Джеймс Бин

Предисловие – Александр Фалк
Президент и Генеральный директор
компании Altova, создателя xmlspy

XML для ПРОЕКТИРОВЩИКОВ

ПОВТОРНОЕ ИСПОЛЬЗОВАНИЕ И ИНТЕГРАЦИЯ

КУДИЦ-ОБРАЗ

James Bean
Relational Logistics Group and Global Web Architecture Group

XML FOR DATA ARCHITECTS

DESIGNING FOR REUSE AND INTEGRATION

MORGAN KAUFMANN PUBLISHERS
AN IMPRINT OF ELSEVIER SCIENCE

Amsterdam Boston Heidelberg London New York Oxford Paris San Diego
San Francisco Singapore Sydney Tokyo

XML для проектировщиков

Повторное использование и интеграция

Джеймс Бин

КУДИЦ–ОБРАЗ
Москва • 2004

ББК 32.973-18

Д. Бин

XML для проектировщиков. Повторное использование и интеграция. – М.: КУДИЦ-ОБРАЗ, 2004. – 256 с.

Данная книга посвящена вопросам проектирования XML-структур, ориентированных на повторное использование.

Необходимость в XML обусловлена широчайшими возможностями этого метаязыка для организации работы с независимыми от платформы данными и интеграции корпоративных приложений. Невероятно мощные современные технологии, ориентированные на электронный бизнес, такие как ebXML и Web-сервисы, немыслимы без XML. И, несмотря на такую значимость XML, существует крайне мало изданий, затрагивающих вопросы проектирования эффективных, удобных и пригодных для повторного использования XML-языков. Этот пробел восполняет данная книга.

Также в этой книге детально рассмотрены XML-Схемы, приведены справочные таблицы соответствия типов данных различных СУБД типам данных XML-Схем и, в завершении, обозреваются основные технологии мира Web-сервисов – SOAP, WSDL и UDDI. Книга ориентирована на проектировщиков моделей данных и разработчиков приложений, использующих XML.

ISBN 5-9579-0043-5
ISBN 1-55860-907-5

Джеймс Бин

XML для проектировщиков. Повторное использование и интеграция

Учебно-справочное издание

Переводчик: В.В. Акимов

Научный редактор: А. Л. Соколов, инструктор Учебного центра IBM

ООО «ИД КУДИЦ-ОБРАЗ» 119049, Москва, Ленинский проспект, д. 4, стр. 1А.
Тел.: 333-82-11, ok@kudits.ruhttp://books.kudits.ru

Подписано в печать 12.08.2004.
Формат 70×90/16. Бумага газ. Печать офсет.
Усл. печ. л. 18,72. Тираж 2000. Заказ 1716

Отпечатано в ОАО «Щербинская типография» 117623, г. Москва, ул. Типографская, д. 10

ISBN 5-9579-0043-5 © ООО «ИД КУДИЦ-ОБРАЗ», 2004

ISBN 1-55860-907-5

Предисловие

Вызовы, с которыми сталкиваются проектировщики данных в современном, быстро меняющемся мире, умножаются. Они являются прямым следствием возрастающей сложности, обусловлены необходимостью взаимодействия самых разных систем и возникновением новых технологий. С появлением в 1998 году языка XML и в 2001 году XML-Схемы к этой смеси добавился еще один вызов проектировщикам данных. Не сразу стало понятно, какое место эти технологии займут в общем потоке, поскольку они представляли парадигму, радикально отличающуюся от традиционных реляционных моделей данных.

XML – расширяемый язык разметки – первая технология, представляющая последовательный, результативный и расширяемый механизм для описания данных и метаданных в одном и том же документе. На первый взгляд все выглядело как достаточно простое расширение существующего, но оказалось, это более глубокая технология, поскольку она позволяет получать самоописывающие данные, упрощает реализацию интероперабельности и дает возможность хранить самые разные виды данных, такие, как текстовые документы, реляционные данные, объектно-ориентированные структуры и иерархическую информацию.

В результате, всего за несколько лет XML стал всеобщим языком Интернета, он используется в таких разных прикладных областях, как археология (например, http://www-oi.uchicago.edu/OI/PROJ/XSTAR), регистрация налогов (например, http://xml.coverpages.org/irs.html), генетика (www.fruit-fly.org/sequence) и во многих других. XML быстро стал необходимым инструментом для описания, хранения и преобразования данных, когда аналогичных результатов нелегко добиться в рамках реляционной модели.

Из-за гибкости и иерархической природы XML требует от проектировщика данных овладения дополнительными навыками. Эта книга предоставляет читателям необходимые знания и техники. James Bean, опытный эксперт по проектированию баз данных, архитектуре и всем аспектам XML, поэтапно дает вам необходимые сведения по XML, приводит информацию о типах документов и XML-Схемах, о проектировании архитектуры XML с ориентацией на повторное использование.

