Введение в сервис-ориентированную архитектуру

Содержание

Введение в SOA. Время SOA еще не пришло

Аннотация

Рост интереса к сервис-ориентированной архитектуре (service-oriented architecture - SOA) во много связан с поддержкой новой идеологии построения информационных систем, сменившей собой клиент-серверную архитектуру, крупными игроками рынка, такими как Microsoft, IBM, SAP. Тем не менее, на пути к всеобъемлющей интеграции приложений, за которую так ратуют сторонники SOA, стоит немало вопросов и проблем, делающей конкретную реализацию этой красивой концепции весьма затруднительной.

Проблема в корне

SOA набирает обороты. Об этом свидетельствует как постоянно возрастающий интерес к Web-сервисам, на технологии которых основана эта концепция, так и данные аналитических компаний. По оценкам экспертов Gartner, к 2006 году более 60% компаний будут рассматривать сервис-ориентированную архитектуру как основу для построения и выполнения своих бизнес-приложений, а специалисты Giga Research считают, что в ближайшие два-три года большинство производителей буду использовать технологии, основанные на Web-сервисах в качестве расширения существующих решений.

Нередко SOA называют новой идеологией, третьей ступенью развития ИТ-систем, последующей за мейнфреймами и, ставшей уже классической, клиент-серверной архитектурой. Тем не менее, совершенно очевидно, что идея SOA вовсе не нова, более того – ИТ-специалисты вынашивают ее в своих умах уже третье десятилетие. Проблема интеграция приложений, как новых программных решений, так и унаследованных систем, существует с момента появления первого приложения, когда возникла задача одновременного использования данных двумя программными решениями. Однако, конкретная реализация SOA стала возможной лишь с появлением Web-сервисов, история которых насчитывает меньше десятилетия.

Эволюция программных архитектур

Классическим примером первой ступени, мейнфрейма, может служить система SAP R/2. В клиент-серверной архитектуре таким образцом стала широко распространненая SAP R/3, а в качестве сервис-ориентированного решения компания SAP позиционирует свою новую интеграционную платформу NetWeaver. "Теперь SAP производит не только и не столько ERP-систему, а скорее специализированную оболочку — набор средств для быстрой и эффективной интеграции всего того, что уже работает на предприятии, - отмечает **Тимур Аитов**, директор по развитию бизнеса РБК СОФТ. - Сервисно-ориентированный подход, лежащий в основе платформы NetWeaver, позволяет резко облегчить и ускорить внедрение корпоративной информационной системы на любом предприятии" - подчеркивает он.

Сервис-ориентированная архитектура основана на технологии Web-сервисов, способность к саморазвитию которых и обеспечивает так необходимую адаптивность ИТ-инфрастуктуры предприятия. Однако, между очевидным потенциалом Web-сервисов и их реальными возможностями сегодня существует довольно большой разрыв. Именно в базировании SOA на Web-сервисах кроется ее основная проблема. Перевод бизнес-критичных приложений корпорации на использование их в рамках этой концепции требует наличия единых стандартов на Web-сервисы. Существующих требований явно недостаточно для решения сложных, комплексных задач, а работа над созданием более совершенных стандартов еще очень далека от совершения и в этом немалую роль играет противостояние крупнейших гигантов ИТ-индустрии, таких как Microsoft, с остальным миром производителей программных продуктов.

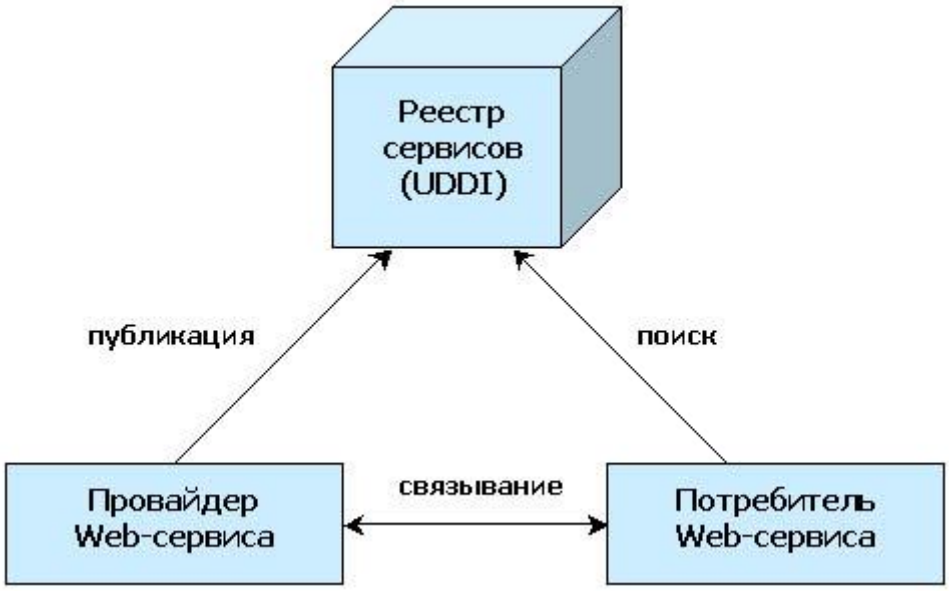
Справедливости ради, стоит отметить, что после анонсирования Microsoft Visual Studio Team System 2005, а также предложений IBM по конкретным решениям, основанным на SOA, это противостояние стало минимальным, и можно ожидать продвижения в разработке единых стандартов, регламентирующих использование Web-сервисов. Для успеха концепции сервис-ориентированной архитектуры необходимо наличие единого интерфейсного языка, описывающего Web-сервисы. В последнее время активно ведется работа над совершенствованием такого языка – Web Services Description Language (WSDL).

Некоторые существующие предварительные спецификации на Web-сервисы уже сегодня призваны гарантировать доставку сообщений SOAP по назначению и поддержку стабильности состояния длительных транзакций. Тем не менее, по мнению многих экспертов, говорить о повсеместном использовании подобных «сырых» спецификаций еще очень рано. Отсутствие единых стандартов на Web-сервисы не позволяет компании реализовать критичные бизнес-процессы в рамках SOA и в значительной степени сдерживает развитие этой концепции.

«Подводные камни» SOA

Классическое представление SOA реализуется в виде треугольника, во которого главе стоит реестр сервисов (Universal Description, Discovery and Integration - UDDI), а в двух остальных вершинах – провайдер и потребитель Web-сервиса.

Классическое представление концепции SOA



Реестр содержит исчерпывающую информацию по всем сервисам и службам, которую обязан опубликовать в нем провайдер соответствующего сервиса. Потребитель сервиса отправляет необходимый запрос в реестр, который и обеспечивает связывание его с провайдером. Еще одна проблема SOA связана с отсутствием четкого перечня требований к UDDI. Более того, совершенно непонятно, каким образом осуществляется выбор лучшего сервиса для выполнения бизнес-функции по запросу потребителя.

Другой немаловажный вопрос, связанный с самой технологией Web-сервисов, - использование протокола HTTP для передачи данных. С одной стороны это позволяет реализовать саму идеологию интеграции распределенных гетерогенных систем и обеспечить ее универсальность, а с другой - вызывает большое количество проблем, связанных с надежностью и безопасностью проведения транзакций. Более того, при использовании Web-сервисов, когда компоненты одного приложения могут отключаться и взаимодействовать с компонентами совершенно другого приложения, особенно остро встает проблема разграничения доступа и построения распределенной архитектуры защиты.

Основным достоинством концепции SOA, позволяющей реализовать адаптивность и эластичность ИТ-инфраструктуры является так называемая слабая связанность, которая подразумевает под собой отсутствие жестких связей между программными компонентами, что позволяет значительно оперативнее создавать новые приложения. Тем не менее, здесь же

кроется и другой «подводный» камень SOA, а точнее проблема ее конкретной реализации. На первое место при создании SOA-приложений выходит проектирование интерфейса, что, в свою очередь, выдвигает новые требования к разработчикам, связанные с кардинальной переменой самой идеологии программной разработки, которые не могут обеспечить современные средства проектирования, изначально ориентированные на классическую клиент-серверную архитектуру.

Подобных, достаточно спорных вопросов в реализации SOA довольно много. Безусловно, использование XML обеспечивает необходимую универсальность, но с другой стороны, закономерно встает проблема производительность SOA-приложений, скорость работы которых напрямую связана с так называемым «весом» XML-транзакций, на обработку которых требуется значительно большее время, нежели на классические двоичные транзакции. Сторонники SOA видят решение этой проблемы в появлении специализированных программных решений – «ускорителей» XML.

Опять же, всеобъемлющая интеграция приложений, являющаяся одним из главных аргументов сервис-ориентированной архитектуры, также не является универсальной. В том числе, в такой важной области как поддержка унаследованных бизнес-критичных приложений. Конечно, сама концепция SOA подразумевает подобную интеграцию, но безболезненно она пройдет лишь для приложений изначально спроектированных под SOA. А как же быть с программными решениями, которые существуют в компании достаточно давно и в наибольшей степени отвечают реализации тех или иных бизнес-функций? Ответ очевиден – необходимы дополнительные усилия разработчиков для создания промежуточного программного слоя, обеспечивающего интеграцию – задача достаточно сложная и не всегда однозначно решаемая.

Верно, но рано

Несмотря на все перечисленные трудности, возникающие при конкретной реализации SOA, ее идеология максимально удовлетворяет идее построения адаптивной и действительно эластичной ИТ-инфраструктуры предприятия, которая жизненно важна для бизнеса компании в условиях высокой конкуренции на современном рынке. Обеспечить подобную адаптивность позволяет принципиально новый подход к созданию приложений, отличающийся от традиционного наследования кода, - создание решений более высокого уровня из сервисов низкого. Само появление подобной концепции не что иное, как закономерный шаг в поиске ответов на вопросы интеграции приложений и оперативной разработки новых программных решений для решения возникающих задач.

Тем не менее, говорить о широком применении и конкретных реализациях SOA еще рано. Аналитики из компании ZapThink, которая специализируется на вопросах и проблемах сервис-ориентированной архитектуры, считают, что ближайшие два-три года станут периодом совершенствования концепции SOA в области стандартов и программных решений. Необходима тщательная детализация архитектуры и проработка многих технологических вопросов, в частности, задачи взаимодействия между Web-сервисами в рамках выполнения определенной бизнес-задачи и соблюдение SLA (Service Level Agreement – соглашение об уровне обслуживания) в случае использования «стороннего» сервиса. Ведь именно потенциальная возможность построения распределенных гетерогенных систем (и, как следствие, возможность передачи части бизнес-функций на аутсорсинг удаленному сервису исходя из относительной стоимости организации подобного сервиса собственными силами) – одно из основных достоинств SOA.

Важно отметить само «зачаточное» состояние рынка SOA-решений – ни один поставщик на сегодняшний день не в состоянии предоставить полнофункциональное решение. Существует множество подходов к реализации сервис-ориентированной архитектуры, каждый из которых лоббируется определенным игроком ИТ-рынка. Свои принципы построения адаптивного предприятия предлагают такие крупные компании, как Microsoft, IBM, SAP, Hewlett-Packard, Computeer Associates, Bea Systems, Sybase и другие.

Решающими вопросами для компании, принявшей решение на перестройку своей ИТ-инфраструктуры в соответствии с принципами SOA, должны быть опыт консультанта, реализующего проект и мировой опыт подобных решений, а также целесообразность подобного перехода для бизнеса компании в конкретно взятых условиях. Тем не менее, концепции SOA еще во многом надо окончательно «созреть» до повсеместного применения.

Алексей Куваев / CNews

Ресурсы

Адрес статьи: http://www.cnews.ru/newcom/index.shtml?2005/02/04/174034

SOAP Версия 1.2 Часть 0: Учебник для начинающих

Аннотация

Документ SOAP Версия 1.2 Часть 0: Учебник для начинающих не является нормативным, однако должен стать доступным учебным пособием по функциям SOAP Версия 1.2. В частности, он описывает функции SOAP на примере различных сценариев использования и призван дополнить нормативный текст Части 1 и Части 2 спецификаций SOAP 1.2.

Введение

SOAP Версия 1.2 Часть 0: Учебник для начинающих не является нормативным документом, но призван стать доступным учебным пособием по функциям SOAP Версия 1.2. Описывая типичные структуры SOAP-сообщений и шаблоны обмена SOAP-сообщениями, он предназначен для помощи техническим специалистам в понимании того, как использовать SOAP.

В частности, настоящий учебник описывает функции SOAP на примере различных сценариев использования и призван дополнить официальный текст SOAP Версия 1.2 Часть 1: Структура сообщений (здесь и далее [SOAP Часть 1]) и SOAP Версия 1.2 Часть 2: Приложения (здесь и далее [SOAP Часть 2]) спецификаций SOAP Версия 1.2.

Предполагается, что читатель знаком с основами синтаксиса XML, включая использование пространств имен и информационных множеств XML, а также с такими концепциями Web как URI и HTTP. Настоящий документ предназначен скорее для проектировщиков приложений, нежели для специалистов, занимающихся непосредственной реализацией SOAP, хотя для последних он также может быть полезен. Данный учебник нацелен на освещение основных функций SOAP Версия 1.2 и не претендует на полноту описания всех нюансов или частных (пограничных) случаев. Он не заменяет основные спецификации, которые должны быть использованы читателем для более полного понимания SOAP. С этой целью настоящий учебник снабжен исчерпывающим количеством ссылок на основные спецификации в тех случаях, когда вводятся либо используются новые понятия.

[SOAP Часть 1] определяет оболочку SOAP, описывающую общую структуру представления содержимого SOAP-сообщения, а также идентифицирующую кто должен взаимодействовать с сообщением, надо ли взаимодействовать с сообщением как с целым или же только с какой-либо его составляющей, и является ли обработка этой составляющей сообщения обязательной или необязательной. Оболочка SOAP также определяет структуру протокольной привязки, которая описывает, как может быть создана спецификация привязки SOAP к другому нижележащему протоколу.

[SOAP Часть 2] определяет модель данных SOAP, подробную схему написания кода для типов данных, используемых для передачи вызовов удаленных процедур (RPC), а также конкретную реализацию структуры привязки к нижележащему протоколу, определенную в [SOAP Часть 1]. Эта привязка позволяет обмениваться SOAP-сообщениями либо в качестве полезной нагрузки запросов и откликов посредством HTTP-метода POST, либо отправлять SOAP-сообщение в качестве отклика на HTTP-метод GET.

Настоящий документ (учебник) не является нормативным. Это означает, что он не описывает спецификацию SOAP Версия 1.2 полностью. Примеры, приведенные в нем,

призваны дополнить формальные спецификации и, в случае расхождений в интерпретации, формальные спецификации, конечно, имеют приоритет. Примеры, показанные здесь, являют собой предпочтительные варианты использования SOAP. В реальных сценариях использования, SOAP скорее всего будет частью общего решения и, без сомнения, найдутся иные специфичные для приложений требования, не охваченные этими примерами.

Общий обзор материала

SOAP Версия 1.2 дает определение основанной на XML информации, которая может быть использована для обмена структурированной и типизированной информацией между узлами в децентрализованной, распределенной среде. [SOAP Часть 1] поясняет, что SOAP-сообщение формально задается как информационное множество XML [XML Infoset]], дающее абстрактное описание его содержимого. Информационные множества могут иметь различные представления, один из общих примеров которых находится в документе XML 1.0 [XML 1.0].

SOAP является парадигмой, в основе своей не имеющей состояний, характеризующейся односторонним обменом сообщениями. Однако приложения могут создавать более сложные шаблоны взаимодействия (например, запрос/отклик, запрос/множественные отклики и т. д.), сочетая односторонний обмен с функциональностью, предоставляемой нижележащим протоколом и/или специфичной для конкретного приложения информацией. SOAP умалчивает о семантике каких бы то ни было специфичных для приложения данных, которые он передает - это является прерогативой таких вопросов, как маршрутизация SOAP-сообщений, надежность передачи данных, обеспечение безопасности посредством брандмауэра и т. д. SOAP представляет собой структуру, опираясь на которую, можно весьма гибко передавать специфичную для приложений информацию. Помимо этого, SOAP дает полное описание необходимых действий SOAP-узла при получении SOAP-сообщения.

Раздел 2 настоящего документа представляет собой введение в основные функции SOAP, начиная с простейших сценариев использования, а именно отправки SOAP-сообщения "в один конец", и кончая различными видами обмена сообщениями типа "запрос-отклик", включая RPC. В этом разделе также описаны ситуации возникновения ошибок.

Раздел 3 содержит обзор модели обработки SOAP, которая описывает правила начального построения сообщений, правила, согласно которым сообщения обрабатываются при получении в промежуточных или конечных пунктах назначения. Также описываются правила, согласно которым отдельные составляющие сообщения могут быть вложены, удалены или изменены действиями узлов-посредников.

Раздел 4 настоящего документа описывает способы передачи SOAP-сообщений при реализации различных сценариев использования. Он описывает HTTP-привязку SOAP, определенную в [SOAP Часть 2], а также пример того, как SOAP-сообщения могут передаваться с помощью email-сообщений. HTTP-привязка вводит в употребление два доступных приложениям шаблона обмена сообщениями, один из которых использует HTTP-метод POST, а другой - HTTP-метод GET. Также представлены примеры того, как могут быть реализованы вызовы удаленных процедур RPC (в частности те, которые предназначены для "безопасного" поиска информации) при обмене SOAP-сообщениями в соответствии с архитектурными принципами World Wide Web.

Раздел 5 настоящего документа дает трактовку различных подходов применения SOAP в более сложных сценариях использования. Описывается гибкий механизм, реализуемый

посредством заголовочных элементов, которые могут быть адресованы определенным промежуточным SOAP-узлам с целью предоставления взаимодействующим приложениям дополнительных услуг. Также описывается использование различных схем написания кода для сериализации специфичных для приложений данных в SOAP-сообщениях.

Раздел 6 описывает различия настоящей спецификации и спецификации SOAP Версия 1.1 [SOAP 1.1].

Раздел 7 содержит ссылки на другие документы.

Для удобства пользования на термины и понятия, используемые в данном учебнике, даны гиперссылки на их определения в основных спецификациях.

Соглашения об условных обозначениях

Все примеры SOAP-оболочек и SOAP-сообщений в настоящем учебнике представлены как [XML 1.0] XML-документы. [SOAP Часть 1] поясняет, что SOAP-сообщение формально определяется как информационное множество [XML InfoSet], которое представляет собой абстрактное описание его содержимого. Различие между информационными множествами XML SOAP и упомянутыми выше XML-документами вряд ли интересно тем, кто использует данный учебник в качестве введения в SOAP, те же, кто интересуется этим (как правило, это те, кто портирует SOAP на новые протокольные привязки, где сообщения могут иметь альтернативные представления), должны толковать эти примеры как ссылки на соответствующие информационные множества XML. Дальнейшее изложение этого вопроса можно найти в разделе 4 настоящего документа.

Префиксы пространств имен "env", "enc" и "rpc", используемые в разделах настоящего документа, обозначают пространства имен "http://www.w3.org/2003/05/soap-envelope" , "http://www.w3.org/2003/05/soap-encoding", и "http://www.w3.org/2003/05/soap-rpc" соответственно.

Префиксы пространств имен "xs" и "xsi", используемые в разделах настоящего документа, обозначают пространства имен "http://www.w3.org/2001/XMLSchema" и "http://www.w3.org/2001/XMLSchema-instance" соответственно, которые определены в спецификациях XML Schema [XML Schema Часть 1], [XML Schema Часть 2].

Необходимо отметить, что выбор любого другого префикса пространства имен может быть произвольным и то, какой это будет префикс, с точки зрения семантики не имеет значения.

Унифицированные идентификаторы ресурсов (Uniform Resource Identifier, URI) пространства имен общей формы "http://example.org/..." и "http://example.com/..." представляют зависимые от конкретного приложения либо контекстно-зависимые URI [RFC 2396].

Основные сценарии использования

В своей основе SOAP-сообщение является единицей односторонней передачи данных между SOAP-узлами, от SOAP-отправителя к SOAP-получателю, поэтому SOAP-сообщения должны компоноваться приложениями с целью реализации более сложных шаблонов взаимодействия, начиная шаблонами взаимодействия типа "запрос/отклик" и кончая множественными двунаправленными (двусторонними) "диалоговыми" вариантами обмена сообщениями.

Учебник раскрывает структуру SOAP-сообщений и обмена ими в нескольких простых сценариях использования, применительно к реализации приложения бронирования путешествия. Различные аспекты сценария этого приложения будут использоваться в учебнике и в дальнейшем. В этом сценарии приложение бронирования путешествия осуществляет бронь на планируемую поездку для сотрудника компании в ходе взаимодействия с сервисом продажи билетов. Информация, которой обмениваются приложение бронирования путешествия и сервис продажи билетов, посылается в виде SOAP-сообщений.

Конечным получателем SOAP-сообщения, отправленного приложением бронирования путешествия, является сервис продажи билетов, но на пути к конечному получателю SOAP-сообщение может проходить через один или несколько SOAP-посредников, каждый из которых может некоторым образом на него воздействовать. Простейшими примерами подобных SOAP-посредников могут быть узлы, протоколирующие запросы, осуществляющие их аудит или вносящие изменения в каждый запрос на бронирование путешествия. Примеры, а также более детальное обсуждение поведения и роли SOAP-посредников отложим до раздела 5.1.

Раздел 2.1 описывает запрос на бронирование путешествия в виде SOAP-сообщения, который дает возможность описать различные составляющие SOAP-сообщения.

В разделе 2.2.1 с помощью того же сценария демонстрируется отклик сервиса продажи билетов в виде другого SOAP-сообщения, формирующего часть диалогового обмена сообщениями как набор различных альтернатив, удовлетворяющих ограничениям присланного запроса на бронирование путешествия.

Раздел 2.2.2 предполагает, что будущим путешественником были одобрены различные параметры бронирования путешествия, и далее, посредством обмена сообщениями между приложением бронирования путешествий и сервисом продажи билетов, смоделированного как вызов удаленной процедуры (RPC), подтверждается оплата брони.

Раздел 2.3 демонстрирует примеры обработки ошибок.

SOAP-сообщения

Пример 1 показывает данные, необходимые для работы приложения бронирования путешествий, в виде SOAP-сообщения.

Пример 1

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
          env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
          env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <n:name>Eke Jygvan Шyvind</n:name>
  </n:passenger>
 </env:Header>
 <env:Body>
  <p:itinerary
    xmlns:p="http://travelcompany.example.org/reservation/travel">
   <p:departure>
     <p:departing>New York</p:departing>
     <p:arriving>Los Angeles</p:arriving>
     <p:departureDate>2001-12-14</p:departureDate>
     <p:departureTime>late afternoon</p:departureTime>
     <p:seatPreference>aisle</p:seatPreference>
   </p:departure>
   <p:return>
     <p:departing>Los Angeles</p:departing>
     <p:arriving>New York</p:arriving>
     <p:departureDate>2001-12-20</p:departureDate>
     <p:departureTime>mid-morning</p:departureTime>
     <p:seatPreference/>
   </p:return>
  </p:itinerary>
  <q:lodging
   xmlns:q="http://travelcompany.example.org/reservation/hotels">
   <q:preference>none</q:preference>
  </q:lodging>
 </env:Body>
</env:Envelope>
```

Образец SOAP-сообщения бронирования путешествия, содержащий заголовочный блок и тело сообщения

SOAP-сообщение примера 1 содержит два специфичных для SOAP субэлемента, находящихся внутри более общего элемента `env:Envelope`, а именно `env:Header` и `env:Body`. Содержимое этих элементов определяется приложением и не рассматривается спецификацией SOAP, хотя в дальнейшем будет немного сказано о том, как подобные элементы должны обрабатываться.

Заголовочный элемент SOAP не является обязательным, однако он был включен в пример для того, чтобы объяснить некоторые функции SOAP. Заголовок SOAP является расширением, предоставляющим способ передачи в SOAP-сообщениях информации, вообще говоря не являющейся полезной для приложения. Подобная «контрольная» информация включает, например, директивы прохождения сообщения или контекстную информацию, относящуюся к обработке сообщения. Это позволяет подстраивать SOAP-сообщения под каждое конкретное приложение. Следующие непосредственно за `env:Header` дочерние

элементы называются заголовочными блоками. Они представляют логическую группировку данных, которые, как показано позже, могут быть индивидуально адресованы SOAP-узлам, встречаемым сообщением на пути от отправителя к конечному получателю.

Заголовки SOAP были созданы в предположении появления различных вариантов использования SOAP, многие из которых будут вовлекать во взаимодействие другие обрабатывающие SOAP-сообщения узлы, называемые SOAP-посредниками – на пути сообщения от начального отправителя SOAP-сообщения до конечного SOAP-получателя. Это позволяет SOAP-посредникам предоставлять дополнительные сервисы. Заголовки, как показано далее, могут быть просмотрены, вставлены, удалены или пересланы SOAP-узлами, встреченными на пути SOAP-сообщения. (Однако, необходимо помнить, что спецификации SOAP не описывают содержимое заголовочных элементов, или то, как SOAP-сообщения маршрутизируются между узлами. Они также не описывают каким образом определяется маршрут сообщения и т. д. Эти вопросы решаются приложением в целом и могут быть предметом рассмотрения других спецификаций.)

Тело SOAP-сообщения является обязательным элементом внутри `env:Envelope`, содержащим основную информацию SOAP-сообщения, которая должна быть передана из начальной точки пути сообщения в конечную.

В примере 1, заголовок содержит два заголовочных блока, каждый из которых определен в собственном пространстве имен XML, и отражает некоторый аспект общей обработки тела SOAP-сообщения. Для приложения бронирования путешествия, подобная, принадлежащая к запросу в целом, «метаинформация» содержится в заголовочном блоке `reservation`, который представляет собой ссылку на этот экземпляр запроса, а также содержит его временную отметку. «Метаинформация», служащая для идентификации будущего путешественника, содержится в блоке `passenger`.

Заголовочные блоки `reservation` и `passenger` должны обрабатываться следующим SOAP-посредником, встреченным на пути сообщения, либо, в случае отсутствия узла-посредника, конечным получателем сообщения. На тот факт, что сообщение адресовано следующему SOAP-узлу, встреченному на пути сообщения, указывает присутствие атрибута `env:role` со значением «http://www.w3.org/2003/05/soap-envelope/role/next» (здесь и далее просто «next»). Этот атрибут реализует роль, исполнять которую обязаны все SOAP-узлы. Присутствие же атрибута `env:mustUnderstand` со значением «true» указывает на то, что узел (узлы), обрабатывающий заголовочные блоки, должен обрабатывать их в строгом соответствии с их спецификациями либо не обрабатывать SOAP-сообщение вовсе и выдать сообщение об ошибке. Необходимо отметить, что если заголовочный блок обрабатывается, ввиду ли присутствия `env:mustUnderstand=»true»` либо по какой-либо другой причине, такой блок должен быть обработан в соответствии со спецификацией этого блока. Спецификации заголовочных блоков определяются конкретным приложением и не являются предметом рассмотрения SOAP. Подробное изложение обработки SOAP-сообщения в зависимости от значений вышеупомянутых атрибутов содержится в разделе 3.

Решение по поводу того, какие данные поместить в заголовочный блок и тело SOAP-сообщения, должно быть принято на этапе проектирования приложения. Необходимо помнить, что заголовочные блоки могут быть адресованы различным узлам, которые могут встречаться на пути сообщения от отправителя к конечному получателю. Подобные промежуточные SOAP-узлы могут предоставлять дополнительные услуги, используя данные, содержащиеся в этих заголовочных блоках. В примере 1, данные о пассажире помещены в заголовочный блок для того, чтобы проиллюстрировать их использование SOAP-посредником для выполнения некоторой дополнительной обработки. Например, как показано далее в

разделе 5.1, уходящее сообщение изменяется SOAP-посредником путем добавления в качестве еще одного заголовочного блока политик совершения путешествия пассажиром.

Элемент `env:Body` и ассоциированные с ним дочерние элементы `itinerary` и `lodging`, вовлечены в обмен информацией между начальным отправителем SOAP-сообщения и SOAP-узлом на его пути, выступающим в роли конечного получателя SOAP-сообщения, которым является сервис продажи билетов. Поэтому элемент `env:Body` с его содержимым всецело адресован конечному получателю и должен быть им понят. Средства, посредством которых SOAP-узел может выполнять эту роль, не определяются спецификацией SOAP. Они определяются общей семантикой приложения и ассоциированного с ним потоком сообщений.

Необходимо отметить, что SOAP-посредник может играть роль конечного получателя для рассматриваемого сообщения и таким образом также может обрабатывать `env:Body`. Однако, хотя такой тип поведения и не может быть предугадан, это не означает, что обработка сообщения может производиться без должного внимания, поскольку в этом случае могут быть искажены первоначальные намерения отправителя сообщения. Это может иметь нежелательные побочные эффекты (такие как необработка заголовочных блоков, которые могут быть адресованы узлам-посредникам, находящимся далее на пути следования сообщения).

SOAP-сообщение, подобное представленному в примере 1, может быть передано посредством различных нижележащих протоколов и использоваться в различных шаблонах обмена сообщениями. Например, для web-доступа к сервису продажи билетов, оно может быть помещено в тело http- запроса POST. В случае привязки к другому протоколу, оно может быть отправлено в email-сообщении (см. раздел 4.2). Раздел 4 далее опишет то, как SOAP-сообщения могут передаваться посредством различных нижележащих протоколов. Сейчас же предположим, что механизм для передачи сообщения уже существует, и в дальнейшей части этого раздела сконцентрируемся на деталях SOAP-сообщений и их обработки.

Обмен SOAP-сообщениями

SOAP Версия 1.2 является простой схемой обмена сообщениями для передачи информации в виде информационных множеств XML между SOAP-отправителем, инициирующим обмен, и конечным SOAP-получателем. Более интересные сценарии обычно используют обмен множеством сообщений между двумя узлами. Простейшим примером подобного обмена является шаблон "запрос-отклик". Некоторые более ранние реализации [SOAP 1.1] акцентировали внимание на использовании этого шаблона как средстве передачи вызовов удаленных процедур (RPC), однако важно отметить, что не каждый обмен SOAP-сообщениями типа "запрос-отклик" может или должен быть реализован в виде вызовов удаленных процедур. Последние как правило используются, когда необходимо реализовать некое определенное программное поведение, использующее обмен сообщениями и удовлетворяющее заранее определенному описанию удаленного вызова и его результата.

Более широкий набор сценариев использования, которые укладываются в определение шаблона обмена сообщениями типа "запрос-отклик", может быть реализован просто с помощью XML-контента в SOAP-сообщениях при реализации двунаправленного "диалога", где семантика определяется отправляющим и получающим приложениями. Раздел 2.2.1 описывает случай, когда обмен XML-контентом в SOAP-сообщениях между приложением бронирования путешествия и сервисом продажи билетов осуществляется посредством

шаблона диалогового обмена сообщениями, в то время как раздел 2.2.2 предлагает пример реализации обмена уже в виде вызовов удаленных процедур (RPC).

Диалоговый обмен сообщениями

Продолжая рассмотрение сценария бронирования путешествия, пример 2 демонстрирует SOAP-сообщение, полученное от сервиса продажи билетов в ответ на запрос о бронировании, реализованный сообщением примера 1. Этот отклик выделяет из запроса некоторую информацию, а именно выбор аэропорта в городе отправления.

Пример 2

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-29T13:35:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <n:name>Eke Jygvan Шyvind</n:name>
  </n:passenger>
 </env:Header>
 <env:Body>
  <p:itineraryClarification
    xmlns:p="http://travelcompany.example.org/reservation/travel">
   <p:departure>
     <p:departing>
       <p:airportChoices>
          JFK LGA EWR
       </p:airportChoices>
     </p:departing>
   </p:departure>
   <p:return>
     <p:arriving>
       <p:airportChoices>
         JFK LGA EWR
       </p:airportChoices>
     </p:arriving>
   </p:return>
  </p:itineraryClarification>
 </env:Body>
</env:Envelope>
```

