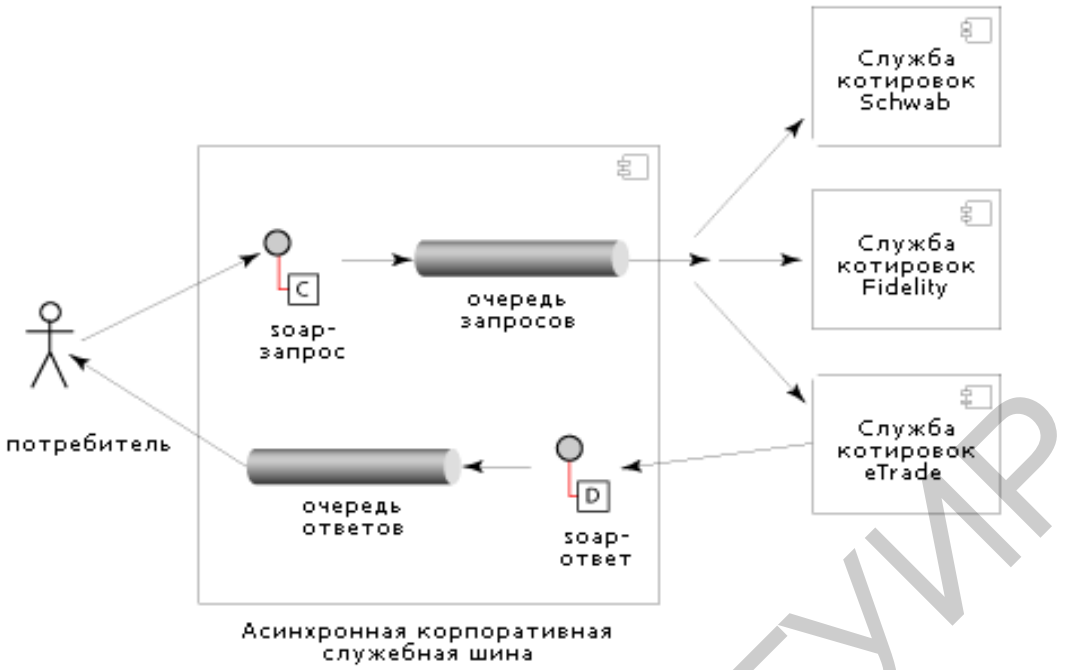

Рис. 3.6. Асинхронный вызов через посредника

Клиент асинхронно вызывает сервис при помощи брокера, действуя по следующему алгоритму:

- клиент передает SOAP-запрос в виде сообщения в очередь запросов. Клиент заканчивает работу и может использовать данный поток для выполнения других задач;
- каждый провайдер является потребителем очереди запросов, что делает их конкурирующими потребителями. Система обмена сообщениями определяет, какой провайдер может получить сообщение, и гарантирует его получение только одним провайдером. Как это работает на самом деле, зависит от реализации системы обмена сообщениями;
- победивший провайдер получает сообщение из очереди запросов;
- провайдер выполняет сервис;
- провайдер передает SOAP-ответ в виде сообщения в очередь ответов. Теперь провайдер заканчивает свою работу и может использовать свой поток для других задач (например, для ожидания следующего запроса);
- прослушивающий поток клиента получает сообщение, содержащее SOAP-ответ.

Забота о выборе провайдера ушла от клиента, т. к. этот выбор реализуется в прокси-сервисе брокера. Это упрощает жизнь потребителю. В конце концов, процесс выбора в прокси-сервисе может не отличаться от процесса, использовавшегося потребителем. Однако, *поскольку UDDI-сервер и прокси-сервис инкапсулированы в брокере*, могут быть легко реализованы определенные функ-

циональные возможности, такие как кэширование UDDI-информации и получение уведомлений из UDDI-сервера для прокси-сервиса о неактуальности находящейся в кэше информации.

3.4. Другие свойства ESB

ESB – это не только транспортный уровень. Шина должна обеспечить поддержку посреднических функций, упрощающих взаимодействие сервисов (например, функций поиска сервисов, имеющих необходимые для выполнения требований клиента возможности, или функций, устраняющих несоответствия интерфейсов клиентов и поставщиков, которые по своим возможностям являются совместимыми). Шина также должна поддерживать различные способы подключения и отключения, например, адаптеры для существующих приложений или поддержка бизнес-соединений, позволяющих работать с внешними партнерами при взаимодействии по сценарию «бизнес – бизнес».

3.4.1. Интеграционные возможности

ESB предоставляет также возможность выйти за рамки вызова сервиса и интегрировать приложения и части SOA, используя другие технологии. Вызов сервиса почти всегда является двунаправленной операцией, т. е. запрос передается от потребителя к провайдеру, а ответ посылается в обратном направлении. Другие интеграционные технологии работают как однонаправленные операции, когда отправитель передает информацию адресату и не ждет ответа; адресат просто получает информацию без необходимости ответа.

3.4.2. Передача данных

Иногда приложению нужно просто передать данные другому приложению без необходимости вызова процедуры получателя и определенно без ожидания результата. Отправителю не нужно указывать получателю, что делать с данными, он просто делает эти данные доступными.

Использование ESB для передачи данных увеличивает способности найти получателя и надежно передать данные.

Отправитель не обязан знать, как найти получателя, он только должен знать, как найти ESB и доверить ей поиск получателя. ESB также отвечает за надежную передачу данных. Отправитель может просто направить данные в ESB и быть уверенным, что они будут переданы.

3.4.3. Уведомление о событиях

Иногда об изменении, произошедшем в одном приложении, необходимо известить другие приложения. Например, если потребитель изменяет свой адрес в одном приложении, остальные приложения со своей собственной базой данных нужно известить об этом, для того чтобы они смогли исправить свои записи.

Для решения поставленной задачи одно приложение может вызвать сервис в другом приложении, еще раз для информирования его об изменении, но при таком подходе существуют три проблемы:

- во-первых, вызов сервиса является слишком специфичным в плане указания того, что делать получателю с информацией;
- во-вторых, вызов стремится быть двунаправленным, заставляя отправителя ждать ответа (даже в асинхронном режиме), чего на самом деле он не хочет делать;
- третьей и самой важной проблемой при вызове сервиса для уведомления о событии является то, что вызов сервиса по своей сути является процессом «один к одному», «потребитель – провайдер», в то время как уведомление о событии по сути является процессом «один ко многим», широковещательным, предназначается для всех заинтересованных получателей. Используя вызов сервиса, отправитель должен был бы отслеживать всех заинтересованных получателей и вызывать сервис для каждого из них по отдельности. Такую широковещательную передачу уведомления лучше оставить брокеру, находящемуся между отправителем и получателями.

Использование ESB для уведомления о событиях увеличивает ее способности хранить список заинтересованных получателей и гарантировать доставку уведомления каждому из них. Она дает возможность получателю отправить уведомление только один раз и быть уверенным в его доставке всем заинтересованным получателям, кем бы они ни были. Поскольку эта операция является однонаправленной, отправитель может заняться другой работой во время доставки уведомлений, и уведомления могут доставляться параллельно.