Одним из главных стандартов, открываемых в этой книге, является W3C XML-Схема, предоставляющая существенные возможности для описания структуры XML-документов, а также для введения ограничений на данные, содержащиеся в отдельных XML-элементах. Это осуществляется с помощью так называемых *простых типов (simpleType)*, которые похожи на типы данных, используемые в реляционных базах данных, и одна из глав книги посвящена связи между типами баз данных и простыми типами XML-Схемы, а также *сложных типов (complexType)*, описывающих порядок следования элементов и вложенность XML-элементов друг в друга.

Благодаря этим возможностям XML-Схема является стержневой технологией для проектировщиков данных, и эти функции широко освещаются в данной книге, особенно в части стандартов именования, повторного использования и общего проектирования системы.

Важность XML-Схемы для проектировщиков данных также отражается в позитивной обратной связи, которую мы постоянно получаем от клиентов нашего xmlspy®, многие из которых используют инструмент графического проектирования XML-Схемы из этого продукта для проектирования сложных моделей данных XML точно так же, как они используют графические инструменты для создания ER- или UML-диаграмм для реляционных и объектно-ориентированных моделей данных.

Другой аспект, требующий рассмотрения в этом отношении, – широкое принятие XML большинством поставщиков программного обеспечения, такими, как Microsoft с ее средой .NET и продуктами Office 2003 или Oracle с поддержкой XML в Oracle 9*i*. В результате XML становится вездесущим: он и на персональных компьютерах, и в хранилищах данных, а переориентация существующих XML-активов позволяет организациям повысить производительность и эффективность. Все вышеперечисленное предоставляет проектировщику данных отчетливое понимание важности XML и уникальности его характеристик.

Все это объясняет, почему вы держите в руках данную книгу…

Alexander Falk
President&CEO
Altova, Inc.

Благодарности

Я благодарю издателей Morgan Kaufmann и, в частности, я хочу выразить признательность Lothlorien Hornet за энтузиазм, руководство и настойчивость, а также Corina Derman за постоянное содействие. Я также благодарю Kelly Mabie и сотрудников Graphic World. Я признателен членам моей семьи, партнерам по бизнесу, клиентам и друзьям, которые поддержали мою работу, в том числе: Sue Bean, Jack Bauer, David Bean, Lisa Bean ("SA"), Kimberly Bean ("Bug"), Beth Bauer, Norm Johnson, Sandy Johnson, Barb Wakefield, Dick Schreiber, Nick Torrez, Lara Tang, Gloria Michalak, Wally Sellman, Caren Shiozaki, Jerry Halterman, Mike Ruttledge, Deb Barra, Dennis Barra, Mike Nicewarner, Lori Gubernat, John Gubernat, Lorraine Cooper, Jerry Blidy, Dave Blidy, Renee Adams, Jackie Barkworth, Aprill Barnes, Tari Mattson, Kay Turner, Jarrod Reed, Bob Takoushian, John Sherrie, Patrick Vincent, Jorden Woods, Arka Mukherjee, Ph.D., Muralidhar Satyanarayana, и многим другим (вы знаете, кто вы).

Я также хочу выразить признательность следующим людям, технологическим компаниям и организациям по стандартизации технологий: Dr. Charles Goldfarb, Tim Burners-Lee, Alexander Falk, Altova, Microsoft, IBM, Sybase, Oracle, DAMA и W3C. Достижения этих людей и организаций привели к значительному развитию бизнеса и технологий и широкому распространению XML.

Введение

Традиционное предприятие сталкивается со значительным количеством технологических вызовов, последние из которых явились результатом появления в бизнесе новых парадигм (Интернет, Всемирная паутина, экономическое давление и глобальная электронная коммерция). Дополнительную сложность вносит применение традиционных методов разработки, создание вертикальных систем, недостаточная поддержка стандартов данных, приобретение различающихся бизнесов и технологий и реализация программных пакетов. Все эти вызовы требуют решений, включающих обмен данными между разными платформами и совместное использование данных предприятия.

Типичное современное бизнес-предприятие часто состоит из нескольких автономных подразделений. Зачастую каждое из этих подразделений и системы, обеспечивающие их функционирование, имеют собственное определение ключевых данных. Независимо от используемой технологии, бизнес-концепция, описывающая покупателя, поставщика или продукцию и обрабатываемая одной из этих систем, имеет тот же общий контекст, что и прочие системы. Однако в деталях характеристики данных этих бизнес-концепций обычно заметно отличаются. Такие характеристики данных известны как *метаданные*. Возможно, наиболее часто метаданные определяют как "данные о данных". И действительно, метаданные вполне соответствуют этому простому определению, поскольку включают формат, правила и ограничения, описывающие данные.