SOAP-сообщение, отправленное в ответ на сообщение примера 1.

Как было описано ранее, элемент `env:Body` содержит основной контент сообщения, удовлетворяющий схеме описания в пространстве имен XML http://travelcompany.example.org/reservation/travel, который в этом примере включает в себя список различных вариантов аэропортов. В этом примере в качестве отклика возвращены заголовочные блоки примера 1 (с измененными значениями нескольких субэлементов). Это

позволяет осуществлять корреляцию сообщений на SOAP-уровне, однако подобные заголовки имеют также и другие применения, определяемые спецификой приложения.

Обмен сообщениями в Примерах 1 и 2 является случаем, когда посредством SOAP-сообщений осуществляется обмен XML-контентом, удовлетворяющим некоторой определенной приложением схеме. Напомним, что обсуждение средств передачи сообщений содержится в разделе 4.

Однако легко видеть, как подобный обмен может быть построен с помощью двунаправленного "диалогового" шаблона обмена сообщениями. Пример 3 демонстрирует SOAP-сообщение, отправленное приложением бронирования путешествия в ответ на сообщение примера 2 и содержащее аэропорт, выбранный из списка доступных аэропортов. Заголовочный блок `reservation` с тем же значением субэлемента `reference` содержится в каждом сообщении этого диалога, тем самым предоставляя, в случае необходимости, возможность коррелировать сообщения на уровне приложения.

Пример 3

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation
     xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
    <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
    <m:dateAndTime>2001-11-29T13:36:50.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <n:name>Eke Jygvan Шyvind</n:name>
  </n:passenger>
 </env:Header>
 <env:Body>
  <p:itinerary
   xmlns:p="http://travelcompany.example.org/reservation/travel">
   <p:departure>
     <p:departing>LGA</p:departing>
   </p:departure>
   <p:return>
     <p:arriving>EWR</p:arriving>
   </p:return>
  </p:itinerary>
 </env:Body>
</env:Envelope>
```

Отклик на сообщение примера 2, реализующий диалоговый обмен сообщениями.

Вызовы удаленных процедур

Одна из целей SOAP Версия 1.2 - инкапсулировать функциональность, реализуемую посредством вызовов удаленных процедур, используя широкие возможности применения и гибкость XML. Раздел 4 SOAP Часть 2 определяет унифицированное представление вызовов RPC и откликов на них посредством SOAP-сообщений. Этот раздел продолжает разбор

сценария бронирования путешествия для иллюстрации использования SOAP-сообщений для вызова удаленных процедур и получения откликов на них.

С этой целью следующий пример демонстрирует процесс оплаты путешествия с помощью кредитной карточки. (Предполагается, что диалоговый обмен сообщениями, описанный в разделе 2.2.1, завершился подтверждением маршрута путешествия.) В дальнейшем подразумевается, что оплата происходит в контексте всей транзакции, когда снятие денег со счета с помощью кредитной карточки осуществляется только при условии подтверждения и брони путешествия и проживания (процесс бронирования проживания не показан в примерах, но предполагается, что он происходит аналогичным образом). Приложение бронирования путешествия предоставляет информацию о кредитной карточке, а также информацию об успешном завершении различных операций, происходящих в результате снятия денег со счета с помощью кредитной карточки и возврата кода бронирования. Это взаимодействие между приложением бронирования путешествия и сервисом продажи билетов с целью получения брони и ее оплаты реализовано посредством SOAP RPC.

Для вызова SOAP RPC требуется следующая информация:

1. Адрес SOAP-узла места назначения;
2. Имя процедуры либо метода;
3. Наименования и значения всех аргументов, передаваемых процедуре или методу вместе с выходными параметрами и возвращаемым значением;
4. Четкое разделение аргументов, используемых для идентификации web-ресурса, являющегося действительным местом назначения RPC, от аргументов, содержащих данные и контрольную информацию, используемых для обработки вызова ресурсом места назначения RPC.
5. Определение шаблона обмена сообщениями, а также так называемого "Web-метода" (о нем будет рассказано несколько позже), которые будут использоваться для передачи RPC.
6. Данные, которые могут быть переданы как часть заголовочных блоков SOAP. Эти данные не являются обязательными.

Вышеперечисленная информация может быть выражена посредством широкого набора средств, включая формальный Interface Definition Language (IDL, Язык Определения Интерфейсов). Необходимо заметить, что SOAP не предоставляет никаких средств IDL, ни формальных ни неформальных. Надо также отметить, что приведенная выше информация слегка отличается от информации, которая в общем случае необходима для вызова других RPC, не имеющих дела с SOAP.

Относительно пункта 1 приведенного выше списка можно сказать, что, с точки зрения SOAP, существует SOAP-узел, который "удерживает" или "поддерживает" пункт назначения RPC. Это SOAP-узел, играющий роль конечного SOAP-получателя. Как этого требует пункт 1, конечный получатель может идентифицировать пункт назначения поименованной процедуры либо метода по его URI. Каким образом пункт назначения становится доступен по URI - зависит от нижележащего протокола привязки. Одна из возможностей - URI, идентифицирующая пункт назначения, которая может быть включена в заголовочный SOAP-блок. Некоторые протокольные привязки, такие как, например, HTTP-привязка SOAP, определенная в [SOAP Часть 2], предлагают механизм передачи URI вне SOAP-сообщения. Говоря в общем, одним из вопросов, описываемых спецификацией протокольной привязки, должно быть описание способа передачи URI пункта назначения в контексте использования

данной привязки. Раздел 4.1 дает несколько конкретных примеров того, как URI передается в случае стандартизованной протокольной HTTP-привязки SOAP.

Пункты 4 и 5 приведенного выше списка необходимы для гарантии того, что RPC-приложения, использующие SOAP, могут делать это (т. е. использовать SOAP) образом, совместимым с архитектурными принципами World Wide Web. В разделе 4.1.3 показывается, как можно использовать информацию пунктов 4 и 5.

В оставшейся части этого раздела предполагается, что RPC, передаваемые в SOAP-сообщении как показано в примере 4, адресованы нужному узлу и успешно отправлены. Целью этого раздела является заострение внимания на синтаксических аспектах запросов RPC и откликов на них, передаваемых SOAP-сообщениями.

Пример 4

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
 <env:Header>
   <t:transaction
           xmlns:t="http://thirdparty.example.org/transaction"
           env:encodingStyle="http://example.com/encoding"
           env:mustUnderstand="true" >5</t:transaction>
 </env:Header>
 <env:Body>
  <m:chargeReservation
      env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
         xmlns:m="http://travelcompany.example.org/">
   <m:reservation xmlns:m="http://travelcompany.example.org/reservation">
    <m:code>FT35ZBQ</m:code>
   </m:reservation>
   <o:creditCard xmlns:o="http://mycompany.example.com/financial">
    <n:name xmlns:n="http://mycompany.example.com/employees">
           Eke Jygvan Шyvind
    </n:name>
    <o:number>123456789099999</o:number>
    <o:expiration>2005-02</o:expiration>
   </o:creditCard>
  </m:chargeReservation>
 </env:Body>
</env:Envelope>
```

SOAP-запрос RPC с обязательным заголовком и двумя входными параметрами.

RPC сам по себе переносится как дочерний элемент `env:Body` и реализован структурой `struct` , которой присваивается имя процедуры или метода, в данном случае `chargeReservation`. (Структура `struct` является понятием модели данных SOAP, определенным в [SOAP Часть 2], которая описывает тип структуры или записи, присутствующий в некоторых широко распространенных языках программирования). Конструкция RPC в примере (чье формальное описание подробно не приводится) имеет на входе два параметра: `reservation`, соответствующий планируемому путешествию, определяемому посредством кода брони `code`, и информацию о кредитной карточке `creditCard`. Дальнейший фрагмент кода также является структурой `struct`, которая включает три элемента: имя держателя кредитной карточки `name`, номер кредитной карточки `number` и дату истечения срока действия кредитной карточки `expiration`.

В этом примере атрибут `env:encodingStyle` со значением http://www.w3.org/2003/05/soap-encoding показывает, что содержимое структуры `chargeReservation` было сериализовано согласно правилам написания кода SOAP - специальным правилам, определенным в SOAP Часть 2 Раздел 3. Хотя SOAP и определяет эту схему написания кода, необязательно использовать именно ее. Из спецификации ясно, что для описания данных, специфичных для конкретного приложения, внутри SOAP-сообщения могут быть использованы другие схемы написания кода. Для упоминания о других схемах написания кода используется атрибут `env:encodingStyle`, уточняющий заголовочные блоки и субэлементы тела сообщения. Выбор значения этого атрибута зависит от конкретного приложения; способность вызывающего и вызываемого приложений взаимодействовать между собой предполагается установленной "out-of-band". Раздел 5.2 демонстрирует пример использования другой схемы написания кода.

Как было отмечено в пункте 6 вышеприведенного списка, при использовании RPC может также потребоваться передача дополнительной информации, которая может быть важна для обработки вызова в распределенной среде, но не является частью формального описания процедуры или метода. (Необходимо отметить, что предоставление такой дополнительной контекстной информации нехарактерно для RPC, но, вообще говоря, может потребоваться для обработки какими-либо распределенными приложениями.) В примере RPC передается в контексте всей транзакции, которая состоит из несколько процессов, каждый из которых должен успешно завершиться, прежде чем успешно завершится сам RPC. Пример 4 показывает как заголовочный блок `transaction`, адресованный конечному получателю (это предполагается отсутствием атрибута `env:role`), используется для передачи подобной информации. (Значение "5" в некоторых идентификаторах транзакции является отвлеченным и имеет смысл только для приложения. Здесь нет необходимости в дальнейшем обсуждении семантики заголовка примера 4, специфичной для конкретного приложения, так как это не является предметом дискуссии о синтаксических аспектах SOAP-сообщений RPC.)

Предположим, что RPC в примере осуществления платежа был спроектирован с целью включения описания процедуры, которое указывает на наличие двух параметров на выходе: один представляет собой код ссылки брони, а другой - URL, по которой могут быть получены детали, касающиеся осуществленной брони. Отклик RPC возвращается в элементе SOAP-сообщения `env:Body`, который реализован как структура `struct`, состоящая из имени процедуры `chargeReservation` и добавляемого, как правило, слова "Response". Двумя параметрами на выходе, сопровождающими отклик RPC, являются: буквенно-цифровой код брони `code`, и `viewAt`, - URI местоположения брони.

Это показано в примере 5а, где заголовок идентифицирует транзакцию, в рамках которой этот RPC выполняется.

Пример 5а

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
 <env:Header>
     <t:transaction
        xmlns:t="http://thirdparty.example.org/transaction"
          env:encodingStyle="http://example.com/encoding"
            env:mustUnderstand="true">5</t:transaction>
 </env:Header>
 <env:Body>
     <m:chargeReservationResponse
         env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
             xmlns:m="http://travelcompany.example.org/">
```

```
        <m:code>FT35ZBQ</m:code>
        <m:viewAt>
          http://travelcompany.example.org/reservations?code=FT35ZBQ
        </m:viewAt>
      </m:chargeReservationResponse>
 </env:Body>
</env:Envelope>
```

Отклик RPC с двумя параметрами на выходе для RPC, показанного в примере 4

RPC часто имеют описания, в которых какой-либо один параметр на выходе отличен от других - это так называемое "возвращаемое" значение.Соглашение об обозначениях SOAP RPC предоставляет способ, позволяющий отличать это "возвращаемое" значение в описании процедуры от других параметров на выходе. Чтобы продемонстрировать это, пример осуществления платежа изменен так, что описание процедуры, аналогичное описанию процедуры в примере 5a, имеющее те же два параметра на выходе, дополнено "возвращаемым" значением, которое представляет собой список с двумя возможными значениями "confirmed" и "pending". RPC, удовлетворяющий этому описанию, показан в примере 5b, где, как и ранее, SOAP-заголовок идентифицирует транзакцию, выполняемую этим RPC.

Пример 5b

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
 <env:Header>
    <t:transaction
       xmlns:t="http://thirdparty.example.org/transaction"
         env:encodingStyle="http://example.com/encoding"
          env:mustUnderstand="true">5</t:transaction>
 </env:Header>
 <env:Body>
    <m:chargeReservationResponse
       env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
           xmlns:rpc="http://www.w3.org/2003/05/soap-rpc"
             xmlns:m="http://travelcompany.example.org/">
       <rpc:result>m:status</rpc:result>
       <m:status>confirmed</m:status>
       <m:code>FT35ZBQ</m:code>
       <m:viewAt>
        http://travelcompany.example.org/reservations?code=FT35ZBQ
       </m:viewAt>
    </m:chargeReservationResponse>
 </env:Body>
</env:Envelope>
```

RPC с "возвращаемым" значением и двумя параметрами на выходе для RPC, приведенного в примере 4.

В примере 5b возвращаемое значение идентифицируется элементом `rpc:result` и содержит XML Qualified Name (принадлежащее типу `xs:QName`) другого элемента внутри структуры `struct`, которым является `m:status`. Он, в свою очередь, содержит действительное возвращаемое значение - "confirmed". Эта техника позволяет строго типизировать согласно некоторой схеме действительное возвращаемое значение. Если элемент `rpc:result`

отсутствует, как в примере 5a, возвращаемое значение отсутствует либо принадлежит типу `void`.

В то же время, в принципе, использование SOAP для RPC не зависит от использования каких-либо особенных средств передачи RPC и получения отклика на него. Протокольная привязка, поддерживающая шаблон обмена сообщениями SOAP типа "запрос-отклик", более естественным образом может подходить для таких целей. Протокольная привязка, поддерживающая этот шаблон обмена сообщениями, предоставляет возможность корреляции запроса и отклика на него. Конечно, проектировщик RPC-приложения, может ввести корреляционный ID вызова и его отклика в SOAP-заголовке, таким образом делая RPC независимым от любого нижележащего механизма передачи. В любом случае проектировщики приложений должны иметь представление обо всех характеристиках протоколов, используемых для передачи SOAP RPC, таких как задержка передачи, синхронность и т. д.

В общем случае использования HTTP в качестве нижележащего протокола передачи, стандартизованном в SOAP Часть 2 Раздел 7, вызов RPC естественным образом мапируется на HTTP-запрос, а RPC-отклик мапируется на HTTP-отклик. Раздел 4.1 содержит примеры передачи RPC с использованием HTTP-привязки.

Однако, следует помнить, что хотя большинство примеров SOAP RPC и используют HTTP-привязку, существуют также и другие транспорты.

Сценарии обработки сообщений об ошибках

SOAP предоставляет модель обработки ситуаций, когда при обработке сообщения возникает ошибка. SOAP отдельно рассматривает условия, при которых ошибка возникла, и возможность указания на факт возникновения ошибки узлу, сгенерировавшему ошибку, либо любому другому узлу. Возможность указания на факт возникновения ошибки зависит от используемого механизма передачи сообщений, и одним из аспектов спецификации привязки SOAP к нижележащему протоколу является определение механизма оповещения об ошибках в случае их возникновения. В дальнейшей части этого раздела предполагается, что определенный механизм передачи сообщений уже используется для оповещения о возникающих ошибках во время обработки полученных сообщений, и дальнейшее внимание сосредотачивается на структуре SOAP-сообщения об ошибке.

SOAP-элемент `env:Body` имеет также еще одну характерную роль - он может содержать информацию об ошибке. Модель обработки ошибок SOAP (см. SOAP Часть 1 Раздел 2.6) требует, чтобы все специфичные для SOAP и приложений ошибки были описаны с использованием *единственного* специального элемента `env:Fault`, содержащегося в элементе `env:Body`. Элемент `env:Fault` содержит два обязательных субэлемента, `env:Code` и `env:Reason`, и (необязательно) специфичную для приложения информацию в субэлементе `env:Detail`. Другой необязательный субэлемент `env:Node` посредством URI определяет SOAP-узел, сгенерировавший ошибку. Отсутствие этого субэлемента означает, что ошибка была сгенерирована конечным получателем сообщения. Существует также еще один необязательный субэлемент, `env:Role`, определяющий роль, исполняемую сгенерировавшим ошибку узлом.

Субэлемент `env:Code` элемента `env:Fault` сам по себе подобен обязательному субэлементу `env:Value`, назначение которого определено в спецификации SOAP (см. SOAP Часть 1 Раздел 5.4.6), также как и назначение необязательного субэлемента `env:Subcode`.

Пример 6а демонстрирует SOAP-сообщение, присланное в ответ на RPC примера 4, и указывающее на неудачное завершение обработки RPC.

Пример 6а

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
              xmlns:rpc='http://www.w3.org/2003/05/soap-rpc'>
  <env:Body>
   <env:Fault>
     <env:Code>
       <env:Value>env:Sender</env:Value>
       <env:Subcode>
        <env:Value>rpc:BadArguments</env:Value>
       </env:Subcode>
     </env:Code>
     <env:Reason>
      <env:Text xml:lang="en-US">Processing error</env:Text>
      <env:Text xml:lang="cs">Chyba zpracovбnн</env:Text>
     </env:Reason>
     <env:Detail>
      <e:myFaultDetails
        xmlns:e="http://travelcompany.example.org/faults">
        <e:message>Name does not match card number</e:message>
        <e:errorcode>999</e:errorcode>
      </e:myFaultDetails>
     </env:Detail>
   </env:Fault>
 </env:Body>
</env:Envelope>
```

Пример SOAP-сообщения, указывающего на неудачное завершение обработки RPC примера 4

В примере 6а высокоуровневый элемент `env:Value` использует стандартизованное XML Qualified Name (принадлежащее типу `xs:QName`) для того, чтобы определить, что оно является ошибкой `env:Sender`, указывающей на то, что ошибка является синтаксической либо сообщение содержит некорректную информацию. (Когда ошибка `env:Sender` получена отправителем, предполагается, что будет сделано некоторое корректирующее действие, прежде чем аналогичное сообщение будет отправлено снова). Элемент `env:Subcode` не является обязательным, но если он присутствует, как в приведенном примере, он уточняет родительское значение. В примере 6а, элемент `env:Subcode` указывает на то, что специфичная ошибка RPC, `rpc:BadArguments`, определенная в SOAP Часть 2 Раздел 4.4, является причиной неудачного завершения обработки запроса.

Структура элемента `env:Subcode` была создана иерархической - каждый дочерний элемент `env:Subcode` имеет обязательный субэлемент `env:Value` и необязательный субэлемент `env:Subcode` - для передачи кода, специфичного для конкретного приложения. Эта иерархическая структура элемента `env:Code` реализует единый механизм передачи кодов ошибок нескольких уровней. Высокоуровневый элемент `env:Value` - основная ошибка, описанная в спецификациях SOAP Версия 1.2 (см. SOAP Часть 1 Раздел 5.4.6), она должна

быть понятна всем SOAP-узлам. Вложенный элемент `env:Value` является специфичным для конкретного приложения и представляет собой дальнейшее развитие и улучшение описания вышеупомянутой основной ошибки с точки зрения приложения. Некоторые из этих значений могут быть хорошо стандартизованы SOAP (например коды RPC, стандартизованные в SOAP 1.2 (см. SOAP Часть 2 Раздел 4.4)) или в некоторых других стандартах, использующих SOAP в качестве инкапсулирующего протокола. Единственным требованием для определения подобных специфичных для приложения значений субкодов является их соответствие пространству имен при использовании любого пространства имен отличного от пространства имен SOAP `env`, определяющего основную классификацию SOAP-ошибок. Со стороны SOAP не существует требований о том, что приложения должны понимать или хотя бы просматривать все уровни значений субкодов.

Субэлемент `env:Reason` не важен для алгоритмической обработки. Он нужен скорее для лучшего понимания ситуации человеком, и хотя это обязательный элемент, его возможные значения не нуждаются в стандартизации. Все, что необходимо - достаточно точно описать ситуацию, в которой возникла ошибка. Субэлемент `env:Text`, иметь один или несколько субэлементов `xml:lang` позволяющий приложениям выдавать сообщения о причинах ошибки на различных языках. (Приложения могут согласовывать конкретный язык текста ошибки посредством встроенного механизма, использующего SOAP-заголовки, однако это находится вне рассмотрения спецификации SOAP.)

Отсутствие субэлемента `env:Node` внутри `env:Fault` в примере 6a говорит о том, что сообщение было создано конечным получателем вызова. Содержимое `env:Detail`, как показано в примере, специфично для конкретного приложения.

Ошибка может быть также сгенерирована во время обработки SOAP-сообщения, если обязательный заголовочный элемент не был понят, либо информация, содержащаяся в нем, не может быть обработана. Ошибки в ходе обработки заголовочного блока могут быть также показаны с использованием элемента `env:Fault` внутри элемента `env:Body`, но со специальным заголовочным блоком - `env:NotUnderstood`, который идентифицирует вызывающий ошибку заголовочный блок.

Пример 6b демонстрирует пример отклика на RPC примера 4, показывающий неудачное завершение обработки заголовочного блока `t:transaction`. Необходимо отметить присутствие кода ошибки `env:MustUnderstand` в `env:Body`, а также идентификацию заголовка, который не был понят с помощью (неприспособленного для этого) атрибута `qname` в специальном (пустом) заголовочном блоке `env:NotUnderstood`.

Пример 6b

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope>
 <env:Header>
    <env:NotUnderstood qname="t:transaction"
                xmlns:t="http://thirdparty.example.org/transaction"/>
 </env:Header>
 <env:Body>
  <env:Fault>
   <env:Code>
    <env:Value>env:MustUnderstand</env:Value>
   </env:Code>
   <env:Reason>
      <env:Text xml:lang="en-US">Header not understood</env:Text>
      <env:Text xml:lang="fr">En-tкte non compris</env:Text>
```

```
        </env:Reason>
      </env:Fault>
    </env:Body>
  </env:Envelope>
```

Пример SOAP-сообщения, показывающего неудачное завершение обработки заголовочного блока примера 4

Если не были поняты несколько обязательных заголовочных блоков, тогда каждый из них может быть определен своим атрибутом `qname` в сериях таких заголовочных блоков `env:NotUnderstood`.

Модель обработки SOAP

После того как были установлены различные синтаксические аспекты SOAP-сообщения наряду с некоторыми простыми шаблонами обмена сообщениями, в этом разделе дается общий обзор модели обработки SOAP (определенной в SOAP Часть 1 Раздел 2). Модель обработки SOAP описывает действия SOAP-узла при получении SOAP-сообщения.

Пример 7a демонстрирует SOAP-сообщение с несколькими заголовочными блоками (их содержимое для краткости опущено). Различные варианты этого сообщения будут использованы для иллюстрации различных аспектов модели обработки в этом разделе и в дальнейшем.