3.4.4. Преимущества применения web-сервисов

Отличием web-сервисов от других подходов к интеграции является то, что клиент может динамически связываться с провайдером сервиса. Это обеспечивается следующими двумя главными функциональными возможностями:

– *самоописание*. Web-сервис описывает себя удобным для машинного чтения способом. Два или более провайдера одного и того же сервиса сразу становятся распознаваемыми, даже если они реализованы абсолютно по-разному, поскольку их описательные интерфейсы соответствуют одному и тому же описанию;
– *обнаруживаемость*. Провайдеры web-сервиса могут быть организованы в автоматически поддерживаемых каталогах. Клиент может просматривать такой каталог для поиска провайдера нужного сервиса.

Эти функциональные возможности web-сервисов очень отличаются от существовавших подходов к интеграции. В них интерфейсы определялись во время компиляции и во время связи потребителя с провайдерами. Форматы сообщений не выражались описательно, они были основаны на внутренней договоренности и не могли заставить следовать этому формату, что приводило к тому, что получатель становился неспособным обработать структуру, созданную отправителем.

Возможность самоописания web-сервиса упростила интеграцию путем объявления интерфейсов, которым нужно было следовать. Динамическое обнаружение сервиса освободило клиента от зависимости от конкретного провайдера с определенным адресом, но эта возможность создала и свои собственные проблемы. Должен ли потребитель выполнять поиск провайдеров сервиса один раз или постоянно?

В ESB применяется третий, компромиссный вариант – разрешить клиенту один раз динамически связаться с прокси-сервисом и все еще иметь возможность использовать нескольких существующих провайдеров и провайдеров, которые могут появиться в будущем, через этот прокси-сервис.

Таким образом, шина ESB не только делает сервисы доступными для вызова, но также предлагает клиенту возможность найти сервисы программным способом.

3.4.5. Шлюз сервисов

Фундаментом синхронной ESB является так называемый шлюз сервисов, который выступает посредником между клиентами сервисов и провайдерами. Шлюз сервисов содействует обработке синхронных вызовов с использованием брокера. Он предоставляет доступ ко всем известным сервисам и прокси-сервисам. Таким образом, *шлюз предоставляет «единое окно» для клиента*, который хочет вызывать любой сервис у любого провайдера, известного шлюзу.

Если сервисы, которые координирует шлюз, являются web-сервисами, то они обладают способностью к самоописанию.

Каждый сервис объявляет свой интерфейс при помощи WSDL, который состоит из следующих четырех частей:

- *типы портов* – набор операций, выполняемых web-сервисом. Типом порта может быть xServices с портами/операциями, например, getX;
- *сообщения* – формат запросов и ответов, например, getXУ (который содержит символ акции) и getXR (который содержит цену);
- *типы* – типы данных, используемых web-сервисом, например, sum и pr (или просто xs:string и xs:decimal);
- *связи* – адрес для вызова операции, например, *http://*.

Такие web-сервисы шлюза (или, более конкретно, их прокси-сервисы) являются также обнаруживаемыми. Шлюз предоставляет эту возможность в виде UDDI-сервиса, как было рассмотрено ранее. Для нахождения адреса сервиса клиент запрашивает в UDDI-сервисе шлюза список провайдеров нужной WSDL-операции и получает в ответе адрес прокси-сервиса шлюза для этой операции.

3.4.6. Шина сообщений

В основе асинхронной ESB лежит известный шаблон Message Bus (Шина сообщений). Шина сообщений представляет собой набор каналов сообщений (известных также как очереди или темы), обычно настроенных как пары каналов «запрос – ответ».

Каждая пара представляет сервис, который может быть вызван клиентом, использующим шину. Клиент передает сообщение в очередь запросов сервиса и затем (в асинхронном режиме) слушает очередь ответов для получения результата. Клиент знает, какой результат соответствует его конкретному запросу, поскольку результат имеет правильный корреляционный идентификатор.

Клиент, вызывающий сервис, на самом деле не знает о том, кто предоставляет сервис. Провайдеры сервисов тоже подключены к шине сообщений и прослушивают ее на наличие сообщений запросов. Если имеется несколько провайдеров сервисов, они соревнуются друг с другом за получение конкретного запроса. Система сообщений, которую реализует шина сообщений, работает как диспетчер сообщений и распределяет сообщения запросов между провайдерами сервисов, таким образом оптимизируя это распределение в зависимости от баланса нагрузки, сетевых задержек и т. д. После получения запроса провайдером сервиса он выполняет сервис и вкладывает результат в виде сообщения в обусловленную очередь ответов.

То есть, провайдер и клиент никогда прямо не знают о месторасположении друг друга, они знают только о шине сообщений и об адресе соответствующих каналов и могут взаимодействовать посредством общих каналов.

Итак, является ли ESB только шиной сообщений? Нет, шина сообщений определенно является основой асинхронной ESB, но полная ESB – это нечто большее. ESB обладает способностями, которые никогда не имели шины сообщений: вышеупомянутыми способностями самоописания и обнаруживаемости.

Недостатком традиционной технологии шины сообщений является отсутствие способности к самоописанию. С точки зрения потребителя существует множество каналов для вызова множества сервисов. Но какой из этих каналов нужен для вызова потребителем желаемого сервиса? Потребитель не может просто передать запрос в любой канал запросов, он должен знать правильный канал, использующийся для вызова конкретного сервиса. Более того, даже после определения (каким-то образом) правильного канала (и канала для прослушивания на наличие ответа), клиент должен знать формат данных, в котором нужно передать запрос (и в каком формате данных ожидать ответ).

Как было рассмотрено ранее, для синхронных web-сервисов эту проблему решает WSDL-файл, который в настоящее время является вариантом стандарта для описания также и асинхронных сервисов. WSDL, связанный с каналом запроса, должен описывать, какой сервис этот канал предоставляет, а также формат сообщения запроса, который должен предоставить потребитель. WSDL должен, вероятно, также указать канал ответов, который прослушивает клиент для получения ответа, и формат ожидаемого ответного сообщения. Таким образом, вызывающее приложение может программно проанализировать пару каналов для вызова сервиса и определить, предоставляют ли они нужный сервис, используя желаемые форматы сообщений запроса и ответа.

Каналы сервисов с самоописанием приводят к появлению другой проблемы, а именно проблемы обнаружения, которую синхронные web-сервисы решают при помощи UDDI. Как было рассмотрено ранее, потребитель запрашивает в UDDI-сервере адрес провайдера web-сервиса, и сервер возвращает URL этого провайдера. Потребитель использует этот URL для вызова сервиса.

ESB нуждается в аналогичном сервисе каталогов с UDDI-подобным API, используя который клиент мог бы запросить адрес сервиса, который реализует желаемую WSDL-операцию. ESB возвращает адрес соответствующей пары каналов запросов и ответов. То есть, ESB-клиент, аналогично UDDI-клиенту, не должен знать ничего кроме следующего:

– WSDL описывает сервис, который хочет вызвать клиент;

– адрес сервиса каталогов ESB (который может быть производным от корневого адреса ESB).