Расширяемый язык разметки (XML) приобрел широкую известность как технологическое решение сравнительно недавно. Его можно с успехом использовать в приложениях самого разного типа для переноса данных между различными бизнес-системами или для описания бизнес-транзакций, запроса к базе данных или запроса к серверу. Благодаря независимости от платформы, транзакции и обмен данными более не связаны с нестандартным или защищенным патентом интерфейсом точка-точка. Вместо того чтобы бурно воспринимать шум, поднятый в прессе вокруг XML, современный специалист в области технологий должен изучить его сильные и слабые стороны, а также понять, как и когда его можно эффективно использовать. XML обладает широкими возможностями, но он вовсе

не является универсальным средством, "серебряной пулей", позволяющей решить все проблемы. Использование XML требует существования определенного процесса разработки, строгого соблюдения стандартов, соединения с другими технологиями и практических навыков.

XML – это сравнительно новая (с 1997 года), но уже устоявшаяся технология. Что бы вы о нем не читали, XML – это, фактически, форма описывающих метаданных. По сути, XML – синтаксис для реализации описывающих контейнеров данных в документе, транзакции или сообщении. Схемы расширяют метаинформационные возможности XML, определяя правила и ограничения для характеристик данных, например их структуру, отношения, допустимые значения и типы данных. Когда информация предприятия, содержащаяся в XML-документе или транзакции, используется совместно разными системами предприятия и, возможно, внешними партнерами, можно довольно быстро убедиться в необходимости использования эффективных подходов к проектированию этих данных.

Данная книга создана для решения подобных вопросов, связанных с метаданными предприятия, но не ограничивается только ими. Существуют разные специфические вопросы, относящиеся к дисциплинам, связанным с метаданными, но наиболее часто люди, занимающиеся ими профессионально, носят звания "проектировщик данных", "аналитик данных", "администратор данных" и "модельер данных" (data modeler). Мир быстро меняется и мы, проектировщики данных, должны быть "агентами перемен". Как следует из возрастающего спроса на содействие в определении схем, XML представляет огромное поле деятельности для проектировщиков данных. Чтобы откликнуться на ожидания лидеров и повысить значимость проектирования данных, проектировщикам данных необходимо получить основательное понимание XML, его возможностей и ограничений. Для успешного использования XML как технологии метаданных проектировщики данных должны реформировать, адаптировать и заново оценить разные аспекты своей дисциплины применительно к XML и связанным с ним процессам. Если таких шагов не сделать, распространение XML на предприятии будет продолжаться, но ценность результатов окажется под вопросом.

Данная книга призвана дать необходимую для этого информацию. Проще говоря, те, кто отвечает за описание, определение и проектирование данных и структур данных, а также те, кто занимается обработкой транзакций и сообщений с данными, найдут в книге много полезного. В главе 1(Обоснованность и целесообразность применения XML) читатель посвящается в основы XML как технологии метаданных и знакомится с принципами его использования. XML не является универсальным решением "на все случаи жизни". Он может применяться в разных ситуациях, и есть ситуации, когда его применение не оправдано. Во многих случаях

XML используется для описания содержимого простых документов. Однако еще большие возможности XML предоставляет для описания данных транзакции или сообщения. В главе 2 (Типы XML-документов) описаны типы документов и применение XML для разных типов документов.

Важно отметить отличие между XML, который предназначен для описания содержимого данных, и схемой, содержащей детальные правила метаданных, применяемые к ссылающемуся на нее документу. С XML можно использовать несколько типов схем. У каждого типа свои достоинства и недостатки. Зачастую бывает трудно определить, какой из типов схемы обеспечивает наилучший результат в конкретном приложении XML. Во главе 2также содержится описание возможностей, достоинств и недостатков трех наиболее часто используемых типов схем. И хотя я привожу описание нескольких типов схем, мой опыт свидетельствует, что для содержимого транзакций и сообщений больше подходят развитые возможности XML-Схемы, соответствующей майским 2001 года рекомендациям комитета W3C. И в последующих главах основной упор делается именно на W3C XML-Схемы.

Главы с 3 по 7 посвящены применению и развитию подходов к метаданным применительно к XML и W3C XML-Схемам. Глава 3 (Необходимость стандартов именования) приводит подходы к построению имен для элементов и атрибутов XML, согласующиеся с практикой предприятия и усиливающие описательные возможности XML. Глава 4 (Сравнение типов W3C XML-Схемы с типами баз данных) описывает возможности XML по "строгой типизации", а также то, как поддерживаемые в нем типы данных соотносятся с аналогичными типами данных наиболее распространенных СУБД. Как известно, для исчерпывающего описания значений данных, недостаточно только типов данных. Здесь также важны дополнительные ограничения, такие, как длина, число десятичных знаков и допустимые значения. Глава 5 (Фасеты типов данных W3C XML-Схемы) описывает эти и другие ограничения (известные как "фасеты"), которые могут применяться к типам данных.