Пример 7a

```
<?xml version="1.0" ?>
 <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
   <env:Header>
     <p:oneBlock xmlns:p="http://example.com"
            env:role="http://example.com/Log">
     ...
     ...
     </p:oneBlock>
     <q:anotherBlock xmlns:q="http://example.com"
       env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
     ...
     ...
     </q:anotherBlock>
     <r:aThirdBlock xmlns:r="http://example.com">
     ...
     ...
     </r:aThirdBlock>
   </env:Header>
   <env:Body >
     ...
     ...
   </env:Body>
 </env:Envelope>
```

SOAP-сообщение, показывающее многообразие заголовочных блоков.

Модель обработки SOAP описывает (логические) действия, предпринимаемые SOAP-узлом при получении SOAP-сообщения. От узла требуется проанализировать SOAP-составляющие сообщения, а именно те элементы, которые находятся в пространстве имен SOAP "env". Такие элементы являются оболочками сами для себя - они имеют и заголовок и тело. Первым шагом, конечно, является общая проверка синтаксической правильности SOAP-сообщения. То есть проверки того, что оно соответствует информационному множеству XML SOAP при условии соблюдения ограничений использования определенных XML-конструкций - Инструкций Обработки и Определений Типов Документа - как это определено в SOAP Часть 1, Раздел 5.

Атрибут "role"

Дальнейшая обработка заголовочных блоков и тела зависит от роли (ролей), исполняемой SOAP-узлом при обработке данного сообщения. SOAP определяет (необязательный) атрибут заголовочного блока `env:role` - синтаксически `xs:anyURI` - идентифицирующий роль, которую исполняет предполагаемый пункт назначения этого заголовочного блока. SOAP-узел нужен для обработки заголовочного блока, если он исполняет роль, идентифицируемую URI. То, каким образом SOAP-узел исполняет некоторую определенную роль, не является предметом рассмотрения спецификации SOAP.

Определены три стандартизованные роли (см. SOAP Часть 1 Раздел 2.2):

- "http://www.w3.org/2003/05/soap-envelope/role/none" (далее просто "none");
- "http://www.w3.org/2003/05/soap-envelope/role/next" (далее просто "next");
- "http://www.w3.org/2003/05/soap-envelope/role/ultimateReceiver" (далее просто "ultimateReceiver").

В примере 7a заголовочный блок `oneBlock` адресован любому SOAP-узлу, играющему определенную приложением роль, определяемую URI http://example.com/Log. Для целей иллюстрации предполагается, что спецификация этого заголовочного блока требует от любого SOAP-узла, играющего эту роль, протоколировать сообщение полностью.

Каждый SOAP-узел, получающий сообщение с заголовочным блоком, имеющим атрибут `env:role` "next", должен быть в состоянии обработать содержимое элемента, так как это стандартизованная роль, которую каждый SOAP-узел обязан исполнять. Заголовочный блок, имеющий этот атрибут, ожидает рассмотрения и (возможно) будет обработан следующим SOAP-узлом на пути сообщения. (Предполагается, что такой заголовок не был удален в результате обработки каким-либо предшествующим узлом.)

В примере 7a заголовочный блок `anotherBlock` адресован следующему узлу на пути сообщения. В этом случае, SOAP-сообщение, получено узлом, играющим определенную приложением роль "http://example.com/Log", и этот узел должен быть также способен исполнять определенную SOAP роль "next". Сказанное справедливо и для узла-конечного получателя сообщения, так как он, очевидно (и полностью), также исполняет роль "next" в силу того, что является следующим на пути сообщения.

Третий заголовочный блок в примере 7a, `aThirdBlock`, не имеет атрибута `env:role`. Он адресован SOAP-узлу, исполняющему роль "ultimateReceiver". Роль "ultimateReceiver" (эта роль может быть объявлена явно или подразумеваться, если в заголовочном блоке атрибут `env:role` отсутствует) исполняется SOAP-узлом, являющимся конечным получателем SOAP-сообщения. Отсутствие атрибута `env:role` в заголовочном блоке `aThirdBlock` означает, что этот заголовочный блок адресован SOAP-узлу, исполняющего роль "ultimateReceiver".

Необходимо отметить, что элемент `env:Body` не имеет атрибута `env:role`. Элемент тела сообщения *всегда* адресован SOAP-узлу, исполняющему роль "ultimateReceiver". В этом смысле элемент тела сообщения, как и заголовочный блок, адресован конечному получателю, но отличен от него тем, что проходит сквозь SOAP-узлы (как правило, SOAP-посредники) неизмененным, если они исполняют роль иную, нежели роль конечного получателя. SOAP не устанавливает какой-либо структуры элемента `env:Body`. Существует лишь рекомендация, что любой субэлемент должен удовлетворять пространству имен XML. Некоторые приложения, например, в примере 1, могут выбирать каким образом организовывать субэлементы `env:Body` в блоки, однако этот вопрос не имеет отношения к модели обработки SOAP.

Другая отличительная роль элемента `env:Body`, - роль контейнера, куда помещается информация о SOAP-специфичных ошибках, т. е. об ошибках, возникающих в случае неудачного завершения обработки элементов SOAP-сообщения - была описана ранее в разделе 2.3.

Если заголовочный элемент имеет стандартизованный атрибут `env:role` со значением "none", ни один SOAP-узел не должен обрабатывать содержимое этого элемента, хотя может его просматривать, если это данные, ссылающиеся на другой заголовочный элемент, адресованный определенному SOAP-узлу.

Если атрибут `env:role` имеет пустое значение, т. е. `env:role=""`, это означает, что соответствующий идентифицирующий роль URI является исходным для SOAP-сообщения. SOAP Версия 1.2 не определяет исходный URI SOAP-сообщения, однако ссылается на механизмы получения исходного URI, определенные в [XMLBase] которые могут быть использованы для работы с любыми относительными URI как с абсолютными. Подобный механизм существует для протокольной привязки с целью определения исходного URI по ссылке на протокол, инкапсулирующий SOAP-сообщение для передачи. (В случае передачи SOAP-сообщений с помощью HTTP, SOAP Часть 2 Раздел 7.1.2 допределяет исходный URI как URI-запрос HTTP-запроса либо как значение заголовка HTTP Content-Location.)

Приведенная ниже таблица суммирует информацию о допустимых стандартизованных ролях, которые могут исполняться SOAP-узлами. ("Да" и "Нет" означают, что узлы соответственно могут или не могут исполнять указанную роль.)

Роль	отсутствует	"none"	"next"	"ultimateReceiver"
Узел				
отправитель	неприменима	неприменима	неприменима	неприменима
посредник	нет	нет	да	нет
конечный получатель	да	нет	да	да

Атрибут "mustUnderstand"

Пример 7b дополняет предыдущий пример, вводя другой (необязательный) атрибут заголовочных блоков - атрибут `env:mustUnderstand`.

Пример 7b

```
<?xml version="1.0" ?>
 <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
   <env:Header>
     <p:oneBlock xmlns:p="http://example.com"
           env:role="http://example.com/Log"
               env:mustUnderstand="true">
     ...
     ...
     </p:oneBlock>
     <q:anotherBlock xmlns:q="http://example.com"
       env:role="http://www.w3.org/2003/05/soap-envelope/role/next">
```

```
        ...
        ...
        </q:anotherBlock>
        <r:aThirdBlock xmlns:r="http://example.com">
        ...
        ...
        </r:aThirdBlock>
      </env:Header>
      <env:Body >
        ...
        ...
      </env:Body>
    </env:Envelope>
```

SOAP-сообщение, демонстрирующее многообразие заголовочных блоков, один из которых обязателен для обработки

После того, как SOAP-узел с помощью атрибута `env:role` верно идентифицирует все адресованные ему заголовочные блоки (и, возможно, также тело сообщения), дальнейшие действия по обработке, которые должны быть предприняты, определяются дополнительным атрибутом `env:mustUnderstand` в заголовочных элементах. Этот необязательный атрибут может включаться в заголовочные блоки SOAP-сообщения для обеспечения уверенности в том, что SOAP-узлы не проигнорируют важные для приложения в целом заголовочные блоки. Если атрибут `env:mustUnderstand` имеет значение "true", это означает, что SOAP-узел, которому адресовано сообщение, **должен** обработать блок согласно спецификации этого блока. Фактически обработка SOAP-сообщения даже не должна начинаться, пока узел не идентифицирует и не "поймет" все обязательные блоки, адресованные ему. Под "пониманием" заголовка подразумевается, что узел готов производить действия, описанные в спецификации этого блока. (Спецификации заголовочных блоков не являются предметом рассмотрения спецификации SOAP.)

В примере 7b заголовочный блок `oneBlock` содержит `env:mustUnderstand`, с установленным значением "true". Это означает, что блок `oneBlock` является обязательным для обработки, если SOAP-узел исполняет роль, определенную "http://example.com/Log". Два других заголовочных блока не имеют атрибута `env:mustUnderstand`, и поэтому SOAP-узел, которому эти блоки адресованы, может их не обрабатывать. (Здесь предполагается, что спецификации блоков позволяют это.)

Значение "true" атрибута `env:mustUnderstand` означает, что SOAP-узел должен либо обработать заголовок в соответствии с семантикой, описанной в спецификации этого заголовка, либо сгенерировать SOAP-ошибку обработки. Обработка заголовка может включать в себя удаление заголовка из любого SOAP-сообщения, повторное помещение заголовка с тем же либо измененным значением или помещение нового заголовка. Невозможность обработки обязательного заголовка влечет прекращение дальнейшей обработки SOAP-сообщения и генерацию SOAP-ошибки. При этом сообщение дальше не передается.

Элемент `env:Body` не имеет атрибута `env:mustUnderstand` , однако *должен* быть обработан конечным получателем. В примере 7b конечный получатель сообщения - SOAP-узел, исполняющий роль "ultimateReceiver" - должен обработать `env:Body` и может обработать заголовочный блок `aThirdBlock`. Он также может обработать заголовочный блок `anotherBlock`, так как этот блок адресован конечному получателю сообщения (в рамках роли "next"), но не обязан делать это, если спецификации обработки блоков не требуют этого. (Для

того, чтобы блок `anotherBlock` был обработан следующим получателем необходимо, чтобы его спецификация содержала `env:mustUnderstand`="true".)

Роль (роли), которые SOAP-узлы исполняют во время обработки SOAP-сообщения, могут определяться многими факторами. Роль может быть известна априори (т. е. заранее) либо может быть установлена некоторыми вспомогательными средствами. Узел также может просмотреть все составляющие полученного сообщения перед его обработкой для определения роли, которую он будет исполнять. Интересная ситуация может возникать, когда SOAP-узел во время обработки сообщения выясняет, что он должен исполнять еще и некоторые дополнительные роли. Не имеет значения, в какой момент это происходит, внешне должно казаться, будто узел продолжает свои действия согласно модели обработки. Все должно выглядеть так, будто эти дополнительные роли были известны узлу с самого начала обработки сообщения. Внешним проявлением такого поведения узла должна быть проверка `env:mustUnderstand` всех заголовков, могущих содержать дополнительные роли, выполненная перед началом какой-либо обработки сообщения. В случае, если SOAP-узел берется исполнять эти дополнительные роли, необходимо быть уверенным, что он готов делать все необходимое согласно спецификациям этих ролей.

Приведенная ниже таблица показывает то, как различные действия по обработке заголовочного блока узлом, которому он был адресован (посредством атрибута `env:role`), определяются атрибутом `env:mustUnderstand`.

Узел	посредник	конечный получатель
mustUnderstand		
"true"	должен обработать	должен обработать
"false"	может обработать	может обработать
отсутствует	может обработать	может обработать

В качестве результата обработки SOAP-сообщения, SOAP-узел может сгенерировать одну единственную SOAP-ошибку, если обработка сообщения завершилась неудачно либо, в зависимости от приложения, сгенерировать дополнительные SOAP-сообщения для отправки другим SOAP-узлам. SOAP Часть 1 Раздел 5.4 описывает структуру сообщения об ошибке в то время как модель обработки SOAP определяет условия, при выполнении которых оно генерируется. Как было показано ранее в разделе 2.3, SOAP-ошибка это SOAP-сообщение со стандартизованным субэлементом `env:Body` имеющим название `env:Fault`.

SOAP различает генерацию ошибки и гарантирование того, что ошибка была возвращена узлу, создавшего сообщение, либо другому узлу, которому эта информация может быть полезна. Однако то, может ли сгенерированная ошибка нужным образом быть передана, зависит от привязки к нижележащему протоколу, выбранной для обмена SOAP-сообщениями. Спецификация не описывает, что происходит, если ошибки были сгенерированы во время передачи сообщений лишь в один конец. Только одна нормативная привязка к нижележащему протоколу, привязка SOAP к HTTP, предлагает HTTP-отклик как средство сообщения об ошибке в приходящем SOAP-сообщении. (См. раздел 4 для более подробной информации о протокольных SOAP-привязках.)

Атрибут "relay"

SOAP Версия 1.2 определяет еще один необязательный атрибут для заголовочных блоков - `env:relay` , принадлежащий типу `xs:boolean`, который показывает, что заголовочный блок, адресованный SOAP-посреднику, должен быть передан дальше, если он *не* был обработан.

Необходимо отметить, что если заголовочный блок обработан, правила обработки SOAP (см. SOAP Часть 1 Раздел 2.7.2) требуют, чтобы он был удален из уходящего сообщения. (Он может, однако, быть вставлен заново, с неизмененным либо измененным содержимым, если в результате обработки других заголовочных блоков стало ясно, что заголовочный блок должен быть оставлен в передаваемом сообщении.) По умолчанию, *необработанный* заголовочный блок, предназначенный роли, которую исполняет SOAP-посредником, должен быть удален перед отправкой сообщения.

Причиной выбора таких действий по умолчанию являются соображения безопасности: гарантируется, что SOAP-посредник не знает о дальнейшей судьбе заголовочного блока после того, как сообщение прошло этот узел. Тем самым предоставляется некоторая дополнительная функция, особенно полезная в случае, если SOAP-посредник не обработал заголовочный блок, скорее всего, вследствие того, что не "понял" его. Определенные заголовочные блоки представляют подобные, зависящие от каждого отрезка пути сообщения, функции потому, что нет смысла гонять необработанное сообщение из конца в конец. Поскольку посредник может не быть в состоянии определить это, он может подумать, что было бы безопаснее, если бы необработанные заголовочные блоки были удалены из сообщения до того, как оно будет отправлено дальше.

Однако бывают случаи, когда проектировщик приложений хотел бы ввести новую функцию, объявляемую с помощью заголовочного блока SOAP, адресованного *любому* посреднику, который может быть встречен на пути SOAP-сообщения. Подобный заголовочный блок может быть полезен тем посредникам, которые "поймут" его, но будет проигнорирован и передан дальше теми, которые его не "поймут". По крайней мере, на некоторых начальных SOAP-узлах на пути сообщения может быть установлено соответствующее программное обеспечение для обработки заголовочного блока с новой функцией, однако оно может оказаться не на всех узлах. Поэтому включение в такой заголовочный блок атрибута `env:mustUnderstand` = "false" необходимо для того, чтобы посредники, не имеющие средств обработки подобного заголовочного блока, не генерировали ошибку. Включение в заголовочный блок дополнительного атрибута `env:relay` со значением "true", позволяет посреднику передать адресованный ему заголовочный блок дальше в том случае, если он не может его обработать.

Адресация заголовочного блока роли "next" наряду с атрибутом `env:relay` , имеющим значение "true", всегда может служить гарантией того, что каждый посредник имеет возможность просмотреть заголовок, потому что одним из возможных вариантов использования роли "next" является передача посредством заголовочных блоков информации, которая должна быть сохранена неизменной на всем пути SOAP-сообщения. Конечно, проектировщик приложения всегда может определить свою роль, позволяющую адресацию заголовочных блоков специальным посредникам, готовым эту роль исполнять. Вследствие этого, ограничений на использование атрибута `env:relay` совместно с любой ролью не накладывается (конечно, кроме ролей "none" и "ultimateReceiver", для которых этот атрибут не имеет смысла).

Пример 7c демонстрирует использование атрибута `env:relay`.

Пример 7c

```
<?xml version="1.0" ?>
 <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
   <env:Header>
     <p:oneBlock xmlns:p="http://example.com"
           env:role="http://example.com/Log"
               env:mustUnderstand="true">
     ...
     ...
     </p:oneBlock>
     <q:anotherBlock xmlns:q="http://example.com"
       env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
       env:relay="true">
     ...
     ...
     </q:anotherBlock>
     <r:aThirdBlock xmlns:r="http://example.com">
     ...
     ...
     </r:aThirdBlock>
   </env:Header>
   <env:Body >
     ...
     ...
   </env:Body>
 </env:Envelope>
```

SOAP-сообщение, показывающее разнообразие заголовочных блоков, один из которых должен быть передан будучи не обработанным

Заголовочный блок `q:anotherBlock`, адресованный "следующему" узлу "next" на пути сообщения, имеет дополнительный атрибут `env:relay`="true". SOAP-узел, который получает это сообщение, сможет обработать этот заголовочный блок, если "поймет" его. Если узел обработает заголовочный блок `q:anotherBlock` , правила обработки требуют, чтобы этот заголовочный блок был удален из сообщения до его отправления дальше. В то же время, если SOAP-узел проигнорирует `q:anotherBlock` (что он вполне может сделать, так как этот заголовочный блок не является обязательным для обработки вследствие отсутствия атрибута `env:mustUnderstand`), в этом случае он должен отправить его дальше по пути следования сообщения.

Обработка заголовочного блока `p:oneBlock` является обязательной и правила обработки SOAP требуют, чтобы он не отправлялся дальше, пока результаты обработки какого-либо другого заголовочного блока требуют его присутствия в уходящем сообщении. Заголовочный блок `r:aThirdBlock` не имеет атрибута `env:relay`, что эквивалентно присутствию этого атрибута со значением `env:relay`="false". Следовательно, этот заголовок не передается дальше до тех пор, пока не будет обработан.

SOAP 1.2 Часть 1 Таблица 3 сводит воедино условия, определяющие когда SOAP-посреднику, исполняющему заданную роль, разрешается передавать необработанные заголовочные блоки дальше.

Использование различных протокольных привязок

Обмен SOAP-сообщениями может быть организован с помощью разнообразных "нижележащих" протоколов, включая протоколы уровня приложения. Спецификация метода передачи SOAP-сообщения от одного SOAP-узла другому с использованием нижележащего протокола называется привязкой протокола SOAP. [SOAP Часть 1] определяет SOAP-сообщение в форме информационного множества XML [XML Infoset], т. е. в терминах элементов и атрибутов информационных параграфов абстрактного "документа", называемого `env:Envelope` (см. SOAP Часть 1 Раздел 5). Любое представление информационного множества SOAP `env:Envelope` конкретизируется протокольной привязкой, одна из задач которой - дать сериализованное представление информационного множества, которое может быть передано следующему SOAP-узлу на пути сообщения таким образом, чтобы была возможность перестроить информационное множество без потери информации. В типичных примерах SOAP-сообщений и во всех примерах этого учебника продемонстрированная сериализация является корректно построенным XML-документом [XML 1.0]. Однако, могут быть и другие протокольные привязки - например протокольная привязка между двумя SOAP-узлами поверх интерфейса ограниченной пропускной способности - где также может быть выбрана альтернативная сжатая сериализация того же информационного множества. Другая привязка, выбранная для другой цели, может предоставлять сериализацию, являющуюся зашифрованной структурой и описывающей то же информационное множество.

В дополнение к предоставлению конкретной реализации информационного множества SOAP между смежными SOAP-узлами на пути SOAP-сообщения, протокольная привязка предоставляет механизмы поддержки функций, необходимых SOAP-приложению. Функция является частью спецификации определенной функциональности, предоставляемой привязкой. Описание функции идентифицируется посредством URI, так что все приложения, ссылающиеся на нее, гарантированно используют одну и ту же семантику. Например, типичный сценарий использования может потребовать реализации множества одновременно действующих обменов типа "запрос-отклик" между смежными SOAP-узлами, в этом случае необходима функция корреляции запроса и отклика. Среди других примеров - функция "шифрованного канала", функция "канала надежной доставки" или определенная функция шаблона обмена SOAP-сообщениями.

Спецификация SOAP-привязки (см. SOAP Часть 1 Раздел 4) помимо других вопросов описывает, какие функции она предоставляет (если она их предоставляет). Некоторые функции могут быть предоставлены непосредственно нижележащим протоколом. Если функция не доступна с помощью привязки, она может быть реализована в рамках SOAP-оболочки с помощью заголовочных блоков SOAP. Спецификация функции, реализованной с помощью заголовочных блоков SOAP, называется SOAP-модулем.

Например, если обмен SOAP-сообщениями происходит напрямую посредством протокола датаграмм UDP, очевидно, что функция, дающая возможность корреляции сообщений, должна быть где-либо реализована - либо непосредственно приложением, либо, что более вероятно, частью информационных множеств SOAP, участвующих в обмене. В последнем случае функция корреляции сообщений в рамках SOAP-оболочки имеет вид, специфичный для конкретной привязки, подобно заголовочному блоку SOAP, определенному в модуле "Корреляция типа запрос-отклик", который в свою очередь идентифицируется посредством URI. Однако, если обмен информационными множествами SOAP происходит с использованием нижележащего протокола, он, очевидно, принадлежит к типу "запрос/отклик". Поэтому приложение могло бы полностью унаследовать эту предоставляемую привязкой функцию, и в ее дальнейшей поддержке на уровне приложения либо на уровне SOAP не было

бы необходимости. (Фактически, HTTP-привязка для SOAP имеет преимущество перед той же функцией HTTP.)

SOAP-сообщение может передаваться несколькими прыжками между отправителем и конечным получателем, где каждый прыжок может иметь свою протокольную привязку. Другими словами, функция (например, функция корреляции сообщений, обеспечения надежности передачи и т. д.), поддерживаемая протокольной привязкой на одном отрезке пути сообщения, может не поддерживаться на другом. SOAP сам по себе не предоставляет какого-либо механизма сокрытия функциональных различий нижележащих протоколов. Однако, любая функция, которая требуется конкретному приложению, но не может быть реализована имеющейся инфраструктурой на *ожидаемом* отрезке пути сообщения, может быть заменена передачей части информационного множества SOAP-сообщения, т. е. специфицированным в каком-либо модуле заголовочным блоком SOAP.

Таким образом, видно, что существует ряд вопросов, которые проектировщик приложения должен решить с целью соблюдения определенной семантики приложения. Включая поиск вариантов, позволяющих использовать преимущества присущей нижележащему протоколу функциональности, которая доступна в выбранной среде.SOAP Часть 1 Раздел 4.2 дает общее описание того, как использующие SOAP приложения могут для соблюдения определенной семантики приложения использовать функциональность, предоставляемую протокольной привязкой. В дальнейшем планируется написать руководства по разработке спецификаций допустимых взаимодействий в рамках выбранной протокольной привязки.

Помимо других вопросов, спецификация привязки должна определять также шаблон(ы) обмена сообщениями, которые она поддерживает. [SOAP Часть 2] определяет два таких шаблона обмена сообщениями: шаблон обмена SOAP-сообщениями типа "запрос-отклик", где одно SOAP-сообщение пересылается между двумя смежными узлами в обоих направлениях, и шаблон обмена SOAP-сообщениями типа "отклик", который состоит из не использующего SOAP сообщения-запроса и следующего за ним SOAP-сообщения, реализованного как часть отклика.

[SOAP Часть 2] также предлагает проектировщику приложений общую функцию, называемую Web-методом SOAP, которая позволяет приложениям полностью контролировать выбор так называемого "Web-метода" - одного из GET, POST, PUT, DELETE, чья семантика определена в спецификациях HTTP [HTTP 1.1]. Web-метод может использоваться поверх привязки. Эта функция гарантирует, что SOAP-приложения могут взаимодействовать поверх привязки образом, совместимым с архитектурными принципами World Wide Web. (Говоря кратко, простота и масштабируемость Web существует в значительной степени благодаря существованию множества "общеупотребительных" методов (GET, POST, PUT, DELETE), которые могут быть использованы для взаимодействия с любым ресурсом Web посредством URI.) Функция SOAP Web-метод поддерживается HTTP-привязкой SOAP, хотя, в принципе, она доступна и для всех других привязок SOAP к нижележащим протоколам.

SOAP Часть 2 Раздел 7 специфицирует одну стандартизованную протокольную привязку с помощью структуры привязки [SOAP Часть 1], определяющую как SOAP может быть использован совместно с HTTP в качестве нижележащего протокола. SOAP Версия 1.2 ограничивается определением HTTP-привязки, позволяющей использовать только метод POST в сочетании с шаблоном обмена сообщениями типа "запрос-отклик" и метод GET в сочетании с шаблоном обмена SOAP-сообщениями типа "отклик". В недалеком будущем другие спецификации смогут описать HTTP-привязку SOAP либо привязки к другим транспортным

протоколам. Эти спецификации позволят использовать другие Web-методы (т. е. PUT, DELETE).

Следующие разделы демонстрируют примеры двух привязок SOAP к нижележащим протоколам, а именно к [HTTP 1.1] и email. Необходимо снова подчеркнуть, что существует только одна нормативная привязка SOAP 1.2 к HTTP [HTTP 1.1]. Примеры раздела 4.2, показывающие email в качестве транспортного механизма SOAP, тем не менее говорят о том, что для передачи SOAP-сообщений возможно использование и других транспортных протоколов, хотя они в настоящее время и не стандартизованы. Заметка W3C [E-mail-привязка SOAP] описывает реализацию структуры протокольной привязки [SOAP Часть 1] с помощью описания экспериментальной привязки SOAP к email-транспорту, в особенности [RFC 2822] - транспорта передачи сообщений.

HTTP-привязка SOAP

HTTP имеет хорошо известную модель взаимодействия и шаблон обмена сообщениями. Клиент идентифицирует сервер по URI, подсоединяется к нему с помощью TCP/IP сети, отсылает HTTP-сообщение-запрос и получает HTTP-сообщение-отклик по тому же TCP-соединению. HTTP полностью коррелирует сообщение-запрос и соответствующий ему сообщение-отклик, поэтому приложение, использующее эту привязку, может реализовать корреляцию между отправленным в теле HTTP-запроса SOAP-сообщением и SOAP-сообщением, возвращенным в качестве HTTP-отклика. Подобным образом, HTTP идентифицирует конечный сервер по URI, URI-запросу, который может также служить идентификатором SOAP-узла сервера.

HTTP могут использовать многочисленные посредники между начальным клиентом и сервером, инициировавшим обмен сообщениями, идентифицируемые по URI-запросу, в этом случае модель запрос/отклик является по существу сериями таких пар. Необходимо отметить, что HTTP-посредник и SOAP-посредник не являются тождественными понятиями.