Этого достаточно для обнаружения каналов запросов и ответов сервиса и для начала вызова сервиса. Более того, сервис каталогов является просто еще одним сервисом, предоставляемым ESB, условно говоря, главным сервисом для поиска других сервисов.

4. АРХИТЕКТУРА СЕРВИСНЫХ КОМПОНЕНТОВ

4.1. Основные понятия SCA

Архитектура сервисных компонентов (Service Component Architecture, SCA) была разработана для упрощения интеграции бизнес-приложений и разработки новых служб.

Если SOA – это абстрактный способ интерпретации служб и их взаимосвязей, то SCA – это реализация архитектуры SOA. Стандарты SCA позволяют создавать службы и обеспечивать их интеграцию.

SCA отделяет бизнес-логику приложений от деталей реализации. Она предлагает модель, в которой интерфейсы, реализации и ссылки определяются не зависящим от технологии способом, и позволяет связывать эти элементы при помощи любой технологически специфической реализации.

Возможность разделять бизнес-логику и инфраструктурную логику уменьшает количество ИТ-ресурсов, необходимых для создания приложения в масштабе предприятия, и разработчики получают больше времени на решение конкретной проблемы бизнеса, не вдаваясь при этом в детали используемой технологии реализации.

Enterprise Service Bus или Integration Bus – это посреднический уровень, который надстраивается над транспортным уровнем Application Server.

Сам продукт Enterprise Service Bus (ESB) предлагает готовые посреднические функции и простые в использовании инструменты, позволяющие быстро проектировать и реализовать ESB как расширение возможностей Application Server.

ESB добавляет к возможностям сервера приложений некоторые дополнительные возможности, рассматриваемые далее. Прежде всего, это встроенные посреднические функции, которые можно использовать для создания интеграционной логики, обеспечивающей возможность взаимодействия систем.

Программная модель SCA поддерживает быструю разработку компонентов посреднических потоков. Integration Developer – это простой в использовании инструмент, поддерживающий ESB.

Используя Application Server, ESB предлагает возможности для обмена JMS-сообщениями и взаимодействие с WebSphere MQ, а также полнофункциональные клиенты для обеспечения возможности взаимодействия систем.

Предлагается поддержка J2EE Connector Architecture на основе адаптеров ESB.

Для правильной реализации SOA необходимо иметь единую модель вызовов и единую модель данных.

Архитектура сервисных компонентов (Service Component Architecture, SCA) – это и есть та модель вызовов: каждый интеграционный компонент описывается интерфейсом.

Службы могут собираться в редакторе сборки компонентов, что позволяет создать очень гибкое и инкапсулированное решение.

В ESB *модели SCA* вводится новый тип компонента: *компонент посреднического потока* (Mediation Flow Component).

С точки зрения SCA компонент посреднического потока ничем не отличается от любого другого компонента.

Бизнес-объекты (Business Objects) – это универсальный формат описания данных. Они используются в качестве объектов-данных и передаются от службы к службе на основе стандарта сервисных объектов-данных (Service Data Object, SDO).

Кроме того, предложен и специальный тип SDO – сервисный объект-сообщение (Service Message Object, SMO).

4.2. Посреднические функции, потребители и поставщики служб

Во взаимодействии служб в SOA определены и поставщики, и потребители служб.

Назначение Enterprise Service Bus – вмешиваться в поток запросов потребителей служб и выполнять дополнительные задачи, связанные с посредничеством, в целях обеспечения гибкости связей. По завершении посреднической работы нужно вызвать соответствующего поставщика службы.

К посреднической работе относятся:

- централизация логики маршрутизации, чтобы поставщика службы можно было явным образом заменить;
- выполнение таких функций, как преобразование протоколов и картирование передачи;
- выполнение функций фасада в целях создания различных интерфейсов между поставщиками и потребителями;
- дополнительная логика для таких задач, как журналирование.

Посредническая система подбирает протокол и подробные сведения о запросе, а также изменяет полученный ответ.

ESB поддерживает различные модели работы с сообщениями, давая возможность выбрать наиболее подходящую, в том числе:

- односторонние взаимодействия;
- «запрос – ответ»;
- публикация/подписка.

4.3. Элементы SCA

Архитектура SCA предоставляет собой абстракцию, которая охватывает сеансовые EJB без сохранения состояния (Stateless), web-службы, POJO, процессы WS-BPEL, доступ к базам данных, корпоративным информационным системам (Enterprise Information System, EIS) и т. п.

SCA охватывает как использование, так и разработку служб. Эта архитектура предлагает единую модель, используемую прикладными программистами и инструментарием.

SCA представляет собой универсальную модель бизнес-служб, которые публикуют бизнес-данные или работают с ними. Сервисные объекты-данные (Service Data Objects, SDO) представляют собой универсальную модель бизнес-данных.

На рис. 4.1. показаны следующие *основные элементы компонента SCA*:

– интерфейс;
– реализация;
– ссылка.

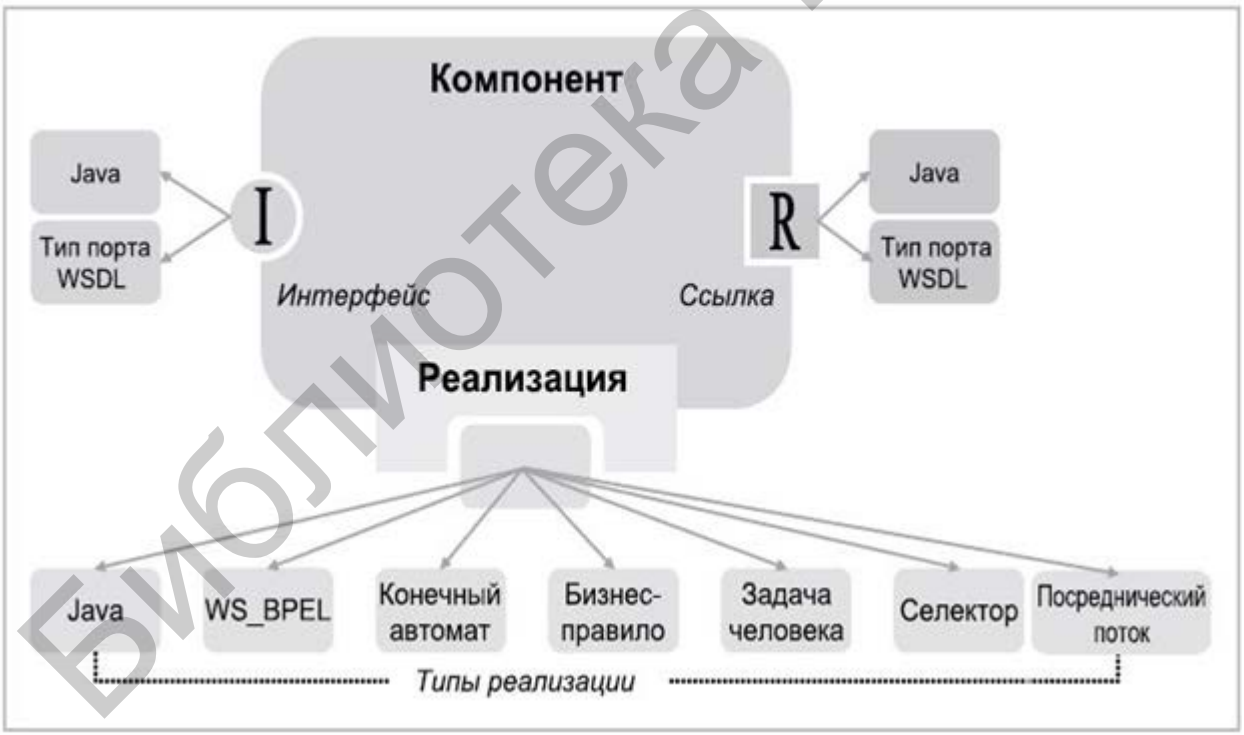

Рис. 4.1. Элементы SCA

Интерфейс службы определяется Java-интерфейсом или типом порта WSDL.