Модели данных, такие, как диаграммы отношений сущностей (ERD), – одна из наиболее важных для проектировщика данных областей деятельности. Подобные модели позволяют определить и наглядно представить структуры данных и их взаимосвязи. Глава 6 (Структурные модели) проводит параллели между традиционными моделями данных и XML-структурами. Чрезвычайно важной формой моделирования является создание "прототипов" XML-транзакций. Любой вид проектирования должен обеспечивать гибкость и возможность повторного использования результата. И то, и другое достигается определением и последующим применением в модели обобщенных шаблонов. Глава 7 (Архитектурные формы контейнеров) расширяет моделирование применением в моделях шаблонов проектирования.

Повторное использование, потенциально, может принести большую пользу в разработке приложений. Однако это также и одно из наиболее запутанных понятий. Во многих технологиях разработки, применяемых на предприятии, повторное использование практикуется явно недостаточно, и даже сама концепция повторного использования часто определяется не аккуратно. Повторное использование метаданных, и формирование стандартов на основе метаданных представляют широкое поле деятельности для проектировщиков данных. Глава 8 (W3C XML-Схемы и повторное использование) описывает парадигму повторного использования и представляет возможности эффективного повторного использования W3C XML-Схемы.

С быстрым принятием и распространением XML смещается соотношение традиционных ролей, областей ответственности и опыта, принятых в технологии. В своем желании использовать мощь XML разработчики приложений часто считают, что XML должен находится в их исключительной зоне ответственности. Однако во многих случаях эти разработчики не обладают достаточными знаниями в областях, связанных с метаданными. Получая кратковременный выигрыш, они приводят к долговременным проблемам, связанным с размножением нестандартных XML-транзакций и схем. Глава 9 (Проектирование и разработка для проектировщиков данных) рекомендует совместную работу, в которой проектирование XML-структур и разработка схем являются областью совместной ответственности разработчиков приложений и проектировщиков данных. Там же читатель знакомится с практическими подходами к проектированию и разработке.

Хотя процессы, подходы и приемы из предыдущих глав относятся к данным транзакций, во многих случаях они применимы и к содержимому документов и сообщений. XML также играет заметную роль в набирающей силу совместной обработке (collaborative computing) и, в частности, в "Web-сервисах". Глава 10 (Web-сервисы. Введение в будущее) представляет концепцию Web-сервисов и описывает использование XML для описания сообщений.

Повсюду в книге приводятся примеры XML и W3C XML-Схем. Эти примеры, по возможности, относятся к важным концепциям данных, общим для большинства отраслей. В некоторых случаях я варьирую применяемые подходы к именованию, а также структуры и формы, чтобы продемонстрировать, что существует несколько вариантов применяемых правил и каждое предприятие может адаптировать и применять свои собственные процессы и стандарты для XML.

Для удобства читателя важные концепции каждой из глав выделены в виде Фактов, Рекомендаций, Техник и Возможностей. *Факты* основываются на доступных источниках или наблюдениях. Несколько субъективные *Рекомендации* – результат исследований и опыта в качестве проектировщика данных. *Техники* приводят конкретные тактические приемы. *Возможности* позволяют читателю решить, стоит ли применять определенный подход для улучшения результата или чтобы избежать потенциальных ошибок.

В дополнение к прочим источникам приведенные концепции собраны в Приложении А и снабжены ссылками на их обсуждение в тексте. В Приложении Б приведены примеры основных синтаксических конструкций W3C XML-Схем. Эти фрагменты синтаксиса помогут проектировщику данных разобраться с наиболее часто используемыми контейнерами, структурами и объявлениями ограничений. В сочетании с информацией, содержащейся в главах, это формирует базовое руководство, которое окажется полезным проектировщикам данных и представителям других технологических дисциплин.

Глава 1

Обоснованность и целесообразность применения XML

Перед тем как приступить к рассмотрению того, почему так важен расширяемый язык разметки (eXtensible Markup Language – XML) и в каких случаях его необходимо использовать, у вас может возникнуть вопрос о его происхождении. XML начинался как "согласованное подмножество" стандартного обобщенного языка разметки (Standart Generalized Markup Language – SGML). SGML в течении нескольких лет использовался для описания и наложения ограничений на содержимое текстоориентированных файлов и документов. В качестве примеров текстоориентированных данных можно привести страницы книги, письма, заметки, брошюры и презентации. Такие документы состоят из строк символов и слов, из которых складывается содержание (чаще их рассматривают как совокупность фраз, предложений, параграфов, разделов и глав). Для слабоструктурированной информации подобного типа редко возникает необходимость в описании столь мелких единиц информации, как отдельные слова, а тем более потребность в описании еще более специфических характеристик – типов данных. Применение здесь SGML заключается в использовании описательных тегов для выделения разделов текста и описания текстовых характеристик и характеристик всего документа в целом. И хотя значение SGML для описания документоориентированных данных неоспоримо, его применение в других областях, таких, как глобальная электронная коммерция (e-commerce) или обмен корпоративными данными, проблематично. Во-первых, SGML отличается сложным синтаксисом, трудным в изучении и использовании. Он также не обладает необходимой поддержкой строго типизированных данных (проектировщики данных узнают "строго типизированные данные" по наличию "типа данных" для каждого элемента данных, столбца, поля, или атрибута).