HTTP-привязка [SOAP Часть 2] использует функцию SOAP Web-метода, позволяющую приложениям выбирать один из так называемых Web-методов - GET или POST - чтобы использовать для обмена сообщениями с помощью HTTP. Кроме того, она использует два шаблона обмена сообщениями, которые дают приложениям два способа обмена SOAP-сообщениями посредством HTTP: 1) использование HTTP-метода POST для передачи SOAP-сообщений в теле HTTP-запроса и HTTP-отклика, и 2) использование HTTP-метода GET в HTTP-запросе для возвращения SOAP-сообщения в теле HTTP-отклика. Первый шаблон использования является HTTP-реализацией функции привязки - шаблона обмена SOAP-сообщениями типа "запрос-отклик", второй - шаблона обмена SOAP-сообщениями типа "отклик".

Цель разработки этих двух способов - реализовать две парадигмы взаимодействия, одинаково хорошо подходящие для World Wide Web. Первый тип взаимодействия позволяет использовать данные в теле HTTP-метода POST для создания или изменения состояния ресурса, идентифицируемого по URI, в соответствии с которым направлен HTTP-запрос. Второй тип шаблона взаимодействия дает возможность использовать HTTP-запрос GET для получения представления о ресурсе без какого-либо изменения его состояния. В первом случае касающийся SOAP аспект вопроса состоит в том, что тело HTTP-запроса POST является SOAP-сообщением, которое кроме того, что должно быть обработано (согласно

модели обработки SOAP) в соответствии со специфичной для приложения обработкой, должно также соответствовать семантике POST. Во втором случае типичной реализацией является получение представления запрашиваемого ресурса в виде SOAP-сообщения, а не в виде HTML- или XML-документа. То есть HTTP-заголовок типа содержимого сообщения идентифицирует это как медиа-тип "application/soap+xml". Вероятно, найдутся создатели Web-ресурсов, которые впоследствии обнаружат, что такие ресурсы проще всего найти и сделать доступными в виде SOAP-сообщений. Надо отметить, однако, что ресурсы могут быть доступны в виде различных представлений; желаемое или предпочтительное представление может быть получено путем опроса приложения с помощью HTTP-заголовка Accept.

Следующим аспектом HTTP-привязки SOAP является проблема определения, какой из вышеуказанных двух шаблонов обмена сообщениями использовать. [SOAP Часть 2] содержит руководство, описывающее в каких обстоятельствах приложения могут использовать тот или иной определенный шаблон обмена сообщениями. Шаблон обмена SOAP-сообщениями с использованием HTTP-метода GET, используется в том случае, если приложение использует этот шаблон только для поиска информации, а сам ресурс в результате взаимодействия остается "нетронутым". Подобные взаимодействия трактуются в спецификации HTTP как безопасные и идемпотентные. Поскольку использование SOAP HTTP-метода GET не разрешено для SOAP-сообщения, содержащегося в запросе, приложения, которым необходим доступ к функциям взаимодействия с внешними объектами, которые поддерживаются только специфичным для привязки выражением внутри информационного множества SOAP (то есть в качестве заголовочных SOAP-блоков), не могут использовать этот шаблон обмена сообщениями. Необходимо отметить, что HTTP-метод POST привязки может применяться во всех случаях.

Следующие подразделы демонстрируют примеры использования этих двух шаблонов обмена сообщениями, определенных для HTTP-привязки.

Использование в SOAP HTTP-метода GET

Использование HTTP-привязки совместно с шаблоном обмена SOAP-сообщениями типа "отклик" ограничивается HTTP-методом GET. Это означает, что откликом на HTTP-запрос GET запрашивающего SOAP-узла будет являться SOAP-сообщение в HTTP-отклике.

Пример 8a показывает HTTP-метод GET, направленный приложением путешественника (продолжая сценария бронирования путешествия) по URI `http://travelcompany.example.org/reservations?code=FT35ZBQ` , где можно увидеть маршрут путешествия. (Как эта URL стала доступна, можно увидеть в примере 5a.)

Пример 8a

```
GET /travelcompany.example.org/reservations?code=FT35ZBQ  HTTP/1.1
Host: travelcompany.example.org
Accept: text/html;q=0.5, application/soap+xml
```

HTTP-запрос GET

HTTP-заголовок Accept используется для индикации предпочитаемого представления запрашиваемого ресурса. В этом примере медиа-тип "application/soap+xml" более предпочтителен для интерпретации компьютером-клиентом, нежели медиа-тип "text/html" - для интерпретации браузером-клиентом человека.

Пример 8b показывает HTTP-отклик на запрос GET в примере 8a. Тело HTTP-отклика содержит SOAP-сообщение, показывающее детали путешествия. Обсуждение содержания SOAP-сообщения отложено до раздела 5.2, поскольку это не имеет отношения к пониманию использования метода GET HTTP-привязки.

Пример 8b

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
       env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
           env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
   <m:dateAndTime>2001-11-30T16:25:00.000-05:00</m:dateAndTime>
  </m:reservation>
 </env:Header>
 <env:Body>
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:x="http://travelcompany.example.org/vocab#"
     env:encodingStyle="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   <x:ReservationRequest
 rdf:about="http://travelcompany.example.org/reservations?code=FT35ZBQ">
      <x:passenger>Eke Jygvan Шyvind</x:passenger>
      <x:outbound>
       <x:TravelRequest>
        <x:to>LAX</x:to>
        <x:from>LGA</x:from>
        <x:date>2001-12-14</x:date>
       </x:TravelRequest>
      </x:outbound>
      <x:return>
       <x:TravelRequest>
        <x:to>JFK</x:to>
        <x:from>LAX</x:from>
        <x:date>2001-12-20</x:date>
       </x:TravelRequest>
      </x:return>
    </x:ReservationRequest>
  </rdf:RDF>
 </env:Body>
</env:Envelope>
```

SOAP-сообщение, полученное в качестве отклика на HTTP-запрос GET примера 8a

Необходимо отметить, что детали брони могут быть получены и в виде (X)HTML-документа, однако вышеприведенным примером показан случай, когда приложение, осуществляющее бронирование, возвращает состояние ресурса (бронирование) в ориентированном на представление данных виде (в виде SOAP-сообщения), который может быть обработан автоматически, в отличие от (X)HTML- документа, который может быть обработан браузером. Действительно, как правило, принимающим приложением является не браузер.

Также, как показано в примере, использование SOAP в теле HTTP-отклика дает возможность реализации специфичных для приложений функций с помощью SOAP-заголовков. Посредством SOAP приложение получает практичную и непротиворечивую структуру, а также модель обработки для реализации таких функций.

Использование в SOAP HTTP-метода POST

Использование HTTP-привязки совместно с шаблоном обмена SOAP-сообщениями типа "запрос-отклик" ограничивается использованием HTTP-метода POST. Необходимо отметить, что использование этого шаблона обмена сообщениями в HTTP-привязке доступно всем приложениям, используют ли они обмен общими XML-данными или обмен RPC, инкапсулированными в SOAP-сообщения (как в следующих примерах).

Примеры 9 и 10 показывают пример HTTP-привязки, использующей шаблон обмена сообщениями типа "запрос-отклик" с тем же набором действий, что в, примере 4 и примере 5а, а именно передачу RPC и ее возврат в теле SOAP-сообщения соответственно. Примеры и содержание этого раздела сконцентрированы исключительно на HTTP-заголовках и их роли.

Пример 9

```
POST /Reservations HTTP/1.1
Host: travelcompany.example.org
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
 <env:Header>
   <t:transaction
           xmlns:t="http://thirdparty.example.org/transaction"
           env:encodingStyle="http://example.com/encoding"
           env:mustUnderstand="true" >5</t:transaction>
 </env:Header>
 <env:Body>
  <m:chargeReservation
     env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
          xmlns:m="http://travelcompany.example.org/">
   <m:reservation xmlns:m="http://travelcompany.example.org/reservation">
    <m:code>FT35ZBQ</m:code>
   </m:reservation>
    <o:creditCard xmlns:o="http://mycompany.example.com/financial">
     <n:name xmlns:n="http://mycompany.example.com/employees">
           Eke Jygvan Шyvind
     </n:name>
     <o:number>123456789099999</o:number>
     <o:expiration>2005-02</o:expiration>
    </o:creditCard>
   </m:chargeReservation>
  </env:Body>
</env:Envelope>
```

RPC примера 4 , переданный посредством HTTP-запроса POST

Пример 9 демонстрирует RPC-запрос, направленный приложению путешествий. SOAP-сообщение отправляется в теле HTTP-метода POST по URI, идентифицирующего ресурс

"Reservations" на сервере travelcompany.example.org. При использовании HTTP URI-запрос указывает на ресурс, которому был "отправлен" вызов. Кроме требования, что URI должен быть корректным, SOAP не накладывает больше никаких формальных ограничений на вид URI-запроса (см. [RFC 2396] для получения дополнительной информации по URI). Однако, одним из архитектурных принципов Web является требование, чтобы все важные ресурсы были идентифицированы URI. Данный факт позволяет предположить, что правильно построенные с точки зрения архитектуры Web SOAP-сервисы будут представлять собой ряд ресурсов, каждый со своим собственным URI. Действительно, многие подобные ресурсы, скорее всего, будут создаваться динамически во время работы сервиса, такого как, например, сервиса продажи билетов, показанного в примере. Так, правильно организованное с архитектурной точки зрения приложение продажи билетов должно иметь различные URI для каждой конкретной брони, и SOAP-запросы, предназначенные для получения или манипулирования этими бронями, будут направлены по их URI, а не по единому URI "Reservations", как это показано в примере 9. Пример 13 раздела 4.1.3 показывает предпочтительный способ обращения к ресурсам, таким как приложение бронирования путешествия. Поэтому отложим до раздела 4.1.3 дальнейшее обсуждение совместимого с архитектурными принципами Web использования SOAP/HTTP.

Если SOAP-сообщение помещается в тело HTTP-сообщения, необходимо выбрать "application/soap+xml" в качестве значения HTTP-заголовка Content-type. (Необязательный символьный параметр, показанный в вышеприведенном примере, может принимать значения "utf-8" или "utf-16", однако если он отсутствует, к телу HTTP-запроса применяются правила кодировки автономного XML-документа [XML 1.0].)

Пример 10 показывает отклик RPC (его детали опущены для краткости), отправленный приложением бронирования путешествий в соответствующем HTTP-отклике на запрос примера 5а. SOAP, используя HTTP-транспорт, полагается на семантику HTTP-кодов статуса для указания в HTTP информации о статусе. Например, серия кодов статуса 2xx показывает, что запрос клиента (включая SOAP-составляющую) был успешно получен, понят, акцептован и т. д.

Пример 10

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
 <env:Header>
       ...
       ...
 </env:Header>
 <env:Body>
       ...
       ...
 </env:Body>
</env:Envelope>
```

Отклик на RPC примера 5а, вложенный в HTTP-отклик, показывающий успешное завершение

При возникновении ошибки обработки запроса, спецификация HTTP-привязки требует использования HTTP 500 "Internal Server Error" с вложенным SOAP-сообщением, содержащим SOAP-ошибку, указывающую ошибку обработки на стороне сервера.

Пример 11 являет собой то же сообщение об ошибке, что и пример 6а, однако на этот раз в него добавлены HTTP-заголовки.

Пример 11

```
HTTP/1.1 500 Internal Server Error
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>
    <env:Fault>
     <env:Code>
       <env:Value>env:Sender</env:Value>
       <env:Subcode>
        <env:Value>rpc:BadArguments</env:Value>
       </env:Subcode>
     </env:Code>
     <env:Reason>
      <env:Text xml:lang="en-US">Processing error</env:Text>
      <env:Text xml:lang="cs">Chyba zpracovбnн</env:Text>
     </env:Reason>
     <env:Detail>
      <e:myFaultDetails
        xmlns:e="http://travelcompany.example.org/faults" >
        <e:message>Name does not match card number</e:message>
        <e:errorcode>999</e:errorcode>
      </e:myFaultDetails>
     </env:Detail>
   </env:Fault>
 </env:Body>
</env:Envelope>
```

Пример SOAP-сообщения в HTTP-отклике, указывающего ошибку обработки SOAP-элемента Body примера 4

SOAP Часть 2 Таблица 16 дает детальное описание поведения при обработке различных возможных кодов HTTP-откликов, например, 2xx (успешный), 3xx (перенаправление), 4xx (ошибка на стороне клиента) и 5xx (ошибка на стороне сервера).

Использование SOAP в соответствии с архитектурными принципами Web

Одной из наиболее важных концепций World Wide Web является идентификация ресурса с помощью URI. SOAP-сервисы, которые используют HTTP-привязку и взаимодействуют с другим программным обеспечением Web, должны использовать URI для указания на ресурсы. Например, очень важным - несомненно, преобладающим - использованием World Wide Web является поиск информации, где доступ к ресурсу, идентифицированному с помощью URI, осуществляется при помощи HTTP-запроса GET без оказания какого-либо влияния на сам ресурс. (В терминах HTTP это называется безопасным и

идемпотентным методом.) Ключевым моментом здесь является то, что владелец ресурса делает доступным его URI, который пользователи могут получить с помощью запроса GET.

Существует много реализаций, в которых SOAP-сообщения спроектированы сугубо с целью поиска информации, когда запрашивается состояние некоторого ресурса (или объекта в терминах программирования), в противовес реализациям, которые выполняют некоторые манипуляции с ресурсом. В таких реализациях использование тела SOAP-сообщения с элементом, представляющим объект, о котором идет речь, для передачи запроса о состоянии ресурса рассматривается как противоречащее духу Web, потому что запрашиваемый ресурс не идентифицирован посредством URI-запроса HTTP GET. (В некоторых SOAP/RPC-реализациях HTTP URI-запрос часто сам по себе не является идентификатором ресурса, однако он представляет собой некоего посредника, который, просмотрев SOAP-сообщение, должен идентифицировать запрашиваемый ресурс.)

Для подчеркивания необходимых изменений пример 12a показывает способ, который **не** рекомендуется для осуществления безопасного поиска информации в Web. Этот пример RPC, передаваемого SOAP-сообщением, снова использует тему бронирования путешествия, где запрос на получение маршрута конкретной брони идентифицирован одним из параметров RPC, `reservationCode`. (Для целей этой дискуссии предполагается, что приложение, использующее этот RPC запрос, не нуждается в функциях, требующих использования SOAP-заголовков.)

Пример 12a

```
POST /Reservations HTTP/1.1
Host: travelcompany.example.org
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
  <env:Body>
    <m:retrieveItinerary
        env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
             xmlns:m="http://travelcompany.example.org/">
      <m:reservationCode>FT35ZBQ</m:reservationCode>
    </m:retrieveItinerary>
  </env:Body>
</env:Envelope>
```

Это представление не рекомендуется в случаях, когда осуществляется "безопасный" (т. е. не имеющий побочных эффектов) поиск информации

Необходимо отметить, что запрашиваемый ресурс не идентифицирован с помощью URI в HTTP-запросе, но, тем не менее, стал доступен после поиска внутри SOAP-оболочки. Как и в отношении других URI, это было бы невозможно сделать только средствами HTTP.

SOAP Часть 2 Раздел 4.1 дает рекомендации по реализации корректных с точки зрения архитектуры Web RPC, осуществляющих безопасный и идемпотентный поиск информации. Это делается посредством разделения аспектов метода и специфичных параметров в определении RPC, которое служит для идентификации нужных ресурсов среди других, осуществляющих иные функции. В примере 12a, запрашиваемый ресурс, идентифицирован с помощью двух признаков: первый из них является маршрутом (часть имени метода), второй же - ссылкой на специальный экземпляр (параметр метода). В подобных случаях

рекомендуется идентифицирующие ресурс признаки указывать в идентифицирующем тот же ресурс HTTP URI-запросе. Например:

`http://travelcompany.example.org/reservations/itinerary?reservationCode=FT35ZBQ`.

Кроме того, если определение RPC таково, что все части его описания метода могут быть истолкованы как идентифицирующие ресурс, целевой узел RPC может быть идентифицирован полностью посредством URI. Если при этом пользователь ресурса может гарантировать, что поисковый запрос является безопасным, в этом случае SOAP Версия 1.2 рекомендует применять Web-метод GET, и шаблон обмена SOAP-сообщениями типа "отклик" используется как описано в разделе 4.1.1. Это гарантирует, что SOAP RPC реализован в соответствии с архитектурными принципами Web. Пример 12b показывает предпочтительный способ того, как запросить SOAP-узел с целью осуществления безопасного поиска ресурса.

Пример 12b

```
GET /Reservations/itinerary?reservationCode=FT35ZBQ  HTTP/1.1
Host: travelcompany.example.org
Accept: application/soap+xml
```

Выполненное в соответствии с архитектурными принципами Web альтернативное представление RPC примера 12a

Необходимо отметить, что SOAP Версия 1.2 не определяет какой-либо алгоритм идентификации URI по определению RPC, предназначенного сугубо для поиска информации.

Необходимо, однако, заметить, что если приложение требует использования функций, имеющих только специфичное для привязки выражение в рамках информационного множества SOAP, т. е., с помощью заголовочных блоков SOAP, то приложение должно использовать HTTP-метод POST с SOAP-сообщением в теле запроса.

Это также требует использования шаблона обмена SOAP-сообщениями типа "запрос-отклик", реализованного с помощью HTTP-метода POST, если описание RPC имеет данные (параметры), которые не идентифицируют ресурс. Даже в этом случае HTTP-метод POST с SOAP-сообщением может быть реализован в соответствии с архитектурными принципами Web. Как и в случае использования GET, [SOAP Часть 2] рекомендует в общем случае включать в HTTP URI-запрос все составляющие SOAP-сообщения, служащие для идентификации ресурса, которому был отправлен запрос POST. Эти же параметры, разумеется, могут быть включены и в SOAP-элемент `env:Body` . (Параметры должны включаться в параметр Body в случае использующего SOAP RPC, поскольку они относятся к описанию процедуры/метода, ожидаемого принимающим приложением.)

Пример 13 аналогичен примеру 9, за исключением того, что HTTP URI-запрос был изменен включением кода брони `code`, служащего для идентификации ресурса (бронь, которая должна быть подтверждена и оплачена)..

Пример 13

```
POST /Reservations?code=FT35ZBQ HTTP/1.1
Host: travelcompany.example.org
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn
```

```
<?xml version='1.0'?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
 <env:Header>
   <t:transaction
           xmlns:t="http://thirdparty.example.org/transaction"
           env:encodingStyle="http://example.com/encoding"
           env:mustUnderstand="true" >5</t:transaction>
 </env:Header>
 <env:Body>
  <m:chargeReservation
      env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
         xmlns:m="http://travelcompany.example.org/">
   <m:reservation xmlns:m="http://travelcompany.example.org/reservation">
    <m:code>FT35ZBQ</m:code>
   </m:reservation>
   <o:creditCard xmlns:o="http://mycompany.example.com/financial">
    <n:name xmlns:n="http://mycompany.example.com/employees">
           Eke Jygvan Шyvind
    </n:name>
    <o:number>123456789099999</o:number>
    <o:expiration>2005-02</o:expiration>
   </o:creditCard>
  </m:chargeReservation>
 </env:Body>
</env:Envelope>
```

RPC примера 4, передаваемый в HTTP-запросе POST в соответствии с архитектурными принципами Web

В примере 13, ресурс, с которым будет осуществляться взаимодействие, идентифицирован посредством двух признаков: первый из них - то, что он является бронью (часть названия метода), а второй - специальный экземпляр брони (который является значением параметра `code` метода). Остальные параметры RPC, такие как номер кредитной карточки `creditCard`, не идентифицируют ресурс, но являются вспомогательными данными, которые должны быть обработаны ресурсом. [SOAP Часть 2] рекомендует, чтобы ресурсы, запрашиваемые посредством использующего SOAP RPC, по возможности, содержали в своем URI, идентифицирующем вызываемый RPC узел, любую подобную идентифицирующую ресурс информацию. Необходимо отметить, однако, что [SOAP Часть 2] не предлагает каких-либо алгоритмов реализации этого. Подобные алгоритмы могут быть созданы в будущем. Отметьте, однако, что все идентифицирующие ресурс элементы были сохранены, как и в примере 9, в SOAP-элементе `env:Body`.

Резюмируя, можно сказать: как видно из приведенных выше примеров, рекомендацией спецификаций SOAP является использование URI в соответствии с архитектурными принципами Web - то есть использование URI в качестве идентификаторов ресурсов. При этом неважно используется ли GET или POST.

Использование SOAP поверх Email

Разработчики приложений могут использовать также email-инфраструктуру для передачи SOAP-сообщений, причем как текст email-сообщений так и их вложения. Примеры, приведенные ниже, иллюстрируют такой метод передачи SOAP-сообщений, однако они не должны трактоваться как некие стандартные способы реализации этого метода.

Спецификации SOAP Версия 1.2 не специфицируют подобную привязку. Хотя существует *неофициальный* W3C Note [E-mail-привязка SOAP], описывающий email-привязку для SOAP. Его основная цель - продемонстрировать применение общей Структуры Протокольной Привязки SOAP, описанной в [SOAP Часть 1].

Пример 14 показывает сообщение, реализующее запрос на бронирование путешествия из примера 1, передаваемый как email-сообщение между отправляющим и принимающим mail-агентами пользователя. (Подразумевается, что отправляющий узел также умеет интерпретировать SOAP и способен обрабатывать любые SOAP-ошибки, полученные в ответ на исходящее сообщение, а также коррелировать любые входящие ответные SOAP-сообщения).

Пример 14

```
From: a.oyvind@mycompany.example.com
To: reservations@travelcompany.example.org
Subject: Travel to LA
Date: Thu, 29 Nov 2001 13:20:00 EST
Message-Id: <EE492E16A090090276D208424960C0C@mycompany.example.com>
Content-Type: application/soap+xml

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
         env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</reference>
   <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
         env:mustUnderstand="true">
   <n:name>Eke Jygvan Шyvind</n:name>
  </n:passenger>
 </env:Header>
 <env:Body>
  <p:itinerary
     xmlns:p="http://travelcompany.example.org/reservation/travel">
   <p:departure>
     <p:departing>New York</p:departing>
     <p:arriving>Los Angeles</p:arriving>
     <p:departureDate>2001-12-14</p:departureDate>
     <p:departureTime>late afternoon</p:departureTime>
     <p:seatPreference>aisle</p:seatPreference>
   </p:departure>
   <p:return>
     <p:departing>Los Angeles</p:departing>
     <p:arriving>New York</p:arriving>
     <p:departureDate>2001-12-20</p:departureDate>
     <p:departureTime>mid morning</p:departureTime>
     <p:seatPreference/>
   </p:return>
  </p:itinerary>
  <q:lodging
     xmlns:q="http://travelcompany.example.org/reservation/hotels">
   <q:preference>none</q:preference>
  </q:lodging>
 </env:Body>
```

```
    </env:Envelope>
```

SOAP-сообщение примера 1, передаваемое в SMTP-сообщении

Заголовок в примере 14 реализован в стандартном для email-сообщений виде [RFC 2822].

Хотя email характеризуется односторонним обменом сообщениями и не гарантирует их доставку, транспортные протоколы, подобные Simple Mail Transport Protocol (SMTP) (SMTP) specification [SMTP], все же предоставляют механизм извещения о доставке, который, в случае с SMTP, называется Delivery Status Notification (DSN) и Message Disposition Notification (MDN). Извещения имеют вид email-сообщений, отправленных на email-адрес, указанный в заголовке email-сообщения. Приложения, так же как и email-пользователи, могут использовать эти механизмы для получения статуса передачи email-сообщения, однако эти извещения, будучи доставлены, являются сообщениями SMTP-уровня. Разработчик приложений должен хорошо понимать возможности и ограничения этих извещений о доставке, также как и риск того, что извещение может не быть сформировано в случае успешной доставки email-сообщения.

SMTP-сообщения статуса доставки являются иными, нежели сообщения, обрабатываемые на SOAP-уровне. SOAP-отклики на содержащиеся в email-сообщении SOAP-данные, получаемые в результате, будут возвращены с помощью email-сообщения, которое может содержать (а может и не содержать) SMTP-ссылку на оригинальное email-сообщение. Использование заголовка `In-reply-to:` [RFC 2822] может помочь в достижении корреляции на SMTP-уровне, но не обязательно позволит коррелировать сообщения на SOAP-уровне.

Пример 15 is продолжает сценарий примера 2 и демонстрирует SOAP-сообщение (детали тела сообщения для краткости опущены), отправленное приложением продажи билетов приложению бронирования путешествия, и проясняющее некоторые детали бронирования, кроме уже переданных email-сообщением. В этом примере `Message-Id` оригинального email-сообщения передан в дополнительном email-заголовке `In-reply-to:`, который коррелирует email-сообщения на SMTP-уровне, однако не позволяет корреляцию на SOAP-уровне. В этом примере приложение для корреляции SOAP-сообщений использует заголовочный блок `reservation` То, каким образом эта корреляция может быть реализована, зависит от конкретного приложения и не является предметом рассмотрения SOAP.

Пример 15

```
From: reservations@travelcompany.example.org
To: a.oyvind@mycompany.example.com
Subject: Which NY airport?
Date: Thu, 29 Nov 2001 13:35:11 EST
Message-Id: <200109251753.NAA10655@travelcompany.example.org>
In-reply-to:<EE492E16A090090276D208424960C0C@mycompany.example.com>
Content-Type: application/soap+xml

<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
     env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
       env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</reference>
```

```
        <m:dateAndTime>2001-11-29T13:35:00.000-05:00</m:dateAndTime>
       </m:reservation>
       <n:passenger xmlns:n="http://mycompany.example.com/employees"
           env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
             env:mustUnderstand="true">
        <n:name>Eke Jygvan Шyvind</n:name>
       </n:passenger>
      </env:Header>
      <env:Body>
       <p:itinerary
          xmlns:p="http://travelcompany.example.org/reservation/travel">
        <p:itineraryClarifications>
           ...
           ...
        </p:itineraryClarifications>
       </p:itinerary>
      </env:Body>
     </env:Envelope>
```

SOAP-сообщение из примера 2, переданное в email-сообщении с заголовком, кореллирующим его с предыдущим сообщением.

Более сложные сценарии использования

Использование SOAP-посредников

Сценарий бронирования путешествия, используемый в этом учебнике, дает возможность раскрыть некоторые методы использования SOAP-посредников. Можно вспомнить, что данный сценарий описывает обмен запросом на бронирование путешествия между приложением бронирования путешествия и приложением продажи билетов. SOAP не специфицирует как определяется маршрут сообщения и насколько точно сообщение его придерживается. Это находится за рамками спецификации SOAP. Хотя она описывает то, как SOAP-узел должен действовать при получении SOAP-сообщения, для которого он не является конечным получателем. SOAP Версия 1.2 делит SOAP-посредники на два типа: на передающих посредников и активных посредников.

Передающий посредник является SOAP-узлом, который, основываясь на семантике заголовочного блока в полученном SOAP-сообщении или на используемом шаблоне обмена сообщениями, передает SOAP-сообщение другому SOAP-узлу. Например, обработка "маршрутизирующего" заголовочного блока, содержащего маршрут входящего SOAP-сообщения, может потребовать, чтобы SOAP-сообщение было передано дальше другому SOAP-узлу, идентифицируемого данными в этом заголовочном блоке. Формат SOAP-заголовка исходящего SOAP-сообщения, т. е. размещение вложенных или удаленных и повторно вложенных заголовочных блоков, определяется по результатам обработки на *передающем* посреднике в соответствии с семантикой обрабатываемых заголовочных блоков.