Аргументы и возвращаемые значения описываются Java-классами, простыми типами Java или XML-схемой.

Java-классы, сгенерированные из SDO, являются предпочтительной формой Java-класса, поскольку здесь есть связь с технологиями XML.

Аргументы, описанные в XML-схеме, предъявляются программистам в виде SDO.

Компонент предъявляет интерфейсы уровня бизнеса прикладной бизнес-логике, чтобы службу можно было использовать и вызывать.

В *интерфейсе компонента* определяются операции, которые могут быть вызваны, и данные, которые могут быть переданы, например аргументы, возвращаемые значения и исключения.

Для импорта и экспорта также существуют интерфейсы, позволяющие вызывать опубликованную службу.

Все компоненты имеют интерфейсы типа WSDL.

Интерфейсы типа Java поддерживаются только Java-компонентами.

Если импортируемый или экспортируемый компонент имеет несколько интерфейсов, то все интерфейсы должны относиться к одному типу.

Компонент может вызываться синхронно или асинхронно.

SCA-службы и не-SCA-службы могут использовать в своих реализациях другие сервисные компоненты. В них нет жестко запрограммированного использования других служб. Объявляются гибкие ссылки, которые называются сервисными ссылками (Service References). Сервисные ссылки разрешаются через связи служб (Wiring).

Можно использовать связи SCA для создания SCA-приложений методом сборки из компонентов. Когда компоненту нужно использовать *службы другого компонента*, он должен иметь партнерскую ссылку или просто ссылку.

Мы можем использовать внутреннюю ссылку (In-line Reference) – это означает, что определение компонента, на который ссылается другой компонент, находится в той же области действия, что и ссылающийся компонент. Иными словами, оба компонента определены в одном модуле.

Компоненты объединяются в модуль, который может быть сервисным модулем или посредническим модулем. Реализации компонентов, которые используются при сборке модуля, могут находиться внутри модуля. Компоненты, относящиеся к другим модулям, могут использоваться путем импорта.

Компоненты в разных модулях можно связать (Wiring) путем публикации служб в виде экспорта, имеющего свой собственный интерфейс, и перетаскивания экспортного модуля на нужную диаграмму сборки с созданием импорта.

При связывании компонентов также можно указывать квалификаторы качества обслуживания для реализаций партнерских ссылок и интерфейсов компонента.

Импорт позволяет использовать функции, не являющиеся частью собираемого модуля. Импортировать можно компоненты из других модулей или не-SCA-компоненты, например, сеансовые EJB без сохранения состояния и web-службы.

4.4. Посреднические модули

Посреднический модуль (Mediation Module) – это новый тип SCA-компонента, который может выполнять обработку или осуществлять посреднические функции при взаимодействии служб.

Посреднический модуль доступен для внешнего доступа *через экспорт* (Export), в котором указываются предъявляемые для доступа интерфейсы. Эти интерфейсы определены в WSDL-документе.

Самостоятельные ссылки (Stand-alone References) предоставляют интерфейс для внешнего доступа только SCA-клиентам. Они не предполагают наличия WSDL-документа. Вместо этого используется объявление интерфейса на Java (которое называется ссылкой).

Посреднический модуль, как правило, вызывает поставщиков служб. Эти поставщики объявляются путем создания импорта (Import), который представляет вызываемую внешнюю службу.

Для каждого импорта или экспорта нужно указать интерфейсы. Каждый интерфейс включает несколько операций, которые, в свою очередь, могут иметь *несколько входных* и *выходных параметров* либо в виде простых типов данных, либо в виде бизнес-объектов. Односторонняя операция имеет только входные параметры.

Для каждого импорта и экспорта должна существовать *привязка* (Binding). Привязка определяет конкретный тип запуска для потребителя или поставщика службы. Enterprise Service Bus поддерживет несколько типов привязок.

Функция (или бизнес-логика), реализованная в удаленных системах (например, в web-службах, функциях EIS, EJB или в удаленных SCA-компонентах), в модели обозначается как импортированная служба.

Импорт (компонент импорта) *имеет интерфейсы, аналогичные интерфейсам удаленной службы* (или их части), с которой он связан и через которые можно вызывать удаленную службу.

Импорты используются в приложении точно так же, как локальные компоненты. Этим создается единая модель для всех функций, независимо от их местоположения и реализации.

Связи импортов необязательно должны быть определены во время разработки. Их можно определить в момент размещения.

JMS-привязка использует протокол JMS, реализуемый в Application Server через сервисную интеграционную шину (Service Integration Bus). Выделяют:

- *привязку с web*-службами при помощи SOAP/HTTP и SOAP/JMS;
- *адаптеры WebSphere*, совместимые с JCA;
- *привязку SCA*, которая используется по умолчанию и применяется для взаимодействия между SCA-модулями.

Экспорт (компонент экспорта) – это публикуемый интерфейс компонента, через который бизнес-служба компонента связана с внешним миром, например, в виде web-службы.

Экспорт имеет интерфейсы, аналогичные интерфейсам компонента (или их части), с которым он связан, и через которые можно вызывать опубликованную службу.

Если перетащить экспорт от другого модуля на диаграмме сборки, будет автоматически создан компонент-импорт.

4.5. Компоненты посреднических потоков

В посредническом модуле может находиться один компонент посреднического потока.

Компоненты посреднических потоков предоставляют один или несколько интерфейсов и используют одну или несколько ссылок на партнера.

И то, и другое решается путем связывания с модулями экспорта или импорта при помощи связей (Wire).

4.6. Сервисные объекты-данные

Бизнес-данные, обмен которыми происходит в интегрированном приложении, в шине Enterprise Service Bus *представлены в виде бизнес-объектов*. Эти объекты основываются на новой технологии доступа к данным – сервисных объектах-данных (Service Data Objects, SDO).

Общим представлением являются сервисные объекты-сообщения (SMO).

В ESB могут обрабатываться *следующие типы сообщений*:

- объекты-данные SDO;
- штрафы данных SDO;
- *сообщения*, вызывающие SCA-компонент (запрос, ответ или исключение);
- SOAP-сообщения;

– JMS-сообщения.

Модель SMO имеет возможности для расширения, поэтому в будущем она сможет поддерживать и другие типы сообщений.

Модель SMO дополняет SDO информацией, связанной с поддержкой системы обмена сообщениями.