Данные, содержащиеся в транзакциях электронной коммерции или обмена корпоративными данными, отличаются от текстоориентированного содержимого. В центре внимания здесь находятся отдельные кусочки информации, обрабатываемые

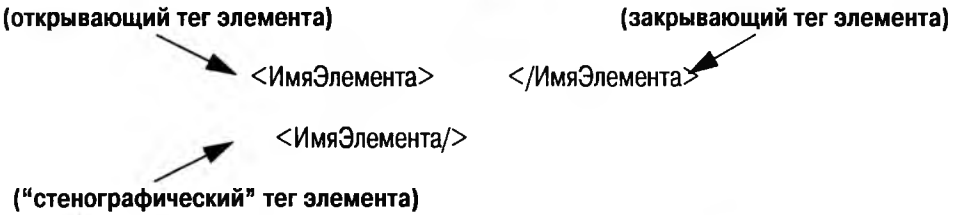
и используемые попеременно. За редким исключением, большинство транзакций электронной коммерции содержат отдельные элементы данных, а не длинные строки текста, например предложения или параграфы. К типичным данным, содержащимся в этих транзакциях, относятся порядковые номера, коды продуктов, цены, денежные суммы, количества, даты, имена и адреса. Использование SGML для описания транзакций электронной коммерции, приведет к чрезмерной многословности транзакции и в то же время не обеспечит важных детальных метаданных, таких, как типы данных или отдельные аспекты типов данных для каждого элемента данных.

XML-документ (например, документ, транзакция или сообщение) состоит из поименованных контейнеров и содержащихся в них данных. XML-документ также называют экземпляром (instance) или экземпляром документа XML. Обычно упомянутые выше контейнеры представлены в виде элементов и атрибутов. Контейнер-элемент может быть определен как контейнер для данных, контейнер для других элементов или контейнер, не содержащий ничего. Контейнер-атрибут может содержать только данные или не содержать ничего. В отличие от элементов, атрибут не может содержать другой атрибут. Внешне XML-документ обычно является файлом. В случае операционной системы Microsoft Windows такой файл имеет расширение имени файла .xml.

Синтаксис объявления контейнеров-атрибутов отличается от синтаксиса объявления элементов. Контейнеры-элементы требуют наличия как открывающего тега, так и закрывающего тега (также известных как тег начала и тег конца). Наличие открывающего и закрывающего тегов увеличивает многословность и, соответственно, размер XML-транзакции. Напротив, синтаксис объявления атрибутов требует единственного тега и значения, которое к нему относится (рис. 1.1).

Внутренняя структура XML-документа напоминает иерархическую структуру каталогов и файлов, отображаемую в Проводнике Microsoft Windows (рис. 1.2). Элементом верхнего уровня XML-документа является корневой элемент, существующий в документе в единственном числе. Это похоже на каталог верхнего уровня для Проводника Microsoft Windows (так называемый Рабочий стол). Как уже было сказано, элементы могут содержать значения данных, другие элементы, комбинацию того и другого или не содержать ничего. Элементы, помещенные внутрь другого элемента, мы будем называть вложенными элементами. Содержащий их элемент – родительским элементом. Вложенные элементы называются также дочерними элементами. Как известно, каталог, отображаемый в Проводнике Microsoft Windows, может содержать другие каталоги, которые, в свою очередь, могут содержать третьи и т. д. Это похоже на вложенность XML-элементов в документе. Каталоги, отображаемые в Проводнике, также могут содержать и файлы, документы и программы – аналогия с XML-элементами, содержащими значения данных. Атрибуты можно сравнить со свойствами каталога или файла. Таким образом, очевидно, что XML-структура сходна со структурами из многих других иерархических, структурных и метаинформационных технологий.

- Синтаксис элемента требует, чтобы имя элемента было заключено в угловые скобки ("<" и ">").
- Синтаксис также требует, чтобы после открывающего (начального) тега следовал закрывающий тег (тег конца).
- Альтернативный "стенографический" синтаксис допускает определение элемента посредством единственного тега, завершающегося символом "/".



- Синтаксис атрибута требует записи имени атрибута в составе пары атрибут-значение.
- Атрибут должен быть определен в элементе.