Активный посредник - SOAP-узел, производящий на основе неописанных в заголовочных блоках входящего сообщения критериев или же на основе используемого шаблона обмена сообщениями некоторую дополнительную обработку сообщения перед тем, как передать его дальше. Одним из примеров подобного вмешательства SOAP-узла может быть, например, шифрование некоторых частей SOAP-сообщения и включение информации о ключе шифра в заголовочный блок. В качестве другого примера можно привести включение некоторой дополнительной информации, содержащей отметку времени создания сообщения или его аннотацию, в новый заголовочный блок исходящего сообщения, для, например, интерпретации этой информации какими-либо последующими узлами.

Одним из механизмов, посредством которого активный посредник может описать производимые с сообщением модификации, является вставка заголовочных блоков в исходящее SOAP-сообщение. Эти заголовочные блоки могут информировать последующие SOAP-узлы, исполнение ролей которых может зависеть от получения таких блоков. Семантика этих вложенных заголовочных блоков должна также предусматривать вызов или таких же или других заголовочных блоков, которые должны быть вложены на последующих посредниках и необходимы для гарантирования дальнейшей безопасной обработки сообщения другими узлами. Например, если сообщение с заголовочными блоками удалено для шифрования, производимого вторым посредником (без раскодирования и изменения оригинальных заголовочных блоков), то информация, говорящая о факте проведения кодирования должна быть сохранена во втором передаваемом сообщении.

В следующем примере SOAP-узел, встреченный на пути сообщения между приложением бронирования путешествия и приложением продажи билетов, перехватывает сообщение, приведенное в примере 1. Примером такого SOAP-узла может служить узел протоколирования всех запросов на путешествия для их офлайнового рассмотрения в бюро

путешествий. Необходимо отметить, что заголовочные блоки `reservation` и `passenger` в том примере применимы для узла (узлов), исполняющих роль "next", означающую, что блок адресован следующему SOAP-узлу на пути сообщения. Заголовочные блоки являются обязательными (атрибут `mustUnderstand` имеет значение "true"). Это означает, что узел должен знать, что ему необходимо делать (согласно внешней спецификации семантики заголовочных блоков). Спецификация протоколирования для таких заголовочных блоков может требовать, чтобы различные детали сообщения записывались на каждом узле, который получает такое сообщение, а также чтобы сообщение передавалось далее неизмененным. (Надо отметить, что спецификации заголовочных блоков должны требовать, чтобы в исходящее сообщение были вложены одни и те же заголовочные блоки, потому что в ином случае, модель обработки SOAP будет требовать их удаления.) В этом случае SOAP-узел действует как передающий посредник.

Более сложным сценарием является сценарий, в котором полученное SOAP-сообщение исправляется образом, который не ожидался инициирующим обмен отправителем. В следующем примере предполагается, что корпоративное приложение путешествий на SOAP-посреднике присоединяет заголовочный блок к SOAP-сообщению примера 1 перед его дальнейшей передачей приложению продажи билетов - конечному получателю. Заголовочный блок содержит ограничения, накладываемые политикой совершения путешествий для данного запрошенного путешествия. Спецификация такого заголовочного блока может потребовать, чтобы конечный получатель (и только конечный получатель, что подразумевается отсутствием атрибута `role`) использовал переданную им информацию во время обработки тела сообщения.

Пример 16 демонстрирует вложение дополнительного заголовочного блока `travelPolicy`, передающим посредником, который предназначен для конечного получателя. Этот заголовочный блок содержит информацию, определяющую обработку запроса на бронирование путешествия на уровне приложения.

Пример 16

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
     env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
        env:mustUnderstand="true">
   <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</reference>
   <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
  </m:reservation>
  <n:passenger xmlns:n="http://mycompany.example.com/employees"
     env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
       env:mustUnderstand="true">
   <n:name>Eke Jygvan Шyvind</n:name>
  </n:passenger>
  <z:travelPolicy
    xmlns:z="http://mycompany.example.com/policies"
      env:mustUnderstand="true">
   <z:class>economy</z:class>
   <z:fareBasis>non-refundable<z:fareBasis>
   <z:exceptions>none</z:exceptions>
  </z:travelPolicy>
 </env:Header>
 <env:Body>
  <p:itinerary
    xmlns:p="http://travelcompany.example.org/reservation/travel">
   <p:departure>
```

```
        <p:departing>New York</p:departing>
        <p:arriving>Los Angeles</p:arriving>
        <p:departureDate>2001-12-14</p:departureDate>
        <p:departureTime>late afternoon</p:departureTime>
        <p:seatPreference>aisle</p:seatPreference>
      </p:departure>
      <p:return>
        <p:departing>Los Angeles</p:departing>
        <p:arriving>New York</p:arriving>
        <p:departureDate>2001-12-20</p:departureDate>
        <p:departureTime>mid morning</p:departureTime>
        <p:seatPreference/>
      </p:return>
    </p:itinerary>
    <q:lodging
      xmlns:q="http://travelcompany.example.org/reservation/hotels">
     <q:preference>none</q:preference>
    </q:lodging>
  </env:Body>
</env:Envelope>
```

Вид SOAP-сообщения примера 1 после того, как активный посредник вложил в него обязательный заголовок, предназначенный для конечного получателя сообщения

Использование других схем написания кода

Хотя SOAP Версия 1.2 и определяет некую схему написания кода (см. SOAP Часть 2 Раздел 3), ее использование необязательно. Из спецификации становится ясно, что для специфичных данных конкретного приложения внутри SOAP-сообщения могут использоваться и другие схемы написания кода. Для этой цели спецификация описывает атрибут `env:encodingStyle`, принадлежащий типу `xs:anyURI`, Этот атрибут служит для определения заголовочных блоков, любых дочерних элементов SOAP-элемента `env:Body`, и любых дочерних элементов элемента `env:Detail`, а также производных от них элементов. Он указывает схему сериализации вложенного содержимого или, по меньшей мере, схему сериализации для одного элемента, которой необходимо придерживаться, пока не будет встречен другой элемент, указывающий на необходимость применения другого стиля написания кода для его содержимого. Выбор значения для атрибута `env:encodingStyle` определяется конкретным приложением. Возможность взаимодействия предполагает установку атрибута "out-of-band". Если атрибут `env:encodingStyle` отсутствует, никаких предположений об используемой схеме написания кода не делается.

Использование альтернативной схемы написания кода проиллюстрировано в примере 17. Продолжая тему бронирования путешествия, этот пример показывает SOAP-сообщение, содержащее детали предстоящего путешествия. Оно было отправлено пассажиру приложением продажи билетов после того, как бронь была подтверждена. (То же сообщение было использовано в другом контексте в примере 8b.)

Пример 17

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
```

```
        <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
           env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
              env:mustUnderstand="true">
         <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
         <m:dateAndTime>2001-11-30T16:25:00.000-05:00</m:dateAndTime>
        </m:reservation>
       </env:Header>
       <env:Body>
        <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                 xmlns:x="http://travelcompany.example.org/vocab#"
          env:encodingStyle="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
         <x:ReservationRequest
       rdf:about="http://travelcompany.example.org/reservations?code=FT35ZBQ">
            <x:passenger>Eke Jygvan Шyvind</x:passenger>
            <x:outbound>
             <x:TravelRequest>
              <x:to>LAX</x:to>
              <x:from>LGA</x:from>
              <x:date>2001-12-14</x:date>
             </x:TravelRequest>
            </x:outbound>
            <x:return>
             <x:TravelRequest>
              <x:to>JFK</x:to>
              <x:from>LAX</x:from>
              <x:date>2001-12-20</x:date>
             </x:TravelRequest>
            </x:return>
           </x:ReservationRequest>
        </rdf:RDF>
       </env:Body>
      </env:Envelope>
```

SOAP-сообщение, показывающее использование альтернативной схемы написания кода для элемента `Body`

В примере 17, тело SOAP-сообщения содержит описание маршрута путешествия с использованием схемы ресурсов и их свойств, описываемые с помощью синтаксиса Resource Description Framework (RDF) [RDF]. (Поскольку синтаксис RDF и его использование не входят в круг вопросов этого учебника, очень кратко можно сказать, что схема RDF сопоставляет ресурсы - такие как ресурс бронирования путешествий, доступный по адресу `http://travelcompany.example.org/reservations?code=FT35ZBQ` - другим ресурсам (или значениям) посредством свойств, таких как `passenger`, дат отправления в путешествие и возвращения из него `outbound` и `return` соответственно. Составление схем RDF для маршрута может выбираться, например, для того, чтобы приложение бронирования путешествия пассажира могло хранить эти схемы в RDF-совместимом календаре, который в этом случае можно будет использовать в более сложных сценариях.)

Различия между SOAP 1.1 и SOAP 1.2

SOAP Версия 1.2 имеет ряд отличий синтаксиса и дает дополнительную (или поясняющую) семантику положений спецификации [SOAP 1.1]. Далее приводится список функций, отличающих две спецификации. Цель этого списка - дать читателю удобную и легкодоступную сводку различий между двумя спецификациями. Функции разделены на категории сугубо для удобства пользования. В некоторых случаях один и тот же пункт помещен в несколько категорий.

Структура документа

- Спецификации SOAP 1.2 представлены в двух частях. [SOAP Часть 1] дает абстрактное, основанное на понятии информационного множества, определение структуры SOAP-сообщения, модели обработки и структуры привязок к нижележащим протоколам, в то время как [SOAP Часть 2] описывает правила сериализации при передаче информационного множества также как и определенную HTTP-привязку.
- SOAP 1.2 не поясняет акронимов.
- SOAP 1.2 была переписана в терминах информационных множеств XML, а не в терминах сериализаций вида <?xml....?>, как это было SOAP 1.1.

Дополнительный или измененный синтаксис

- SOAP 1.2 не разрешает присутствие каких-либо элементов после тела сообщения. Определение схемы SOAP 1.1 допускает такую возможность, однако текст самого описания умалчивает об этом.
- SOAP 1.2 не позволяет появление атрибута `env:encodingStyle` в элементе `env:Envelope`, в то время как SOAP 1.1 допускает появление этого атрибута в любом элементе. SOAP 1.2 специфицирует специальные элементы, в которых этот атрибут может быть использован.
- SOAP 1.2 определяет новый заголовочный элемент `env:NotUnderstood` для передачи информации в обязательном заголовочном блоке, который не может быть обработан, что показывается наличием кода ошибки `env:MustUnderstand`. SOAP 1.1 вводит код ошибки, но не дает детальной информации по его использованию.
- В основанном на информационных множествах описании SOAP 1.2, атрибут `env:mustUnderstand` в заголовочных элементах имеет (логическое) значение "true" или "false", в то время как в SOAP 1.1 он имеет символьные значения "1" или "0" соответственно.
- SOAP 1.2 вводит новый код ошибки `DataEncodingUnknown`.
- Различные пространства имен, определенные двумя протоколами, конечно, различны.
- SOAP 1.2 заменяет атрибут `env:actor` атрибутом `env:role`, имеющим по существу ту же семантику.
- SOAP 1.2 определяет новый атрибут для заголовочных блоков `env:relay`, указывающий должны ли передаваться дальше необработанные заголовочные блоки.
- SOAP 1.2 определяет две новые роли "none" и "ultimateReceiver" с детальной моделью поведения SOAP-узла при выполнении им этих ролей.
- SOAP 1.2 отменила "dot"-нотацию кодов ошибок, которые теперь являются просто XML Qualified Name, где префикс пространства имен является пространством имен SOAP-оболочки.

- SOAP 1.2 заменяет коды ошибок "client" и "server" на "Sender" и "Receiver".
- SOAP 1.2 использует имена элементов `env:Code` и `env:Reason` соответственно, для чего в SOAP 1.1 вызываются `faultcode` и `faultstring`. SOAP 1.2 также разрешает использование множественных дочерних элементов `env:Text` элемента `env:Reason`, уточняемых `xml:lang`, для вывода причины ошибки на различных языках.
- SOAP 1.2 разрешает использование иерархической структуры обязательного субэлемента SOAP `env:Code` в элементе `env:Fault` и вводит два новых необязательных субэлемента `env:Node` и `env:Role`.
- SOAP 1.2 устраняет различие, имевшее место в SOAP 1.1 между заголовком и телом ошибки, что указано присутствием элемента `env:Details` в `env:Fault`. В SOAP 1.2 присутствие элемента `env:Details` не имеет значения относительно того, какая часть SOAP-сообщения об ошибке была обработана.
- SOAP 1.2 использует XML Base [XML Base] для определения исходного URI для соответствующих URI-ссылок, в то время как SOAP 1.1 умалчивает об этом.

HTTP-привязка SOAP

- В HTTP-привязке SOAP 1.2, HTTP-заголовок `SOAPAction` , определенный в SOAP 1.1, удален, а из IANA был заимствован новый статусный HTTP-код 427 для обозначения (по усмотрению HTTP-сервера, инициировавшего обмен сообщениями) необходимости его присутствия для серверного приложения. Содержимое упомянутого выше HTTP-заголовка `SOAPAction` сейчас должно содержать значение (необязательного) параметра "action" медиа-типа "application/soap+xml", который указывается в HTTP-привязке.
- В HTTP-привязке SOAP 1.2 заголовок Content-type должен иметь значение медиа-типа "application/soap+xml", а не "text/xml" как в SOAP 1.1 Регистрация IETF этого нового медиа-типа пока не закончена [SOAP MediaType].
- SOAP 1.2 дает улучшенное детализированное описание использования различных серий статусных HTTP-кодов: 2xx, 3xx, 4xx.
- Из SOAP 1.2 была удалена поддержка структуры HTTP-расширений.
- SOAP 1.2 вводит дополнительный шаблон обмена сообщениями, который может быть использован как часть HTTP-привязки, позволяющей использование HTTP-метода GET для безопасного и идемпотентного поиска информации.

RPC

- SOAP 1.2 вводит элемент `rpc:result` средства доступа для RPC.
- SOAP 1.2 вводит в пространстве имен RPC несколько дополнительных кодов ошибок.
- SOAP 1.2 содержит руководство по определению RPC в соответствии с принципами архитектуры Web в случае, когда единственным назначением RPC является безопасный поиск информации.

Написание кода SOAP

- Для SOAP 1.2 сформулирована абстрактная модель данных, основанная на помеченном графе с ориентированными ребрами. Написание кода SOAP 1.2 зависит от этой модели данных. Соглашения об обозначениях SOAP RPC также зависят от этой модели данных, но не зависят от схемы написания кода SOAP. Поддержка схем написания кода SOAP 1.2 и соглашений RPC SOAP 1.2 необязательны.

- В SOAP 1.2 по сравнению с SOAP 1.1 был изменен синтаксис сериализации массива.
- Поддержка частично переданных и разбросанных массивов, которая была SOAP 1.1, более не доступна в SOAP 1.2.
- SOAP 1.2 разрешает внутристрочную (вложенную) сериализацию multiref-значений.
- Атрибут SOAP 1.1 `href` (принадлежащий типу `xs:anyURI`) в SOAP 1.2 называется `enc:ref` и принадлежит типу `IDREF`.
- В SOAP 1.2 опущенные средства доступа сложных типов приравнены NIL.
- SOAP 1.2 предоставляет несколько субкодов ошибок для указания ошибок написания кода.
- Типы на узлах в SOAP 1.2 сделаны необязательными.
- Спецификация SOAP 1.2 удалила характерные составные значения из модели данных SOAP.
- Спецификация SOAP 1.2 добавила необязательный атрибут `enc:nodeType` к элементам, написанным с помощью схемы написания кода SOAP и идентифицирующим ее структуру (т. е. простое значение, структуру или массив).

SOAP Часть 1 Приложение A описывает правила управления версиями для SOAP-узла, которые поддерживают переход с версии [SOAP 1.1] на SOAP Версия 1.2. В частности, они определяют заголовочный блок `env:Upgrade`, который может использоваться SOAP 1.2-узлом при получении [SOAP 1.1]-сообщения для отправки SOAP-сообщения об ошибке отправителю для указания поддерживаемой версии SOAP.

Ресурс

Данный документ представляет собой перевод на русский язык спецификации SOAP Version 1.2 Part 0: Primer, W3C Recommendation 24 June 2003, английский вариант которой является единственным официальным изданием.

Перевод данной спецификации на русский язык выполнен специалистами компании UBS.

Мы осознаем, что он неидеален, может содержать ошибки и неточности, и работаем над его улучшением.
Все комментарии, касающиеся настоящего перевода, просьба направлять по адресу: info@ubs.ru

WSDL: взгляд изнутри

Аннотация

Статья определяет место WSDL в процессе проектирования Web-сервисов.

Часть I

Введение

Недавно автору статьи довелось использовать язык WSDL (Web Services Description Language, Язык описания Web-сервисов) для нескольких существующих Web-сервисов - на одном клиенте работал сервер и клиентская реализация. У этого клиента и специалистов по обслуживание сервера уже сложились тесные рабочие отношения, но наступил момент, когда возникла необходимость в еще одной клиентской реализации, которую должна была выполнить группа разработчиков, находящихся в противоположенной точке земного шара. В связи с этим потребовалась четкая спецификация, описывающая такие сервисы, а именно для этого и предназначен WSDL. Поэтому автор вознамерился основательно изучить то, что ранее не было достаточно освещено. В результате, он приобрел ценный опыт - ему удалось вспомнить старую и добрую практику проектирования программного обеспечения и обнаружить ряд проблем, характерных для Web-сервисов, WSDL и XML Schema.

В самом начале, на этапе проектирования, было допущено несколько ошибок, которые изначально трудно заметить. Вероятно, эти недочеты не были бы допущены, если бы проектировщики дали формальное определение для своих сервисов на WSDL. Так что, вот основная мысль этих двух статей: *пишите WSDL заранее, не генерируйте его потом, как это часто советуют поставщики*.

При всем внимании, которым были одарены Web-сервисы, часто сложно отделить желаемое от действительного. В предлагаемом цикле статей акцент будет сделан на реальном, а не на потенциальном. В них читатель не найдет краткого обзора WSDL, кроме того предполагается, что он знаком с W3C XML Schema. В первой статье рассказывается, чем могут быть полезны при проектировании Web-сервисов практика проектирования программного обеспечения и опыт в области распределенной обработки данных. В ней рассматриваются некоторые решения, которые должен принимать проектировщик Web-сервисов, приводятся необходимые советы и рекомендации. В остальных статьях будет описан процесс проектирования: во второй статье будут перечислены некоторые "темные стороны" спецификации WSDL 1.1. В нее не войдут определения типов данных - темы третьей статьи, в которой будет представлена W3C XML Schema с позиции того, кто использует ее для определения данных, передаваемых через интерфейсы Web-сервисов.

Проектирование Web-сервисов

Употребляемый автором термин *Web-сервисов* относится исключительно к тому виду технологии, которая сосредоточена на взаимодействии (interoperability). Это означает, что эта технология стандартизирована: гетерогенные системы работают только при наличии отрытых стандартов. В этом случае уместен вопрос: будут ли Web-сервисы решением вашей проблемы? Сегодня одного слова Web-сервисы уже недостаточно, чтобы говорить о солидности компании, их использующей, поэтому будет лучше, если имеются иные веские основания для

их использования. Если вы контролируете оба конца канала, не исключено, что существуют более подходящие технологии. Сейчас - только заря эры открытых стандартов распределенной обработки данных. Поэтому цена поддержки Web-сервисов на этом еще несформировавшемся рынке остается высокой - это и снижение производительности, и увеличение затрат на разработку, и ухудшение защищенности.

Тезис о том, что Web-сервисы являются "дружественными по отношению к брандмауэру" ("firewall friendly"), обманчив. Действительно, обычные брандмауэры оберегают корпоративные ценности от "злоумышленников", которые используют слабые места в прикладном программном обеспечении, появившиеся в результате открытия портов, на которых исполняются защищенные сервисы. С другой стороны, Web-сервисы через этих порты "выставляют" прикладное программное обеспечение. Другими словами, они ослабляют безопасность, предоставляя посторонним лицам доступ к приложениям - именно то, чему сетевой брандмауэр должен был бы воспрепятствовать. Поэтому необходимо новое поколение брандмауэров. На рынке уже появилось несколько новых игроков, предлагающих такие продукты. Поставщики обычных брандмауэров также начинают обращать внимание на эту проблему. Но это только начало, и такая технология должна еще оправдать себя.

Даже если предполагается применять Web-сервисы, необходимо помнить, что необязательно использовать SOAP. Например, автор статьи видел формы, передаваемые как аргумент запроса на удаленный вызов процедуры SOAP (SOAP-RPC). Ему также попадались формы, которые возвращались как часть ответа удаленного вызова процедуры SOAP. Он так и не смог найти дополнительную пользу от применения SOAP. Разве не был бы проще обыкновенный XML или HTML через HTTP?

Недопустимость генерации WSDL

Ответим на вопрос: предназначен ли WSDL 1.1 для восприятия человеком. Типичный совет - генерировать WSDL, а не писать вручную. Автор по этому вопросу придерживается следующего мнения. Возможно, это и верно, если таким образом разрабатывается простой, демонстрационный сервис, но данный подход обречен на провал, когда он применяется к крупным системам. Даже программист, работающий самостоятельно, быстро утратит общее представление о проблеме; ситуация усложняется если разработку совместно ведут различные, территориально разнесенные группы специалистов. Большие распределенные системы требуют *проектирования*; ожидание того, что после объединения они будут совместно работать, приведет к катастрофе.

Если распределенные компоненты могут разрабатываться на различных языках программирования, то для того, чтобы указать, как должны быть задействованы сервисы, необходим Язык описания интерфейсов (Interface Definition Language, IDL), нейтральный по отношению к языку программирования. У CORBA (Общая архитектура посредника запросов к объектам, Common Object Request Broker Architecture), как и у DCOM (Распределенная модель компонентных объектов, Distributed Component Object Model), такой язык есть. IDL - это контракт между инициатором на обслуживание и поставщиком, но он только собирает синтаксис. Семантика остается неосвещенной: IDL оставляет открытым вопрос о том, что делает сервис.

WSDL - это IDL Web-сервисов. Он описывает, как вызывать Web-сервисы. Он также определяет ответы, которые могут быть получены как при успешном вызове, так и нет. Спецификация WSDL жестко регламентирует формат сообщений, используемые протоколы и адрес, по которому находятся сервисы. К сожалению, даже четкое и строгое описании на WSDL не гарантирует высокое качество проектирования. Подобно всем языкам IDL, WSDL

силен в синтаксисе и слаб в семантике. Однако, не стоит им пренебрегать - в конечном счете важна именно семантика, синтаксис же используется просто, чтобы ее раскрывать.

Проектирование интерфейсов

Прежде чем приступить к написанию WSDL, необходимо оговорить с заказчиками то, что Web-сервис должен делать. Следует записать случаи использования, четко определив, как этот сервис взаимодействует со своей средой. Предположим, что программа-агент (actor) с какой-либо целью вызывает Web-сервис. В этом случае, нужно создать не только сценарий "солнечного дня", который реализует поставленную задачу, но сценарий "дождливого дня", когда результат отрицательный. Какие гарантии может предложить система, что цель успешно достигнута? А когда нет? Где находится граница ответственности клиентской части и сервера?

Трудно переоценить важность четкого определения требований. На этом этапа стоит задуматься о цели проекта. В чем конкретно она заключается? Какие данные необходимо получать от клиента, и что должен поставлять сервер? Совершение ошибки на этом шаге чревато большими затратами.

Например, рассмотрим Web-сервис, который в качестве параметров принимает список установленных программ и величину свободного места на диске. Затем этот сервис должен вернуть список приложений, подлежащих обновлению. В случае если места достаточно, проблем не возникает. Однако, как быть, если его недостаточно? Как выбрать продукты, для которых необходимо установить новые версии? Предполагается ли, что клиент сам удаляет старые редакции программ, чтобы освободить место для новых? Это непростые вопросы, для решения которых потребуется разрабатывать сложные алгоритмы, при реализация которых не исключены ошибки. Разумеется, ни одну систему не следует определять подобным образом. Сервер должен предоставлять список приложений, зависимости между ними и требования, которые они предъявляют к ресурсам. Решение же о том, какой пакет установить, является задачей клиента. Поэтому поручив серверу это задание, клиент ничего не выиграет, он только повысит затраты на разработку серверной части.

Таким образом, необходимо проводить анализ затрат на начальном этапе. При этом можно проигнорировать технические детали - сервис можно рассматривать как черный ящик. Но нельзя пренебрегать деталями, касающимися взаимодействия между этим сервисом и его средой.

Следующий шаг - проанализировать, как можно реализовать случаи использования. Будет ли единственный интерфейс (`portType` в WSDL версии 1.1), обеспечивающий всю функциональность? Или несколько интерфейсов? Каждый интерфейс может быть предложен в нескольких конечных точках, но, как правило, он должен быть неделимым. То есть конечная точка должна предоставлять либо всю функциональность, либо ничего. Аналогично, интерфейс должен быть семантически последовательным. Сказанное носит рекомендательный характер и опирается на здравый смысл, хотя ничто в спецификации WSDL не препятствует иной интерпретации.

Требуются ли какие-нибудь поддерживающие интерфейсы? Как и когда они должны вызываться? Какова природа этой зависимости? Пока нет стандартов, а только рекомендации о том, как обращаться с "хореографией сервисов" (service choreographies), как разрешать сервисам выполнять гиперссылки к другим сервисам. Другими словами, это неисследованная проблемная область.

Возвращает ли разрабатываемый сервис результат клиенту? Должен ли клиент располагать информацией о том, что сервис успешно отработал? Должен ли клиент вызывать сервис синхронно? Или асинхронно? Или же сервис будет поддерживать оба типа вызова?

Непрозрачность сети

Сеть не является прозрачной - вызов локального сервиса и его удаленный вызов существенно отличаются друг от друга. В качестве "памятки разработчику" можно порекомендовать тезисы Питера Дойча (Peter Deutsh) "Восемь заблуждений о распределенной обработке данных" (eight fallacies of distributed computing).

Неверная семантика различна для сетевого окружения и локальной реализации: при неудачном вызове не всегда известно, исполнился сервис или нет.

WSDL 1.1 описывает следующие виды вызовов: односторонний (one-way), запрос-ответ (request-response), ответ на требование (solicit-response) и уведомление (notification). Однако, связывания WSDL поддерживают только односторонний вызов и вызов вида запрос-ответ. Также следует понимать, что реально, а что - нет. Другими словами, сегодня невозможно реализовать ситуацию, когда клиент ожидает с сервера асинхронное получение практического, "промышленного качества" и согласованного со стандартами результата. Это следует расценивать как досадное обстоятельство, поскольку это именно тот механизм, который является наиболее гибким в ситуации частичного сбоя.

WSDL позволяет перемешивать и согласовывать виды вызова и транспорт (transport). На этом этапе важно прибегнуть к здравому смыслу. Первое требование - а это не всегда очевидно - необходимо четко представить, как стек протоколов предлагаемого Web-сервиса будет себя вести. В качестве примера рассмотрим, что случится если асинхронно вызвать Web-сервис поверх механизма синхронного транспорта. Предположим, что SOAP-сообщение пересылается в одну сторону по HTTP: бизнес логика на клиенте формирует SOAP сообщение, чтобы вызвать удаленный сервис. В результате запрос HTTP посылается на сервер. Когда сервер получает запрос, он должен немедленно возвратить ответ HTTP, не обслуживая этот запрос. Бизнес логика на сервере должна отрабатывать *после того*, как этот ответ был возвращен. Этот ответ не должен содержать SOAP-сообщение. Ответ HTTP с кодом состояния 200 или 202 не означает, что сервис успешно отработал. Код сбоя, с другой стороны, должен гарантировать, что вызов сервиса не исполнился. Слой SOAP на клиенте не формирует новых сообщений до тех пор, пока он не получит ответ, или пока не истечет время ожидания.