4.6.1. Концепции SDO

Существует несколько связанных с SDO ключевых концепций, которые позволяют понять архитектуру бизнес-объектов, в том числе принципы разработки и использования бизнес-объектов в Enterprise Service Bus.

Базовой концепцией архитектуры SDO является объект-данные (Data Object). Фактически, термин SDO часто используется как синоним термина объект-данные.

Объект-данные – это структура, в которой хранятся элементарные данные, поля с несколькими значениями или и то, и другое.

Объект-данные также содержит ссылки на метаданные, которые являются сведениями о данных, хранящихся в объекте.

В программной модели SDO объекты-данные представлены путем определения Java-интерфейса commonj.sdo.DataObject. Этот интерфейс включает в себя определения методов, которые позволяют клиентам читать и устанавливать свойства, связанные с DataObject.

Еще одной важной концепцией архитектуры SDO является граф данных (Data Graph).

Граф данных – это структура, инкапсулирующая набор объектов-данных.

От объекта-данных, находящегося в графе на верхнем уровне, можно обращаться к другим объектам-данным, переходя по ссылкам из корневого объекта-данных.

В программной модели SDO графы данных представлены путем определения Java-интерфейса commonj.sdo.DataGraph.

4.6.2. Применение SDO в SCA

И SCA, и SDO (основа бизнес-объектов) создавались как взаимодополняющие сервис-ориентированные технологии. Сам язык SCA состоит из некоторого набора готовых графических примитивов, которые интерпретируются и выполняются ПО ESB. SDO передаются от одного графического примитива SCA другому в соответствии с его интерфейсом, при необходимости определяются и выполняются необходимые преобразования данных.

Независимо от того, определен ли интерфейс, связанный с конкретным сервисным компонентом, как Java-интерфейс или как тип порта WSDL, входные и выходные параметры представляют собой бизнес-объекты.

4.7. Пример SDO

В программной модели SDO объекты-данные представлены путем определения Java-интерфейса commonj.sdo.DataObject. Этот интерфейс включает в себя определения методов, которые позволяют клиентам читать и устанавливать свойства, связанные с DataObject.

Например, рассмотрим моделирование данных о заявке на перевозку ГУ-XX из компонента АСУ XXX «Формирование портфеля заявок» при помощи объекта данных SDO.

В приведенном ниже коде показано, как следует использовать API DataObject для получения свойств объекта-данных ГУ-XX.

```
private com.ibm.websphere.bo.BOFactory boFactory = null;
private com.ibm.websphere.sca.ServiceManager serviceManager = null;

serviceManager = new com.ibm.websphere.sca.ServiceManager();
boFactory = (com.ibm.websphere.bo.BOFactory) serviceManager
                        .locateService("com/ibm/websphere/bo/BOFactory");

DataObject gU12 = boFactory.create("http://DkrKtgENG/Businessitems",
"GU12");

gU12.setInt("SPEC_COND", gu12DTO.getSpecCond());
gU12.setString("PLANNED_LOADDATE",
gu12DTO.getPlannedLoadDate());
gU12.setLong("REQ_STAUS_INT", gu12DTO.getReqStaGu12Un());
gU12.setLong("REQ_TYPE_TRAF_UN",gu12DTO.getReqTypeTraf());
gU12.setString("TRAFFIC_BY_ROUTES",gu12DTO.getTrafByRoutes());

gU12.setString("CAR_OWNER", gu12DTO.getCarOwner());
gU12.setString("REQ_STATUS", gu12DTO.getReqStatus());
gU12.setString("DEPARTAMENT_NR", gu12DTO.getDepartamentNR());
gU12.setString("REQ_TYPE", gu12DTO.getReqType());
gU12.setString("VID", gu12DTO.getVid());
gU12.setBigDecimal("SUM_TONN", gu12DTO.getTonneSum());
gU12.setBigDecimal("REQ_MONTH", gu12DTO.getReqMonth());
```

```
gU12.setBigDecimal("REQ_YEAR", gu12DTO.getReqYear());
gU12.setBigDecimal("SUM_VAGON", gu12DTO.getVagonSum());
if(gu12DTO.getStartStation()!=null){
gU12.setString("NOD", gu12DTO.getStartStation().getNOD());
}
gU12.setString("CARGO_OPER", gu12DTO.getCargoOper());
gU12.setString("BRANCH_LINE", gu12DTO.getBranchLine());
gU12.setString("BRANCH_LINE_OWNER",
gu12DTO.getBranchLineOwner());
gU12.setString("CARRIER_NAME", gu12DTO.getCarrierName());
gU12.setString("PLAN_NR", gu12DTO.getPlanNR());
gU12.setString("REG_DATE", gu12DTO.getRegDate());
gU12.setString("USER_NAME", gu12DTO.getUserName());
```

Граф данных – это структура, инкапсулирующая набор объектов-данных. От объекта-данных, находящегося в графе на верхнем уровне, можно обращаться к другим объектам-данным, переходя по ссылкам из корневого объекта-данных. В программной модели SDO графы данных представлены путем определения Java-интерфейса commonj.sdo.DataGraph,

На рис. 4.2. приведен граф данных для ГУ-ХХ.