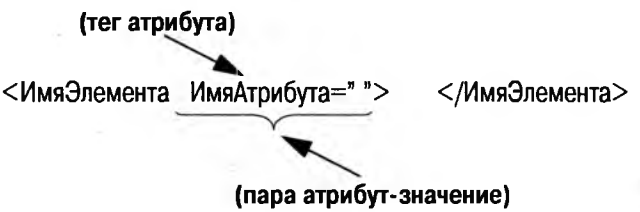


Рис. 1.1. Пример синтаксиса элемента и атрибута в XML

Чтобы обработать XML-документ, прикладная программа должна иметь возможность ориентироваться в его иерархической структуре и извлекать, при необходимости, содержимое контейнеров. Такую функциональность прикладным программам предоставляет служебная программа, известная как *синтаксический анализатор* (парсер). Упрощенно это выглядит так: прикладная программа вызывает синтаксический анализатор, который разбирает XML-документ, идентифицирует каждый контейнер и передает значения данных, содержащиеся в каждом из контейнеров, в прикладную программу. Некоторые из синтаксических анализаторов могут также сравнивать исходный документ с набором правил из некоторых метаданных и извещать прикладную программу при обнаружении противоречивости или ошибки. Набор правил и ограничений для XML-документа определяет так называемая, схема. Такой тип разбора известен как верификация. Как мы увидим в главе 4, разбор и верификация – это процессы, которые необходимы для контроля за типами данных и соответствием прочим правилам из метаданных.

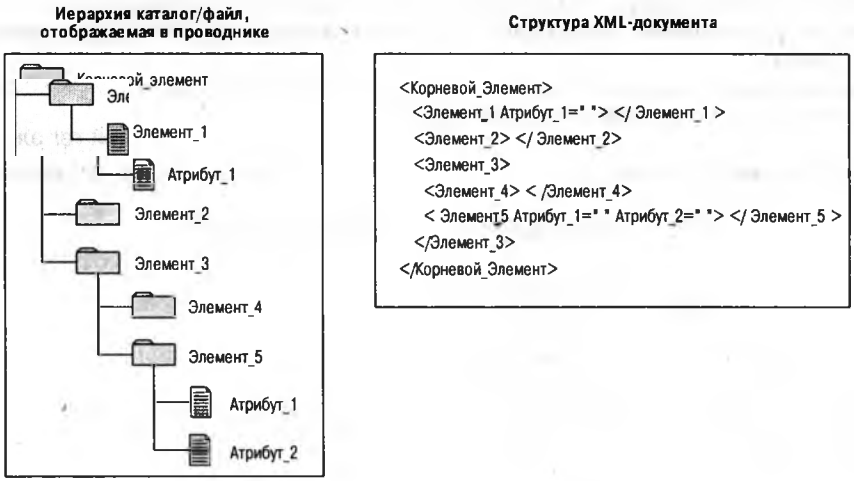


Рис. 1.2. Сходство иерархической структуры каталогов-файлов и XML

Существует два основных типа синтаксических анализаторов: синтаксический анализатор, создающий объектную модель документа (document object model – DOM), и синтаксический анализатор, предоставляющий простой API для работы с XML (simple API for XML – SAX). DOM-синтаксический анализатор строит иерархическую модель разбираемого XML-документа (известную под названием "объектная модель документа"). В ней важные места документа, разные контейнеры (элементы и атрибуты) и характеристики этой модели представлены как узлы (node). При относительно небольшом размере исходного XML-документа и достаточном количестве доступной памяти (DOM-синтаксический анализатор строит модель в оперативной памяти) DOM-синтаксический анализатор демонстрирует неплохую производительность. Однако главным его достоинством является то, что он передает прикладной программе для обработки всю структуру XML-документа полностью. Прикладная программа может перемещаться по документу так, как ей нужно.

SAX-синтаксический анализатор работает по-другому. Его работой управляют события. Здесь все также присутствует иерархическая структура документа и понятие узла, но каждый узел в отдельности возбуждает событие в прикладной программе и не строится законченная структура, представляющая весь XML-документ в целом. Прикладная программа (или связанная с ней логика) должна затем определить для каждого из событий, надо ли ей обрабатывать это событие и как это нужно сделать. SAX-синтаксические анализаторы обычно требуют меньше памяти, чем DOM-синтаксические анализаторы, и достаточно производительны при "обходе" иерархической структуры (пермещении сверху вниз, слева направо). Однако

их использование утяжеляет логику прикладных программ и усложняет навигацию по документу. Возврат к уже пройденным элементам структуры документа может потребовать сложной логики с неоднократными повторными открытиями и повторными разборами документа с начала. Обработка XML-документа с привлечением SAX-синтаксического анализатора примерно аналогична обработке реляционной базы данных с использованием процедурного языка. SQL-запрос формирует выборку из реляционной базы данных. Создается курсор, указывающий на эту выборку, и прикладная программа для обработки каждого экземпляра данных должна перемещаться по выборке и извлекать из нее записи.