Таким образом, односторонний вызов *не* "устраняет" задержки или неисправности в сети. Самое большее - он помогает избежать задержки с бизнес логикой на сервере. Автор употребил "самое большее", подразумевая, что эту функциональность стоит протестировать на инструментальной цепочке сервера, прежде чем полагаться на нее -корректная реализация является нетривиальной задачей для любого поставщика.

Используя HTTP в в качестве транспортного протокола, клиент может быть уверен, что его запрос был доставлен. Стоит заметить, однако, что если клиент не получил ответ, это не значит, что сервер не получил и не обработал корректно этот запрос. Например, сеть может "упасть" во время отправки ответа. Другими словами, реализовать вызов просто, *правильно завершенный вызов - нет*.

Отличие Web-сервисов от распределенных объектов

Объекты характеризуются состоянием. Следует избегать состояния, поскольку оно связывает ресурсы и ослабляет масштабируемость. Слабосвязанная архитектура разрешает отдельным компонентам изменяться, не разрушая систему. Весьма вероятно, что в будущем потребуется расширять Web-сервис. Как же при этом сохранить обратную совместимость?

Несмотря на то, что достоинство удаленного вызова процедуры (RPC) в легкости реализации, он не очень хорошо уживается со слабой связанностью: необходимо передавать фиксированный список параметров при каждом вызове Web-сервиса. Эту проблему можно в некоторой степени сделать менее острой, разрешив некоторым параметрам "не иметь значения". Оставим без рассмотрения возникающие в этом случае проблемы, связанные с обеспечением возможности взаимодействия, - более фундаментальное ограничение заключается в том, что этот прием позволяет исключать параметры, а не добавлять их. Вызовы в стиле документа проявляют себя лучше, когда части документа могут быть определены как необязательные. Они также могут быть спроектированы для расширяемости.

Определение ограниченных интерфейсов

Следует внимательно изучить то, что предлагается: и на уровне данных, и на уровне кодирования данных. Необходимо помнить, что придется продолжить поддерживать поставляемый интерфейс. Чем ограниченнее интерфейс, тем легче его реализация. WSDL же спроектирован для гибкости.. Из-за этой гибкости легко ошибиться в спецификации Web-сервисов и оставить многие опции реализации неохваченными . Следует избегать этого - клиенты и серверы должны уметь обрабатывать *все* сообщения, которые допустимы по контракту.

Типичная ошибка, совершаемая при реализации серверных систем, - установление соединения с базой данных при каждом запросе к этой базе. Автор совершил такую ошибку - выяснилось, что установление соединения с базой данных занимает больше времени, чем все остальное, вместе взятое. Чтобы минимизировать это падение производительности, можно прибегнуть к одному из двух приемов: воспользоваться пулом соединений (connection pool) или хранимой процедурой.

При использовании Web-сервисов стоимость установки соединения также высока. Опять-таки дополнительные затраты на вызов могут быть уменьшены повторным использованием существующего соединения. С другой стороны, необходимо учитывать, что поддержание открытого соединения требует ресурсов. Однако, важно помнить, что расходы, связанные с вызовами Web-сервиса, гораздо более высокие по сравнению с вызовом локальной функции и, следовательно, их следует реже использовать.

Здесь уместно провести аналогию с хранимыми процедурами. Вместо отправки данных и обработки их на клиенте, только чтобы выяснить, что требуются еще данные, хранимые процедуры оперируют с данными в адресном пространстве базы данных. Применение хранимых процедур в приложениях баз данных является спорным моментом - по крайней мере, по двум причинам. Во-первых, язык написания хранимых процедур не стандартизирован, и поэтому их использование ведет к зависимости от поставщика БД. Во-вторых, бизнес логика должна находиться в отдельном слое, а не в базе данных.

Тем не менее, хранимые процедуры решают реальную проблему, и их недостатки должны быть соотнесены с достоинствами в каждом отдельном случае. Аналогично, в области Web-сервисов, клиенту предоставляется контроль над логикой за счет множества мелко структурных вызовов. Крупно структурные сервисы более напоминают хранимые процедуры.

Они не только улучшат производительность, но и упросят процесс реализации, внедрения и управления всей системой.

Итак, практическая рекомендация - проектировать Web-сервисы под ограниченные, но крупно структурные интерфейсы. Ограниченность позволяет показывать как можно меньше, крупность структуры - выполнять как можно больше для одного вызова.

Разнесение бизнес логики и политики

Не следует реализовывать аутентификацию и авторизацию в бизнес логике. Безопасность транспортного уровня может быть как удовлетворительной, так и нет. В разделе Ресурсы приведены ссылки на статьи, в которых рассматривается этот вопрос. Даже если безопасность транспортного уровня и не урезается, все равно пока нет стандартов - они только разрабатываются, и из них стоит отметить создаваемые Техническим комитетом OASIS "Защищенность Web-сервисов" (OASIS Web Services Security Technical Committee). Предлагаемые подходы действительно способствуют разделению бизнес логики и политики (policy): заголовки SOAP вставляются в сервис, чтобы обеспечить характеристики безовасности, а информативная часть сообщения сохраняется для бизнес логики.

Но как же без ложки дегтя - предлагаемый стандарт применим только к Web-сервисам, использующим SOAP. Еще одна неприятность - отсутствие стандартизированного способа указать *Качество защиты* (Quality of Protection, QoP) сервиса в документе WSDL. Microsoft, BEA, SAP и IBM опубликовали спецификацию для "Приложений политики Web-сервисов" (Web Services Policy Attachments), в которой рассматривается этот вопрос. Насколько известно автору, этот документ еще не был передан ни в один орган стандартизации.

Разделение проектирования и реализации

В идеале при проектировании Web-сервисов следует оставить вопросы реализации в стороне. Суть проектирования -посмотреть на систему на высоком уровне абстракции, не обращая внимания на проблемы реализации. Однако, можно выбрать такой, казалось бы подходящий по всем параметрам дизайн проекта, который окажется бесполезным, если инструментальные цепочки, которые предполагается использовать, не поддерживают конструкции в разработанном WSDL. Автор настойчиво рекомендует создание макета по техническим спецификациям для проверки реальности осуществления проекта. Пока этот момент еще не проработан.

Часть II

Введение

В предыдущей статье ("WSDL: взгляд изнутри, часть I") автор привел общее описание процесса проектирования Web-сервисов. В ней отмечалось, что Язык описания Web-сервисов (Web Services Description Language, WSDL) только определяет синтаксис того, как Web-сервис может быть вызван; он не говорит ничего о его семантике. В этой статье этот вопрос получит дальнейшее рассмотрение.

В настоящий момент наиболее широко используется версия WSDL 1.1, опубликованная в качестве Примечания консорциума W3C (W3C Note). Она *не* является официальным стандартом. WSDL 1.1 предлагает широкие возможности для вызова Web-сервисов. При этом поддержка инструментов осуществляется посредством "патчей". Эта версия WSDL была встречена недоброжелательно, поскольку явила собой компромисс между выразительностью и гибкостью, с одной стороны, и многословностью и сложностью, с другой. Прежде чем продолжить, автор вынужден признаться, что он не представляет, как написать действительно четкий, точный WSDL.

Инструменты

Перед тем, как вдаваться в подробности, давайте рассмотрим инструменты, которые могут помочь. Во-первых, при написании WSDL можно воспользоваться XML-редактором, желательно с возможностью проверки валидности WSDL-документа. Модифицируя WSDL для существующих Web-сервисов, автор обнаружил, что очень полезно уметь генерировать, посылать и получать сообщения из редактора; XML Spy от Altova выполняет это для Web-сервисов, использующих SOAP и HTTP. Благодаря этому интерактивные разработка, тестирование и отладка WSDL получают практический смысл. К сожалению, XML Spy не предоставляет такую возможность для других протоколов. Важная функциональность, которое, похоже, отсутствует в сегодняшних инструментах - это возможность проверять ответ сервера на допустимость согласно описанию Web-сервиса.

Модульные описания Web-сервиса

Применяя ключевое слово `import`, можно разделить WSDL на документы-модули. Пример, приведенный в Примечании консорциума W3C состоит из трех документов, которые содержат, соответственно, определения типов данных, абстрактные определения и специфические связывания сервиса. Эти специфичные или конкретные определения сервисов зависят от абстрактных определений сервиса, которые в свою очередь зависят от определений типов данных.

Помимо улучшения читабельности этот подход может также повысить возможность расширения и повторного использования некоторых типов: одни и те же определения типов данных могут быть использованы во многих абстрактных сервисах, а одни и те же абстрактные сервисы могут быть использованы во многих различных связываниях по многим адресам.

Вначале множество Web-сервисов может быть представлено в виде набора из трех документов. Однако, по мере того, как сервис развивается, это может вылиться в дерево документов, у которого в корне находятся определениями типов данных, а его ветви расходятся к нескольким документам абстрактных сервисов и далее к конкретным сервисам.

Элемент `portTypes` - это набор семантически связанных операций, во многом подобный интерфейсу в языке программировании. Элемент `binding` не обязан охватывать все элементы `operation` данного `portType`. Например, определенные в одном `portType` элементы `operation` могут использовать различные транспортные протоколы. Поскольку `service` - это набор элементов `port`, а `port` ссылается на отдельный `binding`, можно определять сервисы, которые не предлагают все `operation`, определенные в `portType`. Однако, наличие таких сервисов является признаком плохо разделенных элементов `portType`.

Отдельный сервис может быть составлен из множества интерфейсов, каждый из которых представляет отдельный аспект этого сервиса. Например, у сервиса может быть порт для своей бизнес логики и еще один порт для доступа к функциям управления.

Пространства имен

Описания Web-сервиса, как только они согласованы, должны быть зафиксированы. WSDL 1.1 предусматривает необязательное использование целевого пространства имен. Однако, удобно назначать пространство имен для недвусмысленной идентификации сервисов и их версий - как принято делать в случае спецификаций стандартов. Сложность состоит в том, что WSDL 1.1 не очень четок относительно того, что подразумевается под целевым пространством имен. Другими словами, что точно вкладывается в это понятие? По мнению автора, под ним понималось те же правила, что и в W3C XML Schema.

Напомним кратко эти правила: то, что помещается в целевое пространство имен регулируется атрибутом `form` в элементах `element` и `attribute`. При отсутствии такого атрибута управление передается набору значений по умолчанию в атрибутах `elementFormDefault` и `attributeFormDefault`, находящихся в корневом элементе `schema`. При отсутствии явного задания значений по умолчанию в силу вступают неявные значения по умолчанию: к пространству имен относятся только элементы, определенные глобально.

В каком виде эти правила W3C XML Schema нашли свое отражение в WSDL 1.1? Во-первых, в WSDL не задействованы элементы `element` и `attribute`. Во-вторых, элементами верхнего уровня в WSDL являются `messages`, `portTypes`, `bindings` и `services`, являющиеся сущностями (entity), которые должны быть помещены в целевое пространство. Очевидно, в данном случае `type` не релевантны. В-третьих, несмотря на то, что теоретически атрибут `form` мог бы использоваться с любыми элементами, определенными в WSDL, это не является принятой практикой. В-четвертых, аналогично сказанному, атрибуты `elementFormDefault` и `attributeFormDefault` могли бы использоваться в элементе `definitions`, но автор с этим не сталкивался.

Таким образом, можно утверждать, что все `message`, `portType`, `binding` и `service` оказываются в целевом пространстве имен, а их потомки - нет.

Удобно ли это правило? А имеет ли это значение? Это имеет смысл, когда сообщения зашифрованы так, что - за исключением этого правила - пространство описания Web-сервиса могло бы "просочиться" в сообщение. Действительно, автор потратил достаточно времени, чтобы выяснить, что пространства имен, которые находились в зашифрованном SOAP-сообщениях, не были в пространстве имен WSDL из-за того, что, согласно принятой, но обескураживающей практике, то же самое пространство имен используется в качестве входного для шифрования SOAP. Непосредственным результатом указанного правила пространства имен является *отсутствие* элементов и атрибутов сообщения в пространстве имен описания Web-сервиса, если только они не помещены туда пространством имен шифрования SOAP.

Если описание Web-сервиса разложено на модули, согласно приведенной выше рекомендации, каждому документу должно быть назначено пространство имен. Описания Web-сервиса не должны импортировать (`import`) определения с одинаковым пространством имен. Спецификация WSDL не формулирует это правило явно, но опять-таки, по мнению автора, наличие элемента `import` в W3C XML Schema явился причиной появление одноименного элемента в WSDL и расширения тех же правил. Schema не разрешает импортирующей и импортируемой схеме совместно использовать пространство имен. По оценке автора, это очень удобное правило, поскольку сложно рассуждать о пространстве имен, если нет уверенности, что определение полное.

Обработка ошибок

Возможно, единственно наиболее важное отличие между демо- и готовым сервисом заключается в качестве обработки ошибок. Тем не менее, в литературе этому вопросу удаляется мало внимания.

Определение отдельных сообщений с целью установления различных сбойных ситуаций может быть удобно - структура таких сообщений может меняться согласно возникшей сбойной ситуации. В Листинге определен Web-сервис, который приведет в бешенство системных администраторов: он запускает двоичный код (binary), по запросу вызывающей программой. Читатель, возможно, будет разочарован, не обнаружив примеров использования пространств имен и разделения на модули - этот пример не рассматривает эти особенности. Он иллюстрирует обработку ошибок - при проектировании была предусмотрена возможность появления двух ошибок: первая - если недостаточно памяти, в этом случае возвращается область выделенной динамической памяти и количество используемой памяти; вторая - если программа попытается разыменовывать нулевой указатель, в этой случае возвращается запись стека.

Типы сообщения о сбоях получены из абстрактного типа. Причина использования абстрактных типов может быть объяснена желанием провести аналогию с отображением ошибочных действий в иерархию наследования во многих языках программирования. Поэтому исправленный Листинг - это более компактное описание Web-сервиса. Разница между наборами сообщений о сбоях, которые допустимы согласно соответствующим документам, незначительны.

Замысел, однако, намного яснее в первом документе. Разработчики клиентской реализации, работая со вторым документом вероятно смогут предположить, что может быть возвращено и сообщение `OutOfMemoryException`. Но из спецификации неясно, имеет ли это значение. Также они не могут быть уверенными, что код клиентской реализации охватывает все сбойные ситуации, поскольку абстрактный класс tException может быть расширен типами, отличными от перечисленных во втором документе.

Автор сталкивался с гибридным подходом, при котором сложный тип не объявлен абстрактным. В этом случае сообщения о сбоях могут быть определены в описании Web-сервиса как производный тип, иногда определяется сложный тип. В результате, возникает еще большая неопределенность относительно точного формата, который получит клиент - автор не рекомендует этот подход.

`document/literal` против `rpc/encoded`

Здравый смысл предписывает использовать текстовый формат передачи при вызове документа и кодированный при использовании RPC (Remote Procedure Call, Удаленный вызов

процедуры). Однако, фундаментальных причин, почему необходимо следовать этому правилу, нет - это скорее исторически сложившиеся стечение обстоятельств, что инструменты, как правило, поддерживают эти комбинации.

Выбор осуществляется между сложностью модели программирования и сложностью маршалинга (marshalling) сообщений. RPC предлагает удобную модель программирования. Она не без изъянов, как отмечалось в предыдущей статье. Однако, также важно знать об обязательствах, которые форматы передачи, используемые с RPC, накладывают на маршалеров: различие между `literal` и `encoded` - это различие между составителем и читателем. Это означает, что у второго формата передачи может быть множество различных представлений для семантически эквивалентных сообщений.

Классический пример - поддержка кодирования SOAP для независимых и встроенных элементов. Понимать все представления - дело получателя. В предыдущем разделе уже говорилось о важности максимально точного определения формата сообщения. Причина тому - высокая стоимость принятия правильного решения читателем. Это, возможно, не имеет значения в средах, в которых доступны сложные инструменты генерации клиентской заглушки (client stub), но при работе на гетерогенных платформах это не всегда так.

Что можно ожидать

Как уже отмечалось, WSDL 1.1 не имеет статуса стандарта. И все же эта спецификация широко используется, часто не оправдывая надежд на возможность взаимодействия. Именно это и является причиной появления Организации по развитию возможности взаимодействия Web-сервисов (WS-I) - не получить право собственности на стандарт WSDL, а определить очертания, "состоящие из набора некоммерческих спецификаций Web-сервисов наряду с уточнениями и поправками к тем спецификациям, которые способствуют возможности взаимодействия".

Конечно, наличие еще одной организации стандартизации вызывает раздражение. Несмотря на заявленные цели, автор не может отделаться от ощущения, что деятельность организаций, схожих с WS-I, может привести к появлению взаимоисключающих стандартов. Тем не менее, он посоветовал бы ознакомиться с разделом 5 "Рабочего проекта принятия Basic Profile" (Basic Profile Approval Draft), в котором содержатся отличное разъяснение некоторых "дыр" WSDL 1.1. И все же автор не одобряет то, что организация уделяет максимум внимания SOAP.

В предыдущей статье также говорилось о Техническим комитете OASIS "Защищенность Web-сервисов" (OASIS WSS TC), который, кажется, становится лидером в области определения стандартов защищенности Web-сервисов. Это еще одна организация, которая решает часть поставленной выше задачи. Но смогут ли подойти друг к другу эти части, и кто собирается их объединять?

Право собственности на будущие версии WSDL, похоже, однозначно остается у консорциума W3C, где Рабочая группа по описанию Web-сервисов (Web Service Description Working Group) занята написанием WSDL 1.2. Согласно ее уставу, выход этой версии запланирован на май 2003 года. Эта срок, очевидно, будет сорван. Тем не менее, группа время от времени публикует рабочие проекты будущей редакции. Так, что же будет со "слабыми сторонами" WSDL , о которых шла речь выше?

Если судить по проекту, доступному на момент написания этой статьи, похоже, подтверждается интерпретация того, что происходит в целевом пространстве имен описания

Web-сервисов. В нем говорится, что "*информационная единица атрибута* `targetNamespace` определяет присоединение пространства имен для компонентов верхнего уровня, определенных в этой *информационной единице элемента* `definitions`. Сообщения, типы порта, связывания и сервисы являются компонентами верхнего уровня". Будет ли WSDL 1.2 поддерживать реализацию нескольких интерфейсов является предметом жарких дебатов. В проекте WSDL 1.2 явно указано, что для используемых пространств имен с импортированными документами применяются те же правила как и в XML Schema. С другой стороны, альтернативный подход по разделению описаний на модули обеспечивается посредством элемента `include`, моделируемого по элементу `include` XML Schema, который не допускает совместного использования пространств имен.

Ресурс

Оригинал статьи: "WSDL Tales From The Trenches, Part 1" <http://webservices.xml.com/pub/a/ws/2003/05/27/wsdl.html> by Johan Peeters, Copyright © 2003 xml.com, O'Reilly & Associates, Inc. Перевод на русский язык - Copyright © 2003 Intersoft Lab. Опубликовано по адресам: http://www.iso.ru/journal/articles/271.html и http://xmlhack.ru/texts/wsdl.tales/wsdlintralook1.html, перепечатка возможна только с согласия Intersoft Lab. Преобразование в формат DocBook - Copyright © 2003 xmlhack.ru.

Элементы сервисно-ориентированного анализа и проектирования. Междисциплинарный подход к моделированию в проектах построения SOA

Аннотация

Настоящая статья является первым русскоязычным описанием основ сервисно-ориентированного анализа и проектирования - междисциплинарного подхода к построению сервисно-ориентированных архитектур (SOA). Показана необходимость разработки сервисно-ориентированного анализа и проектирования на основе комбинации элементов методологий EA, BPM и OOAD с новыми оригинальными конструкциями. Приводятся принципиальные отличия предлагаемого подхода от объектно-ориентированного анализа и проектирования.

Введение

Основные положения концепции SOA и веб-сервисов сформулированы на основе повседневного опыта и признаны в качестве общепринятого архитектурного стиля разработки современных корпоративных приложений. В этом контексте, основной вопрос состоит в следующем: *что делает хорошие сервисы* все более и более критичными для обеспечения успешной реализации SOA.

Существующие на сегодняшний день методологии моделирования, такие как *Object-Oriented Analysis and Design (OOAD), Enterprise Architecture (EA)* и *Business Process Modelling (BPM)* являются высококлассными практиками, которые, безусловно, полезны для идентификации и определения нужных абстракций при построении архитектуры системы. Однако, опыт показывает, что эти практики не оправдывают ожиданий, будучи примененными по отдельности.

В этой статье мы рассмотрим подходящие для наших целей элементы OOAD, EA и BPM. Мы также попробуем построить некий гибридный подход, комбинирующий отдельные элементы вышеуказанных методологий, а также некоторые новые элементы. В результате мы попробуем сформулировать междисциплинарный OOAD-метод, способствующий успешному развертыванию SOA, который мы будем называть *сервисно-ориентированным анализом и проектированием* (Service-Oriented Analysis and Design, SOAD). В этой статье мы делаем лишь первые шаги в описании данного метода.

Понятие сервисно-ориентированности

В ситуации, когда для бизнеса возможно появление новых возможностей или угроз, архитектурный стиль SOA способствует построению корпоративных бизнес-решений, обладающих способностью расширять либо изменять функциональность *по требованию*. SOA-решения состоят из повторно используемых сервисов, обладающих четко описанными, широкодоступными и стандартизованными интерфейсами. SOA предоставляет механизм интеграции существующих унаследованных приложений безотносительно программных платформ и языков программирования, на которых они реализованы.

Концептуально в SOA существует три основных уровня абстракции:

- *Операции*: Транзакции, представляющие собой отдельные логические единицы работы (logical unit of work, LUW). Выполнение операции обычно заключается в однократном или многократном чтении, записи или изменении данных. SOA-операции можно прямо сопоставить с методами в объектно-ориентированном подходе. Они имеют специфичный структурированный интерфейс и возвращают также структурированные отклики. Так же как и в случае методов, выполнение некоторой операции может привести к инициации выполнения других операций.
- *Сервисы*: Представляют собой логические группы операций. Например, если мы посмотрим на *CustomerProfiling (Создание профиля клиента)* как на сервис, то тогда *Lookup customer by telephone (Найти клиента по номеру телефона)*, *List customers by name and postal code (Составить список клиентов с именами и почтовыми индексами)* и *Save data for new customer (Сохранить данные о новом клиенте)* будут ассоциированными с этим сервисом операциями.
- *Бизнес-процессы*: Стабильно существующий набор действий либо деятельностей, выполняемых с определенной бизнес-целью. Бизнес-процессы обычно осуществляют многократные вызовы сервисов. Примерами бизнес-процессов являются: *Initiate New Employee (Принять на работу нового сотрудника)*, *Sell Products or Services (Продавать товары или услуги)* и *Fulfill Order (Исполнить приказ)*.

В терминах SOA бизнес-процесс состоит из совокупностей операций, которые исполняются в заданном порядке согласно заданному набору бизнес-правил. Упорядочение, отбор и исполнение операций называется хореографией сервиса или процесса. Как правило, хореографируемые (здесь мы вводим новый русскоязычный термин, отражающий перевод англоязычного choreographed - прим. перев.) сервисы вызываются для того, чтобы реализовать какие-либо бизнес-события.

С точки зрения моделирования наиболее значимой представляется задача систематизированного описания и создания правильно спроектированных абстракций операций, сервисов и процессов. Связанные с этим вопросы являются в настоящее время наиболее обсуждаемыми среди отраслевых специалистов и ученых мужей. Нам не известен ни один SOA-проект или семинар, в ходе которого данный аспект моделирования не был бы предметом серьезного обсуждения. Поэтому давайте рассмотрим его подробнее.

Почему не достаточно BPM, EA и OOAD

Опыт первых проектов построения SOA показывает, что существующие методологии разработки ПО и нотации, такие как OOAD (Object-Oriented Analysis and Design, OOAD), Enterprise Architecture (EA) и Business Process Modelling (BPM) охватывают лишь часть из того, что необходимо для поддержки парадигмы SOA. Несмотря на то, что SOA-подход интенсивно способствует укреплению хорошо устоявшихся, общеархитектурных принципов разработки ПО, такие как инкапсуляция, модульность и декомпозиция, он также вводит и некоторые новые понятия, такие как хореография сервисов, репозитарии сервисов и шаблоны сервисной шины промежуточного ПО (service bus middleware pattern), которые требуют пристального внимания в процессе моделирования.

Рис. 1 иллюстрирует области применения существующих на сегодняшний день подходов к моделированию (EA, BPM и OOAD). Данный рисунок представляет собой также хорошую отправную точку для дальнейшего обсуждения SOAD. По горизонтальной оси рисунка представлены фазы жизненного цикла проекта, по вертикальной - различные уровни абстракций или области знаний, где обычно применяется моделирование.

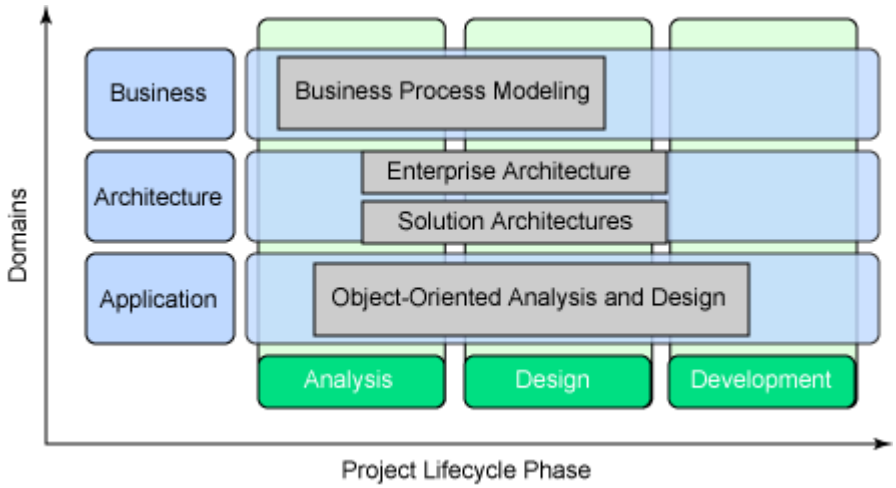


Рис. 1: Позиционирование BPM, EA и OOAD

Концепция SOA в значительной степени принимается легче, поскольку ее техническая основа хорошо известна. Например, применение общеархитектурных принципов создания ПО и объектно-ориентированных техник является общепринятой практикой для любого SOA-проекта. Однако, как уже было сказано, вопрос, наиболее часто задаваемый начинающими - как следует определять нужные сервисы. Мы упомянули ранее, что OOAD, EA и BPM не дают удовлетворительного ответа на этот вопрос, будучи примененными изолированно друг от друга.