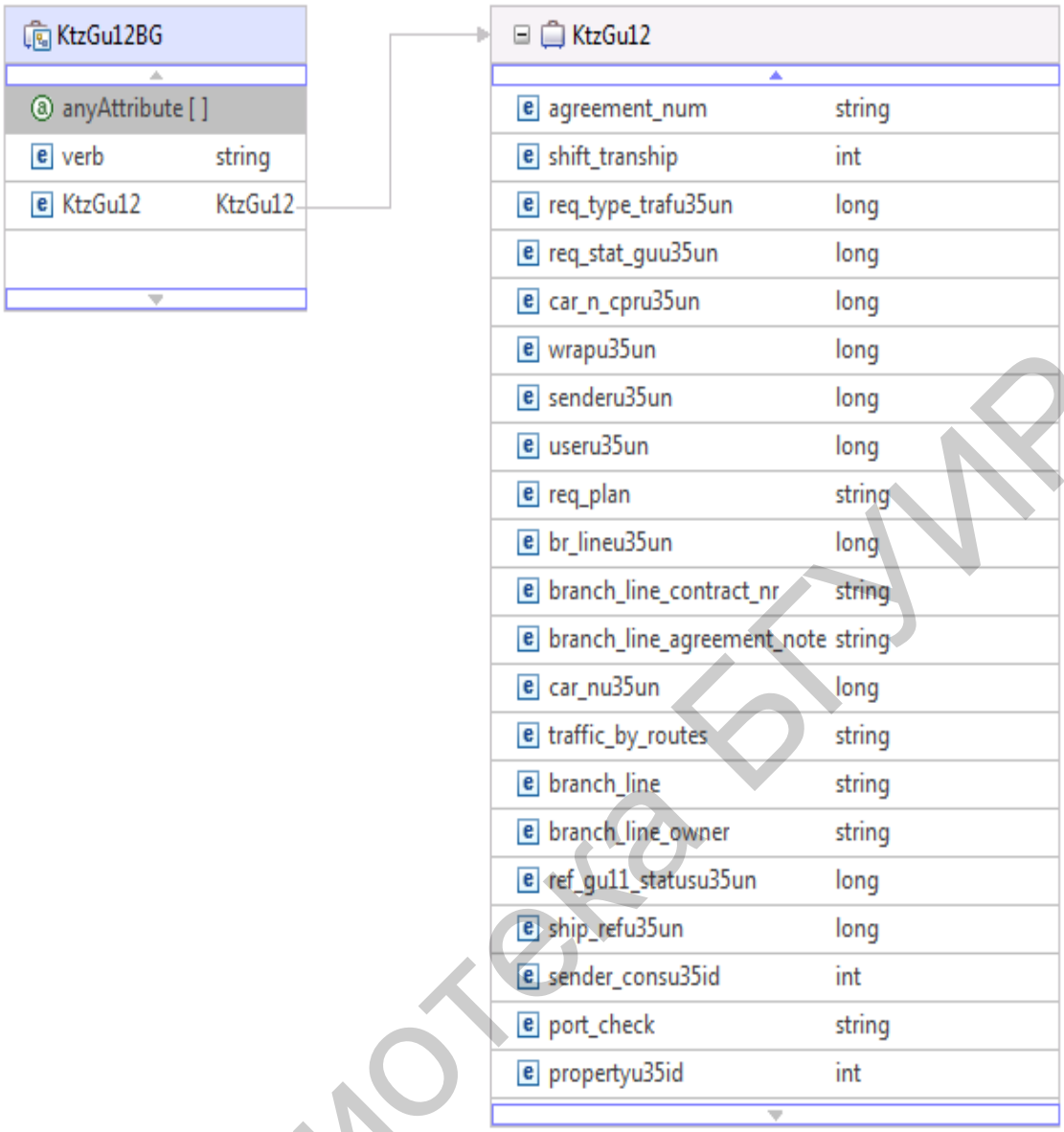

Рис. 4.2. Бизнес-граф ГУ-XX

4.8. Структура документа WSDL

WSDL (Web Services Description Language) – язык описания web-сервисов, основанный на языке XML. В документе WSDL определяется web-сервис с помощью следующих основных элементов:

– <portType> – методы, предоставляемые web-сервисом;
– <message> – сообщения, используемые web-сервисом;
– <types> – типы данных, используемые web-сервисом;
– <binding> – протоколы связи, используемые web-сервисом.

Основная структура документа WSDL выглядит следующим образом:

```
<definitions>
    <types>
        определение типов
    </types>
    <message>
        определение сообщения
    </message>
    <portType>
        определение порта
    </portType>
    <binding>
        определение связей
    </binding>
</definitions>
```

Документ WSDL может также содержать другие элементы, например, элементы расширения и элемент Service, который позволяет объединить вместе в одном отдельном документе WSDL определения нескольких web-сервисов.

Элемент <portType> является наиболее важным элементом в WSDL. Он определяет сам web-сервис, предоставляемые им операции и используемые сообщения.

Элемент <portType> можно сравнить с библиотекой функций (модулем, классом) в традиционном языке программирования.

Элемент <message> определяет элементы данных операции. Каждое сообщение может содержать одну или несколько частей. Эти части можно сравнить с параметрами вызываемых функций в традиционном языке программирования.

Элемент <types> определяет тип данных, используемых web-сервисом. Для максимальной платформонезависимости WSDL использует синтаксис XML Schema для определения типов данных.

Элемент <binding> определяет формат сообщения и детали протокола для каждого порта.

4.9. Пример разработки на ESB-шине

На рис. 4.3–4.7 приведены примеры на языке SCA: формирование сообщений из БД, прием запроса, формирование и вывод.

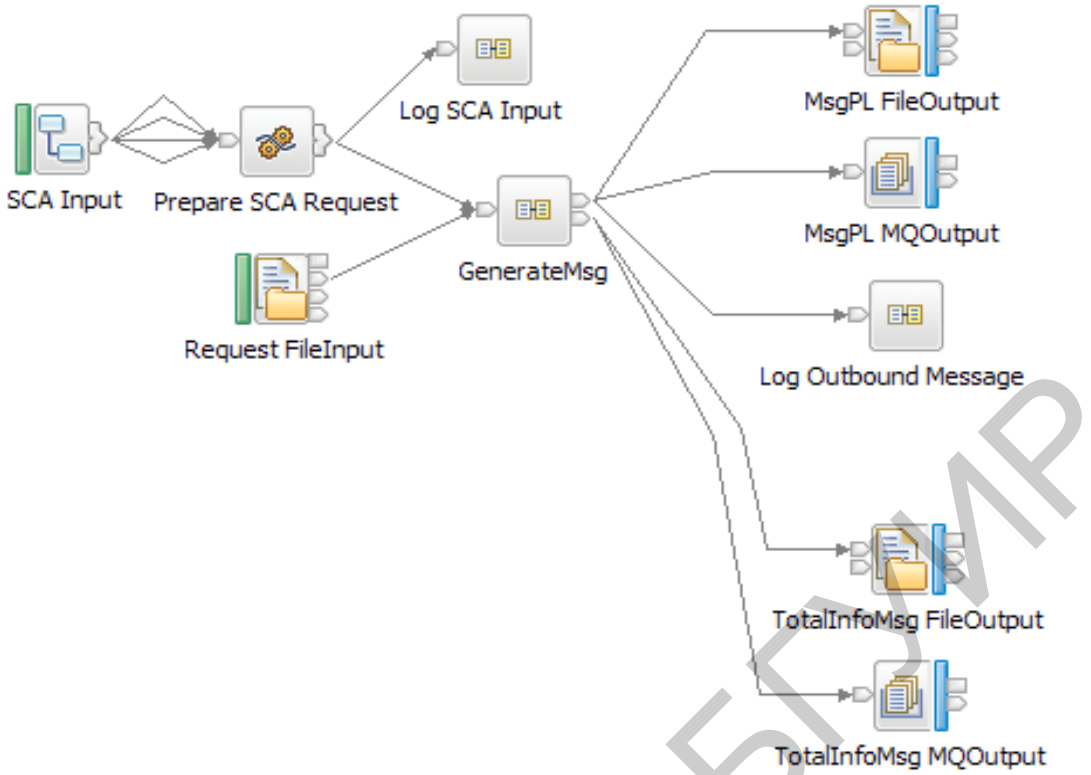

Рис. 4.3. Поток – подготовка запроса на формирование сообщения

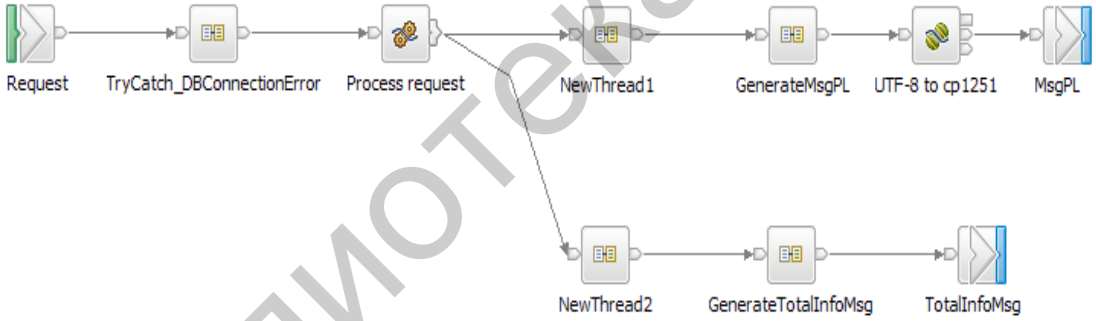

Рис. 4.4. Поток – формирование сообщения (нескольких типов)