Иерархическое упорядочение документа – основа XML. Однако обоснованность и целесообразность использования XML подкрепляется целым рядом важных возможностей и преимуществ. Первое из них – то, что XML – самоописывающий язык.

Самоописывающий язык

Несмотря на развитые описательные возможности и текстоориентированные возможности SGML, XML продвинулся в этом направлении гораздо дальше. XML-документ может включать описываемые контейнеры (элементы и атрибуты), чье использование подчинено строгим правилам. Для описания контейнера объявляется именованный тег. Тег позволяет обозначить каждый контейнер (элемент или атрибут) именем, причем с этим именем можно связать тип данных, которые можно размещать в обозначенных им контейнерах, а также места в документе, где допускается появление этих контейнеров. Таким образом, XML является самоописывающим языком.

Данные, содержащиеся в XML-документе, называют контентом или содержанием. При использовании описательных имен для элементов и атрибутов, содержащих данные, содержание документа становится наглядным и понятным для человека. Человек просто следует внутренней структуре XML-документа и легко определяет тип информации, содержащейся в каждом элементе или атрибуте. К примеру, если содержанием документа является адрес доставки, в нем легко различить контейнеры для адресата, контейнеры для названия города, страны и контейнер с индексом. Точно так же и прикладная программа, при помощи синтаксического анализатора, может отыскивать интересующие ее элементы и атрибуты по их тегам или именам и обрабатывать содержащиеся в них данные. Таким образом, XML-документ, содержащий элементы с описательными именами, не только может быть обработан прикладной программой, но и доступен для прочтения (или проверки) человеком (рис. 1.3).

```
<?xml version="1.0"  encoding="UTF-8"?>
<ExampleCustomers>
 <Customer CustomerID="P123456" CustomerType="Castomer-Prospect"
 Source="List 01-01-03">
  <CustomerAddresses>
   <Address AddressType="Residence-Primary" AddressNo="1">
    <AddressLines>
     <AddressLine LineNo="1">62789 N. Shadow Drive</AddressLine>
     <AddressLine LineNo="2">Building 23</AddressLine>
     <AddressLine LineNo="3">Apt. 236</AddressLine>
    </AddressLines>
    <AddressCity>Chicago</AddressCity>
    <AddressRegion>
     <StateUSAAbbrev StateName="Illinois">IL</StateUSAAbbrev>
    </AddressRegion>
    <AddressPostalCode>60699-0001</AddressPostalCode>
   <AddressCountry CountryCode="USA" CountryNo="840">USA</AddressCountry>
   </Address>
  </CustomerAddresses>
 </Customer>
</ExampleCustomers>
```

В этом примере строки с адресатами, городом, регионом и прочие части адреса обозначены (описаны) именами содержащих их элементов (тегами).

Рис. 1.3. XML – самоописывающий язык

Использование XML для описания содержимого транзакции – значительный шаг вперед по сравнению с традиционными форматами файлов для описания транзакций и интерфесов, при которых передаются только "голые данные", и для определения смысла отдельного значения используется его жестко заданное положение внутри данных (фиксированный формат) или какой-либо иной способ.

Факт. *XML – самоописывающий язык.*

Хотя файлы с жестко заданным положением данных и разделителями вполне функциональны, прикладная программа не может ориентироваться в них и извлекать необходимые значения для обработки без "карты" расположения данных или аналогичной формы разграничения данных. И это может стать для разработчика головной болью при сопровождении программы, расширении ее возможностей, отладке и устранении проблем. Исследование проблемных или ошибочных транзакций требует наличия справочника с картой структуры этих файлов. Кроме того, отсутствие описаний в составе традиционного интерфейсного файла может ограничить разработчику возможности определения контекста использования отдельного значения из совокупности данных.

В перспективе, еще большее значение имеет использование *схемы*, как метода описания содержимого документа. В зависимости от возможностей синтаксического анализатора для описания XML-документа можно использовать несколько разных типов схем. Как показано в главе 2, эти типы схем имеют свои сильные и слабые стороны. Одним из недавних достижений в этой области являются W3C XML-*схемы* (рекомендации консорциума World Wide Web Consortium [W3C], Май 2001). Во избежании двусмысленности с термином *схемы*, начиная с этого места, я буду обозначать синтаксис и возможности рекомендованной консорциумом W3C XML-схемы как "W3C XML-Схемы". Когда я буду говорить о схемах, как об общей концепции, безотносительно конкретного синтаксиса, я буду писать это слово с маленькой буквы (*схемы*). W3C XML-Схемы обеспечивают возможность описания типа данных для каждого контейнера (т. е. для элементов и атрибутов), а также позволяют задавать для них прочие правила и ограничения. Хотя для верификации можно использовать ряд других типов схем, общепризнанно, что W3C XML-Схемы наиболее надежны и более широко применимы при работе с транзакциями и контентом, ориентированным на обработку сообщений. Именно эти схемы будут в центре внимания данной книги.