Методология OOAD, описанная в оригинальных книгах Буча и Якобсона (выпущенными около десятилетия назад), являет собой прекрасную отправную точку для разработки SOA. OOAD имеет дело с абстракциями микро-уровня - классами и экземплярами объектов, хотя применение OOAD-техник и нотации Unified Modeling Language (UML) на архитектурном уровне было общепринятой практикой на протяжении многих лет. Так как модель вариантов использования часто создается отдельно для каждой предметной области и, следовательно, для проекта разработки приложения, часто целостная картина ситуации по всему предприятию оказывается размытой. Кроме того, по различным причинам модели вариантов использования не всегда бывают синхронизированы с их BPM-аналогами.

Подходы к построению корпоративных интеграционных приложений, такие как Treasury Enterprise Architecture Framework (TEAF), Feature-Oriented Domain Analysis (FODA) и Zachman дают высокоуровневое представление об архитектуре приложений, однако не описывают методы идентификации необходимых общекорпоративных абстракций, улучшающих качество свойств повторности использования и жизнеспособности решений.

Такие же BPM-подходы, как, например, BPMI, дают законченное представление о функциональных единицах работы, однако, они, как правило, не дают детального описания предметных областей архитектуры и реализации. Например, до появления таких языков как *Business Process Execution Language for Web Services (BPEL)*, в BPM-нотациях не существовало описания операционной семантики. Более того, мы видели много ситуаций, когда процессы моделирования и разработки были вовсе отделены друг от друга.

Наконец, ни одна из существующих дисциплин не описывает того, как *существующие приложения* могут быть адаптированы для SOA - везде, как правило, используется принцип построения SOA "сверху вниз". Существующие системы обычно хранят большое количество критичных данных и бизнес-логики и легко заменить их нельзя. Следовательно, для этих систем должен применяться также анализ по принципу "снизу вверх" с целью изучения

возможностей их "обертывания" (*здесь мы вводим новый русскоязычный термин "обертывание приложения", означающий облечение приложения в совокупность интерфейсов для интеграции с другими приложениями без изменения внутренней структуры приложения; в оригинале присутствует термин "wrapping" - прим. перев.*) и рефакторинга. Принимая во внимание наличие существующих приложений, мы приходим к необходимости построения некоторой компромиссной, сходящейся (*в том смысле, что данная методология будет сочетать как подход "сверху вниз" так и подход "снизу вверх" - прим. перев.*) методологии построения SOA.

Учитывая вышесказанное, необходим некий гибридный SOAD-подход к моделированию. Этот подход должен вобрать в себя лучшие элементы OOAD, BPM и EA, при необходимости дополнив их новыми элементами. Рис. 2 иллюстрирует средства SOAD (элементы и техники) описываемого нового подхода.

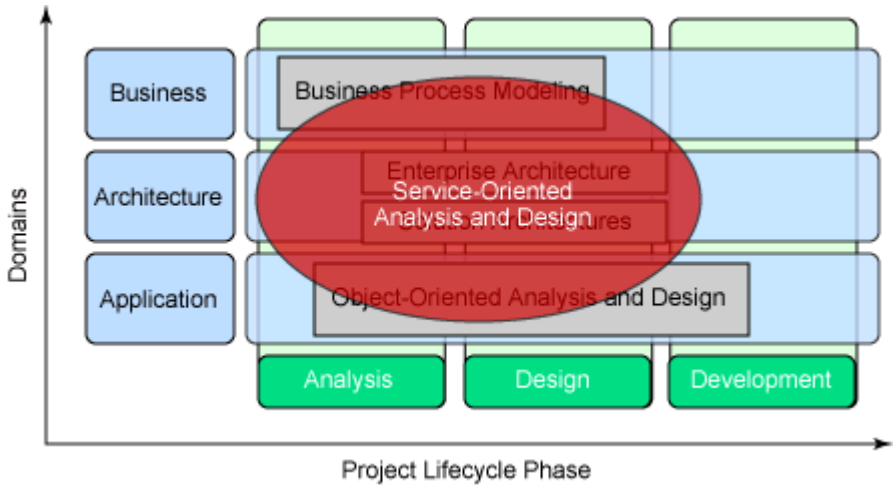


Рис. 2: SOAD и его составляющие: OOAD, BPM и EA

EA

Вовлечение корпоративных приложений и IT-инфраструктуры предприятия в процесс построения SOA может быть серьезным начинанием, затрагивающим множество аспектов бизнеса и организационных единиц. Поэтому в подобных случаях необходимо применять методологии EA и референтные архитектуры EA, такие, например, как The Open Group Architecture Framework (TOGAF) и Zachman, которые обеспечивают архитектурную совместимость отдельных решений.

Основываясь на имеющемся опыте можно утверждать, что большинство существующих EA-методологий имеют ограничения в одной или нескольких предметных областях. Например, если основным предметом рассмотрения методологии является взаимодействие на макроуровне низкоуровневых блоков, представляющих технические устройства, то в этом случае не может быть получено ни описание процесса на бизнес-уровне, ни представление о системе с точки зрения сервисов. Однако, в контексте SOA, подобный образ мышления необходимо сменить на подход, сконцентрированный на логических блоках, представляющих бизнес-сервисы, на определении интерфейсов и *сервисных соглашений Service Level Agreements (SLA)* между сервисами.

Более того, многие референтные архитектуры и методологии уровня предприятия являются довольно общими и не имеют глубокого описания процессов проектирования. Подобные общие архитектуры не дают архитекторам и разработчикам решений конкретных,

тактических советов, следствием чего часто бывает фундаментальный разрыв между корпоративной архитектурой и архитектурой создаваемых решений.

SOAD должен помочь архитекторам SOA в выработке целостного представления о ландшафте сервисов на бизнес-уровне. Это то, что не могут дать сегодняшние EA-методологии без улучшений, адаптирующих их для SOA. Инициатива IBM On Demand Operating Environment (ODOE) является большим шагом в этом направлении.

BPM

BPM является фрагментарной дисциплиной, сочетающей в себе множество различных стилей, нотаций и средств. Например, использование UML обычно простирается от предметной области к BPM-уровню. Другой часто используемой техникой, как указывается в работе Baker и Longman, является описание *событийно-обусловленных процессных цепочек*, представляющих собой концептуальные потоки процессов. Данная техника использует нотацию, отличную от UML.

Более того, существует множество вполне адекватных подобных BPM-техникам подходов, которые можно рассматривать как некие конкурентные преимущества консультационных компаний и вендоров систем класса Enterprise Resourse Planning (ERP). В качестве примеров можно привести ARIS Implementation Platform®, *Line of Visibility Enterprise Modeling (LOVEM™)* и инициативу IBM *Component Business Modeling (CBM)*

Последней тенденцией является определение стандартного пути представления исполняемых потоковых моделей (например, *Business Process Execution Language for Web Services (BPEL)*). BPEL расширяет сферу применения процессных моделей от фазы анализа до фазы реализации. Подобные исполняемые модели поднимают весь спектр новых вопросов, таких как:

- Какие аспекты должны быть описаны посредством BPEL, а какие - посредством WSDL? В чем различие между процессной моделью и более традиционными программными моделями?
- Каким образом такие элементы как нефункциональные требования и характеристики качества сервиса должны учитываться в моделях?
- Какое количество логики должно реализовываться с помощью расширений BPEL-движков для языков программирования, а какое ее количество - с помощью более привычного программирования, например, в J2EE?
- Каким образом можно оценить качество исполняемых процессных моделей и какие лучшие практики могут быть для этого применены?
- Какую роль играет инжиниринг BPEL-потоков? Является ли эта роль ролью в области бизнес-экспертизы (например, роль аналитика) либо ролью в области разработки (например, роль архитектора ПО)?

Все существующие на сегодняшний день BPM-подходы могут рассматриваться в качестве отправной точки для SOAD, однако, их необходимо усилить дополнительными техниками, позволяющими отделить предполагаемые сервисы и их операции от процессных моделей. Более того, процессное моделирование в SOAD должно быть синхронизировано с моделированием вариантов использования (*здесь дается наиболее распространенный в русскоязычной литературе вариант перевода англоязычного термина "use case" - прим. перев.*) на стадии проектирования и отвечать на вопросы, относящиеся к моделированию с помощью BPEL.

Объектно-ориентированная (ОО) парадигма против сервисно-ориентированной (SO)

Объектно-ориентированный анализ является весьма мощным и проверенным подходом, поэтому в SOAD необходимо использовать техники ОО-анализа настолько широко насколько это возможно. Чтобы успешно применять ОО-анализ в SOA-проектах, необходимо проверять одновременно более чем одну систему. При этом модели вариантов использования будут по-прежнему играть важную роль. Однако, SOAD должен быть преимущественно *процессным*, нежели *основанным на вариантах использования*. Поэтому в SOAD должна присутствовать сильная связь между BPM и моделированием вариантов использования.

На уровне *проектирования* цель ОО - сделать быстрым и эффективным проектирование, разработку и выполнение гибких и расширяемых приложений. Объекты являются конструкциями ПО, отражающими поведение реальных сущностей, моделями которых они служат. Например, объект Customer (Клиент) будет наделен именем, контактной информацией, а также может иметь один или несколько ассоциированных с ним объектов, олицетворяющих банковские счета клиента. С точки зрения ОО, все является объектами.

Фундаментальными принципами ОО являются:

- *Инкапсуляция*: Объектом ПО является обособленный программный пакет, имеющий физические свойства (данные) и функциональность (поведение), имитирующие его аналог в реальном мире. Объект Account (Счет), например, имеет баланс и механизм его пополнения либо уменьшения.
- *Сокрытие информации*: Хорошо структурированные объекты имеют простые интерфейсы и не раскрывают для внешнего мира внутренние механизмы своего функционирования. Примером сокрытия информации, заимствованным из реального мира, может служить такая аналогия: нет необходимости детально понимать, как работает автомобиль, чтобы ездить на нем.
- *Классы и экземпляры*: Классы представляют собой шаблоны, определяющие какой тип свойств и поведения имеет данный тип объектов ПО. *Экземплярами* называются обособленные объекты, имеющие индивидуальные значения вышеупомянутых свойств. Создание нового экземпляра класса называется *инстанциацией*. Если использовать биологическую аналогию, то человек будет являться классом. Все люди имеют такие свойства как рост, вес, цвет волос и глаз и т. д. Каждый человек является экземпляром класса, например, HumanBeing, со своими значениями роста, веса и т. д. Классы не имеют ограничений на время существования, в то время как их экземпляры имеют ограниченное время жизни.
- *Ассоциации и наследование*: Способность отражать ассоциации между классами и объектами является ключевым понятием ОО. Наследование можно определить как сильную форму ассоциации, используемую для отражения отношений "является" ("is-a"): аналогичным образом биологические виды организованы в иерархии царств, типов, классов, отрядов, семейств, родов и видов. В дальнейшем мы часто будем встречать аналогии иерархии объектов ПО в реальном мире. Когда мы создаем финансовое приложение *Entities*, нам скорее всего потребуется спроектировать такие классы как CheckingAccount (Проверить баланс счета), SavingsAccount (Сберегательный счет) и LoanAccount (Ссудный счет). Если попробовать разобраться чуть подробнее (см. Рис 3), то можно увидеть, что эти классы имеют много общих свойств, таких как, например, баланс счета, возможность пополнять либо уменьшать баланс счета и так далее.

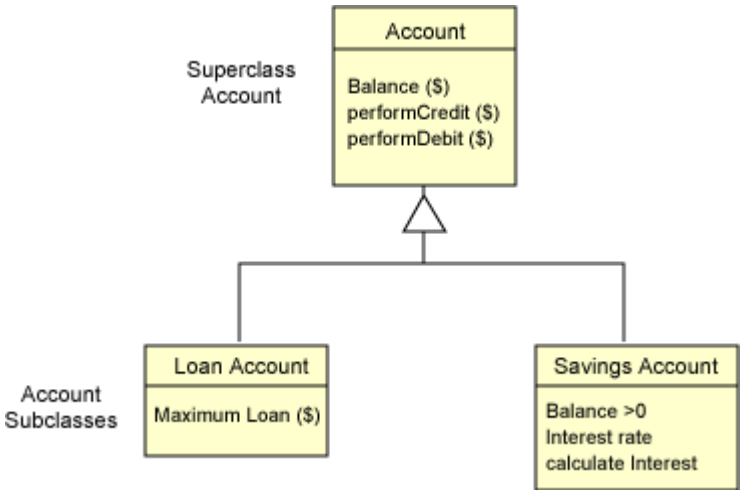


Рис. 3: Пример наследования класса в нотации UML

Вместо того, чтобы дублировать код, который определяет и управляет этими свойствами, можно создать общий родительский класс Account (Счет), который будет иметь баланс счета и сможет обрабатывать транзакции, связанные с пополнением либо уменьшением счета. Все другие классы будут являться некими специализированными формами объекта Account. Например, класс LoanAccount будет иметь отрицательный баланс при значениях между нулем и некоторым условленным максимумом, а SavingsAccount будет иметь положительный баланс, отражать операцию начисления процентов и т. п.

- *Обмен сообщениями*: Для того, чтобы выполнять любую полезную работу, объекты ПО должны иметь возможность взаимодействовать друг с другом. Взаимодействовать они могут, обмениваясь сообщениями. Например, перечисление $1000 со счета, обслуживающего чеки клиента, на сберегательный счет может быть выполнено путем отправки сообщения *debit* с аргументом $1000 экземпляру класса CheckingAccount и соответствующего сообщения *credit* экземпляру класса SavingsAccount. Когда экземпляр получает сообщение, он выполняет соответствующую функцию, называемую методом, которая имеет то же наименование, что и сообщение.
- *Полиморфизм*: Данный термин описывает ситуацию, когда при получении одного и того же сообщения два или более класса реагируют на него различным образом. Например, и FreeCheckAccount и CheckingAccount должны отвечать на сообщение *debit ($100)*, но экземпляр FreeCheckingAccount баланс должен дебетовать счет ровно на $100, в то время как экземпляр CheckingAccount должен дебетовать его баланс на $100 плюс комиссия за транзакцию.

ОО поддерживает весь жизненный цикл создания приложений, включая анализ, проектирование и собственно разработку приложений:

- *OOAD* позволяет определить оптимальный набор объектов и наиболее естественную иерархию классов для их реализации.
- *ОО-разработка* фокусируется на инкрементной разработке приложений, в ходе которой за одну итерацию реализуется лишь один бизнес-сценарий или вариант использования. Такой инструментарий как IBM WebSphere® Studio Application Developer помогает разработчикам быстро создавать и тестировать ОО-приложения.
- *ОО-runtime-среда*, реализуемая, например, с помощью Java™ Virtual Machine и J2EE, имеющей механизм разнесения объектов по разным серверам для обеспечения взаимодействия между ними, выполняет программный код и

предоставляет приложениям сервисы (такие как сбор мусора например (удаление ресурсов, которые больше не нужны как объекты)).

Главная проблема современных ОО-методов проектирования при их использовании в сервисно-ориентированном подходе - степень детализации их моделей находится на уровне класса, что является слишком низким уровнем абстракции для моделирования бизнес-сервисов. Сильные ассоциации, такие как наследование, создают *сильное связывание* (и, следовательно, сильную зависимость) между взаимодействующими сторонами. Сервисно-ориентированная парадигма напротив способствует гибкости и возможности быстрого перестроения систем посредством *слабого связывания*. В настоящее время в концепции SOA не существует поддержки кроссплатформенного наследования и четкого представления *экземпляра* сервиса. Решение данных вопросов позволило бы избежать многих проблем при реализации вспомогательных операций жизненного цикла сервисов, подобных удаленному сбору мусора (*здесь по-прежнему слово "мусор" следует трактовать как программные объекты, в которых отпала необходимость - прим. перев.*).

Приведенные рассуждения делают затруднительным прямое сравнение объектно-ориентированного подхода с сервисно-ориентированным. Однако, объектно-ориентированный подход по-прежнему является весьма полезным при проектировании основных классов и компонентной структуры сервисов. Более того, многие OOAD-техники, такие как карты Classes, Responsibilities and Collaborations (CRC) например, могут оказаться удобнее при моделировании сервисов, если их применять на более высоком уровне абстракции. Позднее мы вернемся в нашей статье к этому вопросу.

Рис. 4 демонстрирует соответствие между уровнями видимости и фокусами объектно-ориентированного, *компонентно-ориентированного* и сервисно-ориентированного подходов. Он также показывает, как данные подходы соотносятся друг с другом в контексте SOA и SOAD.

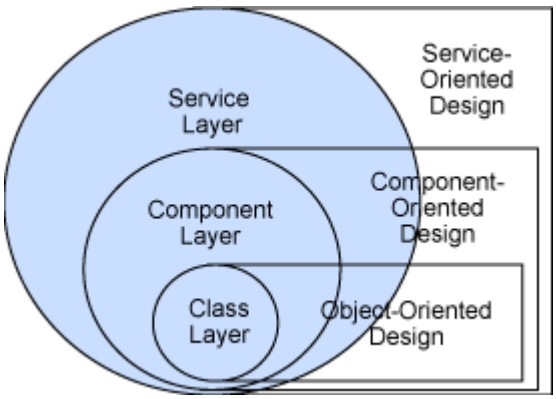


Рис. 4: Уровни проектирования

Говоря о нотации - Unified Modeling Language (UML), будучи улучшенным несколькими дополнительными стереотипами и профилями, вполне естественным образом становится ключевым элементом SOAD.

В процессной области Rational Unified Process (RUP) - признанная OOAD-методология поддержки анализа и проектирования при итеративной разработке ПО, использование UML-моделей высокопродуктивно. Однако, в основе RUP лежат принципы OOAD, поэтому RUP не так легко приспособить к использованию в ходе SOA-проектирования. С точки зрения RUP, архитектурой системы является структура ее основных компонент, взаимодействующих через определенные интерфейсы. Эти компоненты, в свою очередь, состоят из более мелких

компонент и так далее до степени детализации на уровне классов. Архитектура системы в SOA, напротив, состоит из не имеющих состояния (stateless), полностью инкапсулированных и самодостаточных сервисов, которые могут быть объединены общим понятием *бизнес-сервиса* и с высокой степенью точности маппированы в BPM, как это показано на Рис. 5.

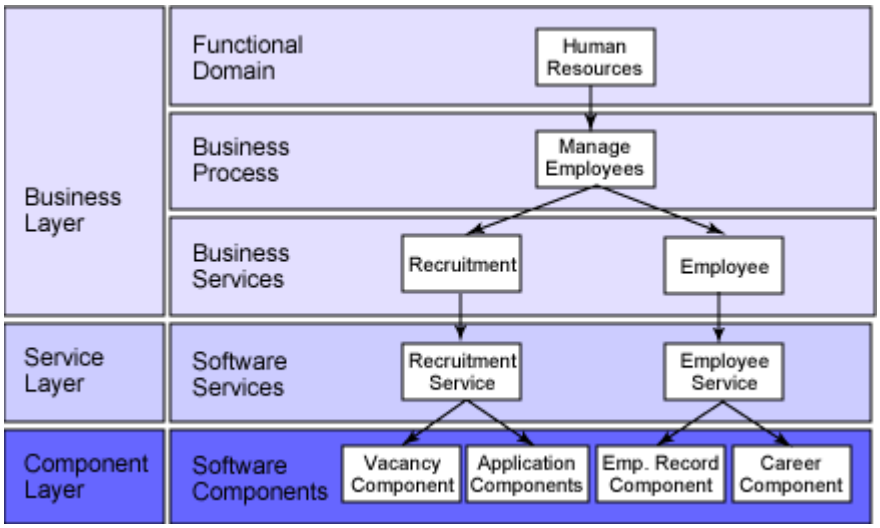


Рис. 5: Иерархия сервисов в SOAD

Данные сервисы формируются из набора совместно функционирующих или оркестрованных сервисов. Это не исключает OO-подход, принятый в RUP, однако вводит иной уровень абстракции. Данный "супер-уровень" служит для инкапсуляции компонентов, спроектированных в виде артефактов RUP (*программных сервисов*), внутри формальной кроссуровневой структуры интерфейса.

Элементы SOAD

В этом разделе мы рассмотрим требования, предъявляемые к SOAD, более подробно и попробуем определить его основные понятия и элементы. Нашей целью будет являться инициация дискуссии по вопросам формализации SOAD с целью определения направлений дальнейшей работы.

Что дает SOAD?

Для SOAD можно сформулировать следующие требования:

- Для SOAD должны быть формально (максимально формально) определены *методология* (*применения SOAD - прим. перев.*) и *нотация*, как это сделано для других методологий проектирования. SOAD не должен начинаться с "чистого листа": комбинируя нужным образом элементы OOAD, BPM и EA, можно определить основные элементы SOAD.
- SOAD должен иметь структурированный метод концептуализации сервисов:
  - Использование OOAD позволяет определить классы и объекты на уровне приложения, в то время как с помощью BPM разрабатываются событийные модели. SOAD же позволяет объединить эти результаты в единое целое.
  - Метод концептуализации сервисов SOAD должен быть ориентирован не на варианты использования, а на бизнес-события и процессы.

Разработка моделей вариантов использования должна являться вторым шагом, выполняемым на более низком уровне абстракции.

- Метод концептуализации должен включать в себя синтаксис, семантику и политики использования. Это необходимо для построения специализированных конструкций моделей, обеспечения семантических переходов между ними и runtime discovery (*динамический поиск сервиса по заданному описанию (например, с помощью WSDL-файла) в UDDI-реестре, привязка к нему и вызов для исполнения - прим. перев.*).

- Для SOAD должны быть сформулированы четкие критерии качества и лучшие практики (дающие, например, ответ на вопрос о степени детализации моделей). Должен быть также решен вопрос о ролях, поднимаемый в BPEL, а именно: каковы сферы ответственности для каждой из ролей (например, для ролей Разработчика, Архитектора и Аналитика)?
- SOAD также должен давать ответ на вопрос: что не является "правильным" сервисом? Например: объект, не обладающий свойством повторности использования, не может являться объектом рассмотрения SOA (т. е. сервисом). Другим примером являются нефункциональные требования систем реального времени, которые не порождают каких-либо накладных расходов при обработке XML-контента.
- SOAD должен способствовать сквозному моделированию (*т. е. созданию моделей на всех уровнях абстракции системы - прим. перев.*) и обладать обширной инструментальной поддержкой. Если предполагается, что SOA привносит в бизнес гибкость и быстроту адаптации, то же самое ожидается и от поддерживающего SOA метода, связывающего бизнес-область с областями разработки архитектуры и проектирования приложений.

Критерии качества

В настоящее время уже можно определить некоторые общие принципы и критерии качества, которые могут послужить основой процесса проектирования при использовании SOAD:

- Корректно созданные сервисы привносят в бизнес гибкость и быстроту адаптации. Они способствуют простоте реконфигурирования и повторности использования систем посредством слабого связывания, инкапсуляции и сокрытия информации.
- Правильно спроектированные сервисы (количество зависимостей между которыми минимизированы, а имеющиеся - четко прописаны) применимы не только в корпоративных приложениях.
- Абстракции сервисов должны образовывать единое целое, должны быть закончены и непротиворечивы. Например, при проектировании сервисов и их операций можно вспомнить о методе *Create, Read, Update, Delete and Search (CRUDS)*.
- Часто высказывается мнение, что сервисы (например, не являющиеся диалоговыми (*т. е. не использующие при взаимодействии диалоговый тип обмена сообщениями - прим. перев.*)) не имеют состояния как объекты (stateless services). Данное утверждение должно быть ослаблено по отношению к сервисам в том смысле, что они могут не иметь состояния настолько, насколько это возможно в конкретной проблемной области или в конкретном контексте.
- Система имен сервисов должна быть понятной экспертам предметной области, не обладающими глубокими техническими знаниями.
- В SOA все сервисы должны следовать единой, легко идентифицируемой (например, с помощью используемых в ходе проектирования шаблонов и образцов) философии проектирования и использовать одни и те же шаблоны взаимодействия.

- При разработке сервисов, а также при использовании их потребителями услуг в дополнение к знанию предметной области должны быть необходимы лишь базовые навыки программирования. Экспертные знания в области промежуточного ПО (ПО промежуточного уровня, middleware) должны быть необходимы только незначительному числу специалистов, работающих в компаниях-вендорах, занимающихся разработкой инструментария и IDE для создания веб-сервисов, а не в компаниях, создающих корпоративные SOA-приложения.

Идентификация и определение сервиса

Техники моделирования на бизнес-уровне, исповедующие принцип "сверху вниз", например CBM, могут служить отправной точкой при моделировании в SOA. Но, как уже отмечалось, реализация SOA редко начинается с "чистого листа" - создание SOA-решения почти всегда заключается в интеграции унаследованных систем путем их декомпозиции на сервисы, операции, бизнес-процессы и бизнес-правила (см. также Рис. 6):

- Существующие приложения и ПО вендоров декомпозируются на наборы отдельных сервисов, каждый из которых представляет собой группу связанных операций (подход "снизу вверх").
- Бизнес-процессы и бизнес-правила формулируются путем абстрагирования при изучении приложений в отдельных BPM, управляемых бизнес-моделью хореографии сервисов.

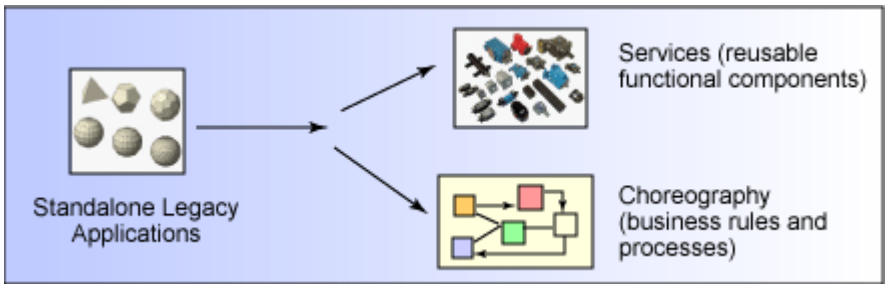


Рис. 6: Декомпозиция сервиса

Для идентификации и определения сервисов могут применяться любые OOAD-техники. Важно, однако, использовать и представление о системе более высокого уровня абстракции. Кроме того, поскольку проекты построения SOA-архитектур претендуют на большее, чем быть просто классическим проектом разработки, в ходе их реализации всегда существует необходимость в применении творческого подхода.

Прямой и непрямой бизнес-анализ

BPM и *прямой* анализ требований посредством интервью с акционерами, а также CBM являются очевидными и хорошо зарекомендовавшими себя способами идентификации сервисов.

Предыдущий опыт показывает, что вышеуказанные способы следует усовершенствовать *непрямыми* техниками анализа. Когда требуется идентифицировать сервисы, необходимо проинтервьировать продакт-менеджеров и других бизнес-лидеров предприятия. Но, например, как должны быть реализованы планирование платежей и система биллинга? Необходимо рассмотреть организационную структуру предприятия, которую предполагается заложить при проектировании системы. Необходимо также принять во внимание все созданные в ходе не-SOA-проектов модели вариантов использования.

Терминология, используемая в маркетинговых презентация, посвященных проектируемой системе, может является еще одним источником информации для идентификации сервисов (в частности, можно использовать глаголы; наречия же, используемые в маркетинговых презентация, в этом смысле дадут не много!).