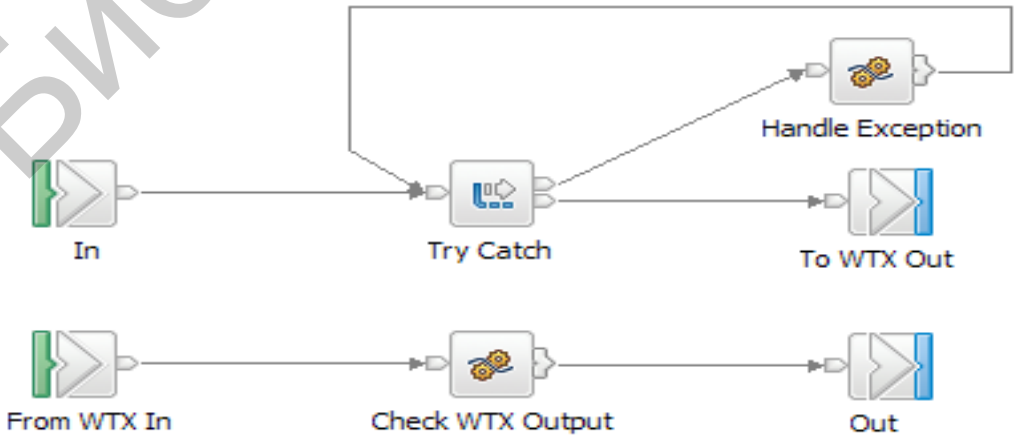

Рис. 4.5. Поток – обработка ошибочных ситуаций и повторение

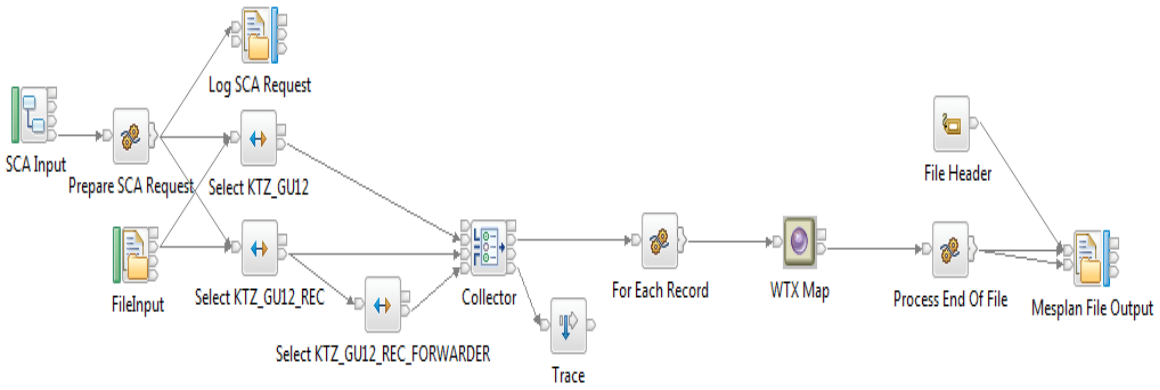

Рис. 4.6. Экспорт из базы данных АСУ XXX в АС «МС»

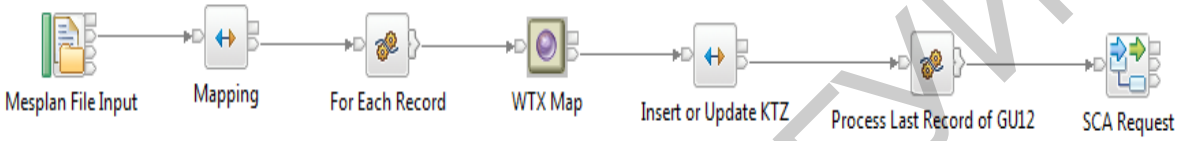

Рис. 4.7. Импорт из АС «МС» в базу данных АСУ XXX

ПРИЛОЖЕНИЕ

Пример файла WSDL для web-сервиса доступа к данным в БД

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://services.nsi"
xmlns:impl="http://services.nsi" xmlns:intf="http://services.nsi"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <wsdl:types>
 <schema elementFormDefault="qualified" targetNamespace="http://services.nsi"
xmlns="http://www.w3.org/2001/XMLSchema" xmlns:impl="http://services.nsi"
xmlns:intf="http://services.nsi" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <element name="getTableInfo">
 <complexType>
 <sequence>
 <element name="user" nillable="true" type="xsd:string"/>
 <element name="table_name" nillable="true" type="xsd:string"/>
 </sequence>
 </complexType>
 </element>
 <complexType name="TableRow">
 <sequence>
 <element maxOccurs="unbounded" name="row" nillable="true" type="xsd:string"/>
 </sequence>
 </complexType>
 <element name="getTableInfoResponse">
 <complexType>
 <sequence>
 <element maxOccurs="unbounded" minOccurs="0" name="getTableInfoReturn"
type="impl:TableRow"/>
 </sequence>
 </complexType>
 </element>
 <element name="getTableData">
 <complexType>
 <sequence>
```

<element name="user" nillable="true" type="xsd:string"/>
<element name="table_name" nillable="true" type="xsd:string"/>
<element maxOccurs="unbounded" minOccurs="0" name="columns"
type="xsd:string"/>
<element maxOccurs="unbounded" minOccurs="0" name="sql_filtre"
type="xsd:string"/>
</sequence>
</complexType>
</element>
<element name="getTableDataResponse">
<complexType>
<sequence>
<element maxOccurs="unbounded" minOccurs="0" name="getTableDataReturn"
type="impl:TableRow"/>
</sequence>
</complexType>
</element>
<element name="getAllTables">
<complexType>
<sequence>
<element name="user" nillable="true" type="xsd:string"/>
</sequence>
</complexType>
</element>
<element name="getAllTablesResponse">
<complexType>
<sequence>
<element maxOccurs="unbounded" minOccurs="0" name="getAllTablesReturn"
type="xsd:string"/>
</sequence>
</complexType>
</element>
</schema>
</wsdl:types>

<wsdl:message name="getTableDataRequest">
<wsdl:part element="impl:getTableData" name="parameters"/>
</wsdl:message>

```
 <wsdl:message name="getTableDataResponse">
 <wsdl:part element="impl:getTableDataResponse" name="parameters"/>
 </wsdl:message>

 <wsdl:message name="getTableInfoResponse">
 <wsdl:part element="impl:getTableInfoResponse" name="parameters"/>
 </wsdl:message>
 <wsdl:message name="getAllTablesResponse">
 <wsdl:part element="impl:getAllTablesResponse" name="parameters"/>
 </wsdl:message>