Декомпозиция предметной области

Декомпозиция предметной области, анализ подсистем, создание целевой модели и другие подобные техники, как было отмечено в Endrei, являются первыми и самыми многообещающими попытками построения метода структуризации процессов SOA или *методологии концептуализации сервисов*, основанными на ранних работах Levi и Arsanjani. В развитие данной темы также внесла свой вклад работа FODA SEI.

Степень детализации моделей сервисов

Правильный выбор уровня абстракции является ключевым моментом при моделировании сервисов. Часто можно услышать совет *моделировать до уровня крупных блоков*. Это некоторое чрезмерное упрощение. Данное утверждение необходимо перефразировать следующим образом: *моделировать настолько крупноблочно насколько это возможно*, однако без потери важных элементов и обеспечивая непротиворечивость и полноту моделей. При этом при необходимости всегда имеется возможность разработки детальных абстракций сервисов. Поскольку SOA не эквивалентна веб-сервисам и SOAP, для получения доступа к сервисам на различных уровнях абстракции могут использоваться различные протокольные привязки. Другим вариантом является связывание нескольких взаимодействующих сервисов в крупные блоки, служащие одним из вариантов "фасадных" архитектурных шаблонов.

Соглашения об обозначениях

Необходимо определить корпоративные соглашения об обозначениях (пространства имен XML, систему имен Java-пакетов, Интернет-доменов). В этой связи можно рекомендовать следующее: всегда обозначать сервисы существительными, а их операции глаголами. Подобное правило является отличной практикой, введенной еще в ходе применения OOAD.

Первые элементы SOAD

Кроме осознания необходимости использования при формализации SOAD комбинации OOAD, BPM и EA, существует также несколько важных понятий и аспектов SOAD, которые пока не получили своего смыслового наполнения:

- Классификация и агрегация сервисов
- Политики и аспекты использования
- Сходящиеся процессы
- Семантические переходы
- Сборка сервисов и передача знаний

Классификация и агрегация сервисов

Сервисы имеют различное назначение и могут использоваться различным образом: например, программные сервисы отличны от бизнес-сервисов (как ранее было показано на

Рис. 5). Кроме того, отдельные сервисы могут быть оркестрованы (собраны) в законченные сервисы более высокого уровня.

Композиция сервисов упрощается исполняемыми моделями (например, выполненными в BPEL) - это то, с чем не оперируют традиционные средства и методы моделирования.

Политики и аспекты использования

Сервис имеет синтаксические и семантические характеристики, а также характеристики качества сервиса (QoS), которые должны учитываться в моделях. В то же время формальные интерфейсные контракты взаимодействия сервисов представляют собой нечто большее, чем может описать язык Web Services Description Language (WSDL). В этой связи спецификация WS-Policy является весьма важной в качестве дополнения к WSDL.

Желательным свойством, наряду с хорошо известным принципом *архитектурной трассируемости*, является *бизнес-трассируемость* (*в оригинале - business traceability; traceability, трассируемость - финансовый термин, означающий возможность отнесения затрат на конкретную операцию или объект затрат - прим. перев.*): должна быть возможность прямой привязки всех возникающих в ходе разработки системы артефактов к языку, понятному неэксперту предметной области. Это особенно важно для абстракций, напрямую используемых на бизнес- и сервисном уровне. Закон Сарбейнса-Оксли (SOX) (*англ. Sarbanes-Oxley act - закон, согласно которому компании, акции которых котируются на фондовых биржах, регулируемых Комиссией по ценным бумагам и биржам, должны вести финансовую отчетность в соответствии с общепринятыми принципами бухгалтерского учета - прим. перев.*) (см. статью Astor) является примером, для реализации которого и может потребоваться бизнес-трассируемость.

Процесс моделирования: сходящийся

В реальном мире не существует проектов, выполняемых с "чистого листа", поскольку всегда должны приниматься во внимание *унаследованные приложения* (термин унаследованные приложения является синонимом для термина "приложения, существующие в настоящее время"). Поэтому, вместо применения в чистом виде подходов к проектированию по принципам "сверху вниз" или "снизу вверх", необходимо применение так называемого *сходящегося* подхода.

Подход к проектированию по принципу "снизу вверх" может привести к бедности абстракций бизнес-сервисов в случае, когда при проектировании руководствуются существующей IT-средой, а не имеющимися и будущими бизнес-нуждами предприятия. С другой стороны, подход "сверху вниз" может породить недостаточные либо нефункциональные характеристики требований, а также подвергнуть опасности показатели других критериев качества архитектуры (например, проблемы производительности - ввиду недостатка нормализации (*имеется ввиду терминологическая и понятийная стандартизация предметной области - прим. перев.*) модели предметной области) наряду с проблемами, ведущими к несогласованности на сервисном или компонентном уровнях.

Семантические переходы

В любой SOA для синтаксиса вызовов важно наличие формального интерфейсного соглашения. При моделировании предметной области также должны быть решены вопросы семантики (значение параметров и т. д.). Ответы на эти вопросы являются ключевыми при разработке любого B2B-сценария или сценария динамического вызова. Данные сценарии

являются краеугольными понятиями концепции SOA, которая привносит гибкость и быстроту адаптации как ответ на новые бизнес-нужды в мире слияний, поглощений, бизнес-реструктуризаций и глобализации.

Сборка сервисов и передача знаний

Данный вопрос относится к области управления знаниями и концепции жизненного цикла сервиса: как правильно разработать сервисы и сделать их доступными для повторного использования сразу после концептуализации?

Сервисы необходимо идентифицировать и описывать, имея ввиду один из главных критериев SOA - свойство повторности использования (и возможность сборки). Если компонент (или сервис) не может быть повторно использован, он не должен представляться как сервис. Он может быть связан с другим сервисом архитектуры предприятия, но не будет сервисом в полном смысле этого слова.

Однако, даже если свойство повторности использования закладывается с самого начала, методология проектирования должна иметь формализованную методологию сборки сервисов. Использование сервиса несколькими клиентами - основная цель проектирования в SOA (например, при создании реестра сервисов - UDDI-реестра масштаба предприятия).

Пример: Наряд на выполнение авторемонтных работ

Предметной областью данного примера является процесс управления оказанием клиентам компании услуг по техобслуживанию автомобилей. Мы приводим этот пример для иллюстрации обсуждающихся в этой статье вопросов, касающихся SOAD.

Наряд на выполнение авторемонтных работ представляет собой соглашение между автосервисной компанией и клиентом, согласно которому автосервисная компания выполняет ряд плановых или аварийных мероприятий по техобслуживанию автомобиля клиента, таких как плановое техобслуживание по достижении пробега в 50000 миль, смена тормозных колодок, замена покрышек или масла.

Бизнес-сценарий (показанный на Рис. 7) в этом случае следующий:

- Наряд на выполнение работ создается по факту обращения (телефонному звонку) клиента.
- Для каждой плановой работы или операции в наряде на выполнение работ создается отдельный пункт, содержащий детальное описание необходимых для ее выполнения запчастей, материальных и трудовых ресурсов.
- Прежде чем назначить клиенту встречу, составленный список запчастей проверяется для уверенности в том, что все необходимые запчасти есть на складе.
- Для выполнения наряда назначается специально оборудованный бокс и выделяется квалифицированный механик.
- Вычисляется и сообщается клиенту ориентировочная стоимость выполнения наряда, клиент подтверждает назначенную встречу либо отменяет ее, в последнем случае наряд на выполнение работ аннулируется.
- К приезду клиента необходимые запчасти и инструменты доставляются в бокс, оборудование собирается и монтируется.
- По прибытию клиента выполняются запланированные работы, а также иные работы, необходимость в которых выясняется при осмотре автомобиля.

- По факту выполнения работ производится учет фактического расхода запчастей, материальных и трудовых ресурсов.
- По завершению всех работ определяется общая стоимость их выполнения.
- Формируется и передается клиенту счет для оплаты выполненных работ.

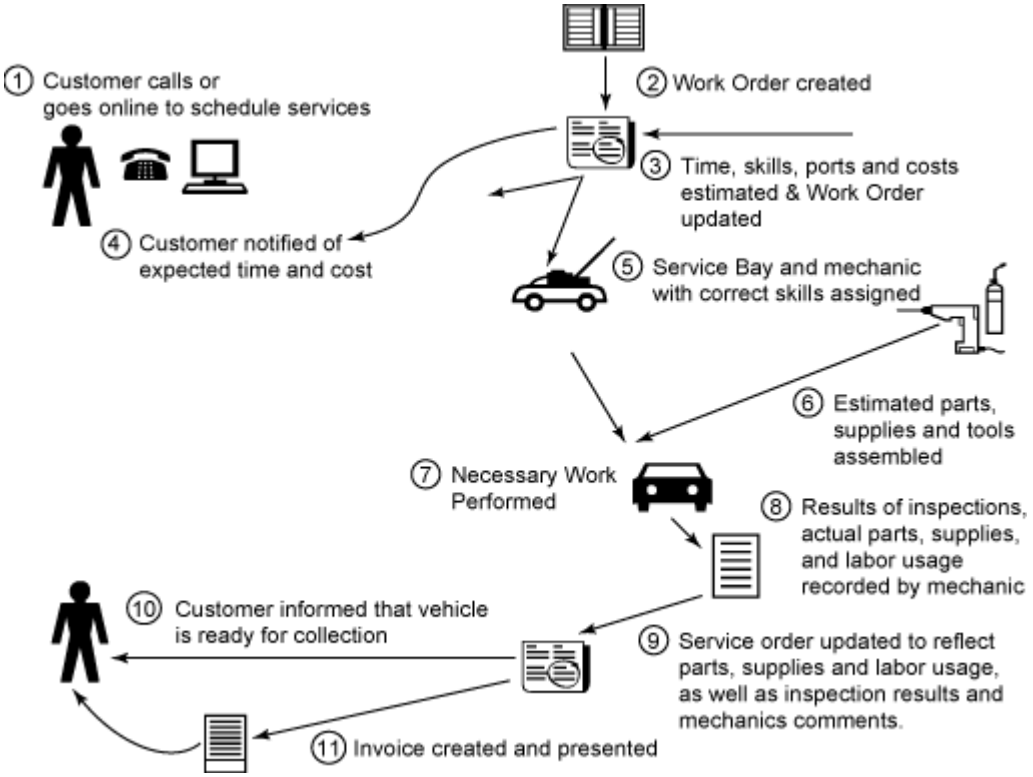


Рис. 7: Укрупненное представление примера

Если вы попробуете создать приложение, реализующее вышеописанный сценарий, может получиться набор классов, подобный показанному на Рис. 8.

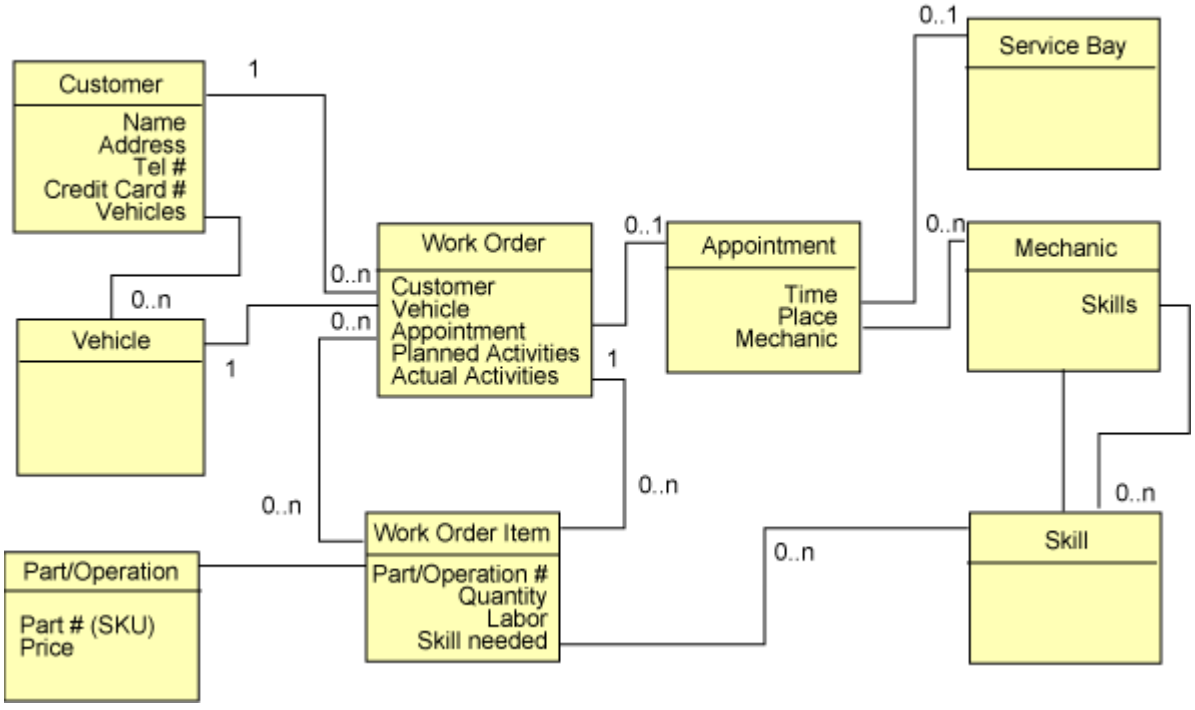


Рис. 8: Диаграмма классов приложения, реализующего пример

Если приложение, реализующее пример, будет создаваться как ОО-приложение, программные объекты должны содержать все необходимые бизнес-правила и описывать осуществляемые бизнес-процессы.

Однако, у данного подхода существует несколько практических недостатков:

- На многих шагах приведенного примера осуществляется взаимодействие существующих унаследованных систем с базами данных (например, биллинговых систем, систем планирования и инвентаризации), которые, скорее всего, не были спроектированы с учетом ОО-парадигмы (в подобных ситуациях может помочь применение adapter pattern или mediator pattern).
- Чтобы сделать систему максимально гибкой, необходимо, чтобы процессы или потоки работ могли быть модифицированы без внесения изменений в программный код. Это может обеспечить соблюдение следующих правил:
    - Для осуществления стандартного сервисного техобслуживания по достижении пробега в 24000 миль требуется 4 литра масла. Дополнительные объемы масла должны использоваться только в случае превышения данной нормы или в случае, если клиенту требуется масло высшего качества (например, синтетическое масло).
    - Существует ряд стандартизированных для каждой отрасли норм трудозатрат, необходимых для выполнения работ по техобслуживанию автомобилей, которые можно почерпнуть из унаследованных систем. Пока норма трудозатрат не будет превышена на X%, клиенту должен выставляться счет согласно стандартной стоимости человеко-часа. Только в случае превышения нормы появляется основание для расчета дополнительной оплаты работ.
    - Если трудозатраты на техобслуживание превышают плановые на Y%, клиент должен своевременно оповещаться об этом.

SOAD прекрасно решает данные проблемы. Поскольку согласно данному подходу сервисы группируются по признаку совместности функционирования, а не наличия свойства инкапсуляции (функционирование плюс данные), перечень сервисов будет несколько отличаться от модели бизнес-объектов.

Например, вероятно следует объединить Work Order (Наряд на выполнение работ) и Work Order Item (Пункт наряда на выполнение работ) с Work Order Services (Услуги, оказываемые согласно наряду на выполнение работ) и создать сервисы Scheduling (Планирование), Catalog (Каталог) и Inventory (Список). Поскольку не существует экземпляров сервисов, не существует и эквивалента отношений между сервисами. Модель сервисов для нашего примера показана на Рис. 9.

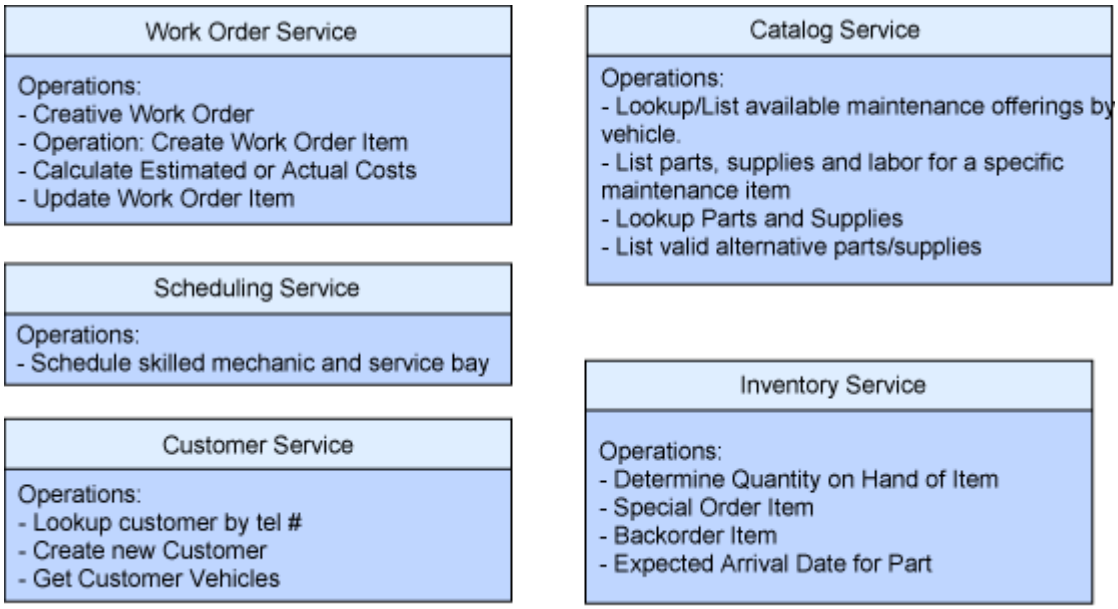


Рис. 9: Модель сервисов для примера

В отличие от ОО-подхода, данная модель не описывает функциональную систему. Она не содержит ни потоков, ни описаний бизнес-событий или бизнес-правил. В SOA хореография бизнес-процессов определяет очередность и последовательность во времени выполнения вызовов сервисов.

Концептуально, весь бизнес, начиная с первого контакта с клиентом и заканчивая завершением работ и оплатой счета, на макро-уровне представляет собой одну логическую единицу работы с продолжительностью от нескольких дней до нескольких недель. Эта логическая единица работы и генерирует прибыль.

Однако, на практике мы имеем дело с целым спектром деятельностей (например, определение состава работ, планирование встречи, выбор необходимых запчастей и ресурсов, выполнение работ по техобслуживанию), которые осуществляются попеременно с относительно продолжительными периодами простоя. В IT-системе не существует реального *процесса*, который длился бы больше нескольких минут, при этом состояние бизнес-процесса характеризуется состоянием данных в базе данных между событиями.

Процессы подобного типа могут быть адекватно описаны с помощью модели перехода состояний (state transition model) (выполненной, например, в UML). Рис. 10 демонстрирует пример того, как можно смоделировать бизнес-поток с использованием метода конечных автоматов. Данная иллюстрация является общим представлением того, как в ходе осуществления бизнес-процесса может меняться состояние наряда на выполнение работ.

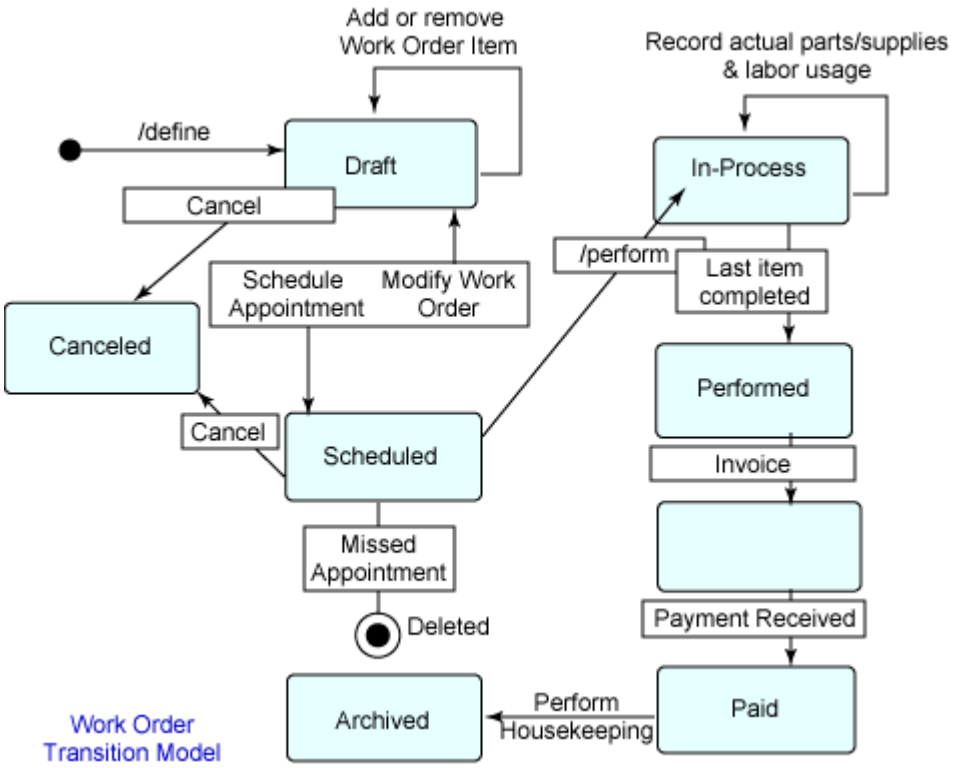


Рис. 10: BPM для примера

Хореография бизнес-процессов фокусируется на транзакциях между состояниями. Отдельные операции отражают соответствующие изменения состояния.

Рис. 11 показывает пример хореографии бизнес-процесса, охватывающей шаги 1-5 бизнес-сценария, показанного на Рис. 10 (например, клиент определяет работы, которые необходимо выполнить для его автомобиля и назначает встречу для выполнения работ).

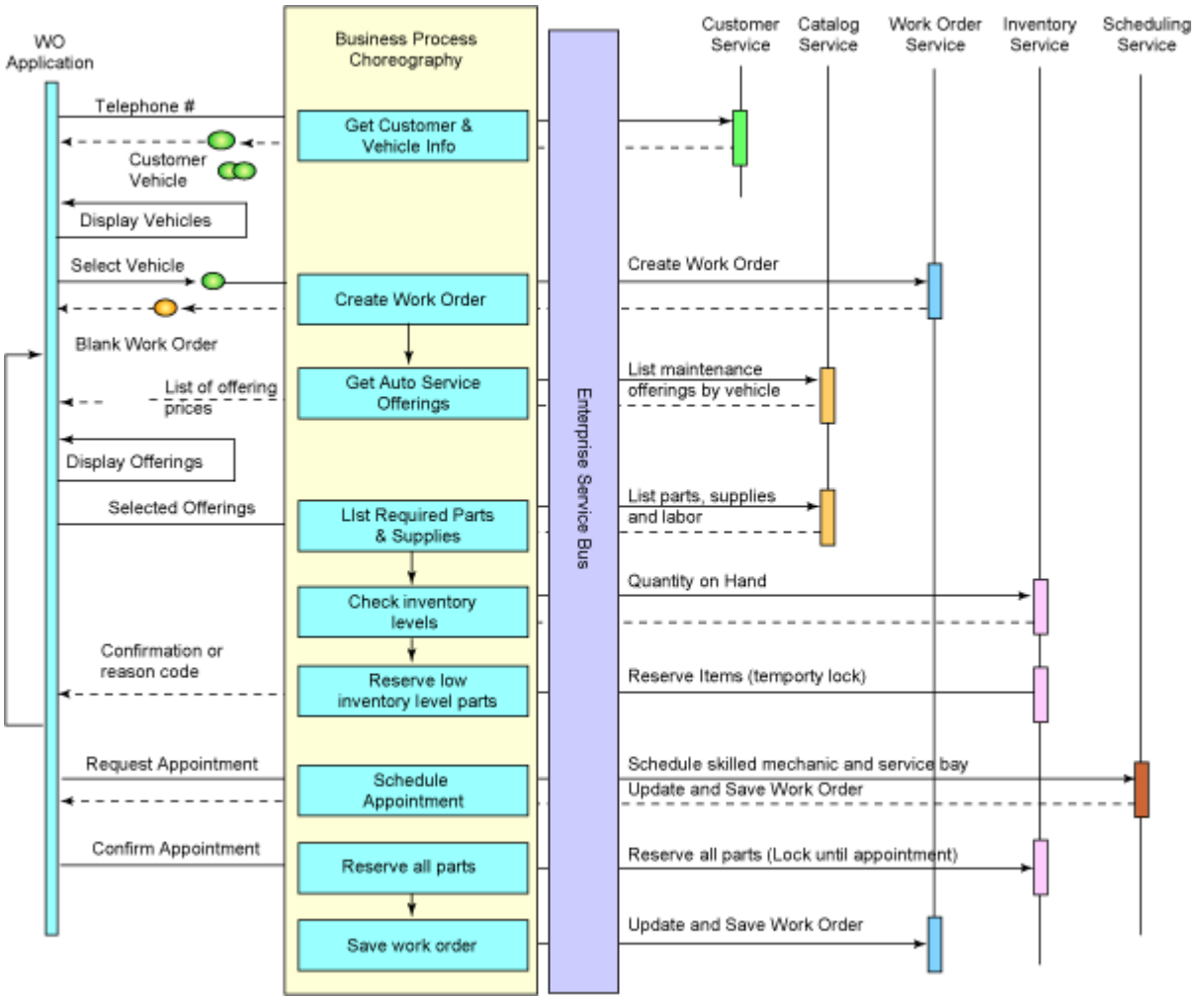


Рис. 11: Модель бизнес-взаимодействия для примера

Выводы и перспективы на будущее

В этой статье мы обсудили и обозначили необходимость разработки нового, сходящегося подхода к анализу и проектированию SOA-систем, служащего неким мостом между бизнесом и IT. Мы также сделали предположение, что этот новый междисциплинарный SOAD-подход должен являться целостной методологией моделирования, строящейся на современных, проверенных и доказавших свою эффективность подходах OOAD, EA и BPM.

В то время как нотация и методология применения SOAD еще только должны будут детально описаны, ключевые элементы подхода, такие как концептуализация (или определение) сервисов, их классификация и агрегация, политики и аспекты использования, сходящийся подход к моделированию, семантические переходы и сборка сервисов (для повторного использования), уже могут быть определены.

SOAD потребует улучшений существующих методов программной инженерии, направленных на дальнейшее повышение удобства их использования и расширения возможностей применения в проектах разработки корпоративных приложений. Через некоторое время будут разработаны и лучшие практики и шаблоны.

Очевидно, что UML будет и в дальнейшем доминировать в качестве нотации моделирования процессов, однако, вероятно, потребуется его улучшение для использования в SOAD с его более широкой предметной областью.

Следующими шагами на пути формализации SOAD будет определение необходимой сквозной методологии применения, нотации, необходимых ролей, их обязанностей и сфер ответственности, а также утверждение предлагаемого подхода в ходе реализации проектов.

Ресурс

Olaf Zimmermann, Старший IT-архитектор (Senior IT Architect), IBM Enterprise
Pal Krogdahl, Системный архитектор (Solution Architect), IBM
Clive Gee, Старший архитектор решений (Senior Solution Architect), IBM

Оригинальный текст статьи можно найти
http://www-106.ibm.com/developerworks/webservices/library/ws-soad1/

Данная статья размещена по адресу http://www.ubs.ru/ws/ws_soad1.html