 <wsdl:message name="getAllTablesRequest">
 <wsdl:part element="impl:getAllTables" name="parameters"/>
 </wsdl:message>
 <wsdl:message name="getTableInfoRequest">
 <wsdl:part element="impl:getTableInfo" name="parameters"/>
</wsdl:message>

 <wsdl:portType name="LoadData">
 <wsdl:operation name="getTableInfo">
 <wsdl:input message="impl:getTableInfoRequest" name="getTableInfoRequest"/>
 <wsdl:output message="impl:getTableInfoResponse"
name="getTableInfoResponse"/>
 </wsdl:operation>
 <wsdl:operation name="getTableData">
 <wsdl:input message="impl:getTableDataRequest" name="getTableDataRequest"/>
 <wsdl:output message="impl:getTableDataResponse"
name="getTableDataResponse"/>
 </wsdl:operation>
 <wsdl:operation name="getAllTables">
 <wsdl:input message="impl:getAllTablesRequest" name="getAllTablesRequest"/>
 <wsdl:output message="impl:getAllTablesResponse"
name="getAllTablesResponse"/>
 </wsdl:operation>
 </wsdl:portType>

 <wsdl:binding name="LoadDataSoapBinding" type="impl:LoadData">
```

```
 <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
 <wsdl:operation name="getTableInfo">
 <wsdlsoap:operation soapAction=""/>
 <wsdl:input name="getTableInfoRequest">
 <wsdlsoap:body use="literal"/>
 </wsdl:input>
 <wsdl:output name="getTableInfoResponse">
 <wsdlsoap:body use="literal"/>
 </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="getTableData">
 <wsdlsoap:operation soapAction=""/>
 <wsdl:input name="getTableDataRequest">
 <wsdlsoap:body use="literal"/>
 </wsdl:input>
 <wsdl:output name="getTableDataResponse">
 <wsdlsoap:body use="literal"/>
 </wsdl:output>
 </wsdl:operation>
 <wsdl:operation name="getAllTables">
 <wsdlsoap:operation soapAction=""/>
 <wsdl:input name="getAllTablesRequest">
 <wsdlsoap:body use="literal"/>
 </wsdl:input>
 <wsdl:output name="getAllTablesResponse">
 <wsdlsoap:body use="literal"/>
 </wsdl:output>
 </wsdl:operation>
 </wsdl:binding>
 <wsdl:service name="LoadDataService">
 <wsdl:port binding="impl:LoadDataSoapBinding" name="LoadData">
 <wsdlsoap:address location="http://ip адрес сервера:порт
подключения/WEBNSI/services/LoadData"/>
 </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

Перечень принятых терминов

Сервис-ориентированная архитектура (Service-Oriented Architecture, SOA) представляет собой стиль создания архитектуры ИТ, направленный на превращение бизнеса в ряд связанных сервисов – стандартных бизнес-задач, которые можно при необходимости вызывать через сеть.

Фундамент SOA IBM (IBM SOA Foundation) – это интегрированный открытый набор программного обеспечения, лучшие методы и шаблоны, которые предоставляют все необходимое для создания SOA.

Сервисы – законченные модули программного обеспечения, каждый из которых выполняет определенную бизнес-задачу. У них есть четко определенные интерфейсы, и они не зависят от приложений и платформ, на которых они работают.

Web-сервисы (web-службы) – это технология, которая позволяет приложениям взаимодействовать друг с другом независимо от платформы, на которой они развернуты, а также от языка программирования, на котором они написаны.

Web-сервис – это программный интерфейс, который описывает набор операций, которые могут быть вызваны удаленно по сети посредством стандартизированных XML-сообщений. Для описания вызываемой операции или данных используются протоколы, базирующиеся на языке XML.

WSDL (Web Services Description Language) – язык описания web-сервисов, основанный на языке XML.

SSL (Secure Sockets Layer) – уровень защищенных сокетов. Криптографический протокол, обеспечивающий безопасную передачу данных по сети Интернет. При его использовании создается защищенное соединение между клиентом и сервером.

ESB (**Enterprise Service Bus**) – корпоративная сервисная шина. Она обеспечивает связь при значительно меньших затратах, чем у традиционных способов связи. ESB позволяет компании объединить всю внутреннюю и внешнюю инфраструктуру надежным и расширяемым способом.

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

SOAP (Simple Object Access Protocol) – этот протокол кодирует сообщения таким образом, чтобы они могли быть доставлены через сеть с использованием транспортного протокола, такого как HTTP, IIOP, SMTP и др.

UDDI (Universal Description, Discovery and Integration) – определяет реестр и соответствующий протокол для обнаружения и доступа к сервисам.

Web Services Inspection Language (язык обследования web-служб, WSIL) – это спецификация на основе XML, которая позволяет находить web-сервисы без использования UDDI.

EAI (Enterprise Application Integration) – объединение программных пакетов предприятия.

URI (Uniform Resource Identifier) – унифицированный идентификатор ресурса.

BPMN (Business Process Modeling Notation) – нотация для наглядного изображения бизнес-процессов.

BPM (Business Process Management) – представляет собой стратегию управления и повышения эффективности бизнеса посредством непрерывной оптимизации бизнес-процесса в замкнутом цикле моделирования, выполнения и мониторинга.

Список использованных источников

1. Биберштейн, Н. Компас в мире сервис-ориентированной архитектуры (SOA) / Н. Биберштейн, С. Боуз. – СПб. : КУДИЦ-Пресс, 2007. – 256 с.
2. Service-Oriented Architecture [Электронный ресурс]. – 2016. – Режим доступа : http://www.oracle.com/us/products/middleware/soa/overview/.
3. SOA и web-сервисы. IBM developerWorks Россия [Электронный ресурс]. – 2016. – Режим доступа : https://www.ibm.com/developerworks/ru/webservices/.
4. Как работает Oracle Service Bus. Oracle [Электронный ресурс]. – 2016. – Режим доступа : http://www.oracle.com/technetwork/ru/middleware/service-bus/service-bus-1534004-ru.html.
5. WebSphere Enterprise Service Bus – Library [Электронный ресурс]. – 2016. – Режим доступа : http://www-01.ibm.com/software/integration/wsesb/library/.
6. Using IBM WebSphere Message Broker as an ESB with WebSphere Process Server / C. Sadtler [et al.]. – IBM Redbooks, 2008. – 368 с.

Св. план 2017, поз. 31

Учебное издание

Пилецкий Иван Иванович
Козуб Виктор Николаевич

СОВРЕМЕННЫЕ СРЕДСТВА ПРОЕКТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ. SOA-АРХИТЕКТУРЫ

ПОСОБИЕ

Редактор *Е. С. Чайковская*
Компьютерная правка, оригинал-макет *М. В. Касабуцкий*

Подписано в печать 19.05.2017. Формат 60×84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 3,84. Уч.-изд. л. 4,0. Тираж 150 экз. Заказ 16.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
ЛП №02330/264 от 14.04.2014.
220013, Минск, П. Бровки, 6