W3C XML-Схемы позволяют записывать правила и ограничения для следующего

- Базовые и производные типы данных (например, числовые, строка, дата, логический и др.).
- Расширенные типы данных (например, пользовательские типы данных, созданные проектировщиком данных).
- Фасеты (например, дополнительные ограничения, такие, как длина, дробные числа и др.).
- Границы значений (например, наибольшее и наименьшее допустимое значения).
- Перечисление (например, определенный набор допустимых значений для элемента или атрибута).
- Вхождения определенного повторяющегося элемента (например, количество и модальность[1]).
- Шаблоны (например, формат и шаблоны редактирования)

Как уже говорилось, синтаксический анализатор – программная утилита, помогающая в навигации по XML-документу и извлечении требуемых данных. В расширение этой концепции верифицирующий синтаксический анализатор сравнивает правила и ограничения, налагаемые схемой, с содержанием XML-

[1] Под модальностью подразумевается допустимость вхождения некоторого элемента. – *Примеч. ред.*

документа и информирует прикладную программу об ошибках. Чтобы иметь возможность применить к документу ограничения (взятые из метаданных), поддерживаемые W3C XML-Схемами, необходимо использовать верифицирующий синтаксический анализатор, который поддерживает этот тип схемы.

Другой важной особенностью XML является возможность задавать имена контейнеров (элементов и атрибутов) в соответствии с принятыми на предприятии стандартами именования, а не на основе жесткого синтаксиса. В отличие от HTML с его предопределенными элементами, атрибуты и элементы, определяемые в XML-документе, задаются, по большей части, разработчиком или исполнителем. В XML-документе существует сравнительно немного предопределенных или зарезервированных тегов (рис. 1.4). Вследствие того, что XML-теги не предопределены, а определяются потребностями предприятия, XML-документ расширяем. Это означает, что при необходимости XML-документ и соответствующая схема могут быть расширены за счет включения новых контейнеров (элементов и атрибутов).

Отчасти прямое сопоставление HTML и XML не вполне корректно. HTML – прекрасная технология для презентации или визуализации информации в браузере. То есть элементы и атрибуты HTML изначально предназначены для визуального форматирования документоориентированного содержания. К примеру, HTML-элемент "<p>" определяет параграф. HTML-элемент "<font>" в комбинации с атрибутами предписывает браузеру отобразить содержимое этого элемента шрифтом указанного размера и стиля.

В дополнение к общей визуализации или отображению содержимого HTML-формы предоставляют возможность получать от браузера вводимую пользователем информацию и передавать собранные данные на сервер. Элементы и атрибуты, используемые в HTML-формах, обеспечивают минимальную поддержку описательных метаданных. Возьмем, к примеру, элемент "<input>", приводящий к появлению в окне браузера поля для ввода данных. Метаданные для верификации содержат атрибуты size и maxlength. Атрибут size задает ширину, которую будет иметь поле ввода при отображении его в браузере, а атрибут maxlength определяет наибольшее количество символов, которое можно ввести в это поле.

В противоположность HTLM, XML ориентирован, главным образом, на описание и хранение данных. Но несмотря на то что наибольший эффект приносит использование XML именно для описания данных, существуют технологии (например, eXtensive Styleheet Language Transform [XSLT]), которые можно использовать для форматирования или преобразования XML-содержания для презентации этой информации пользователю. Например, если применить к XML-документу XML-таблицу стилей, содержащую характеристики шрифта и заголовки, он может быть визуализирован браузером точно так же, как HTML-документ.

HTML-документ

```
<html>
   <head>
      <title>Product Info</title>
   </head>
   <body>
      <p>
        <font face=" Arial" size="2">
          <b> Product</b>
        </font>
       </p>
      <p> Product ID:12345</p>
      <p>Product Description:Widget </p>
      <p>Product Dimensions:</p>
      <p> Weight:2.5 LB</p>
      <p>Length:14.3 IN</p>
      <p>Height:6.1 IN</p>
      <p>Width:21.7 IN</p>
   </body>
</html>
```

<html>, <head>, <body>, <p> и другие теги имеют строго определенную смысловую нагрузку и все они предопределены синтаксисом HTML

XML-документ (экземпляр)

```
<?xml version="1.0" encoding="UTF-8"?>
<Product>
 <ProductID> 12345</ProductID>
 <ProductDescription> Widget</ProductDescription>
 <ProductDimensions>
  <Weight Unit= "LB">2.5</Weight>
  <Length Unit=" IN">14.3</Length>
  <Height Unit=" IN">6.1</Height>
  <Width Unit=" IN">21.7</Width>
 </ProductDimensions>
</Product>\
```

</Product>, <ProductID>, <ProductDescription> и другие теги определяются разработчиком или исполнителем

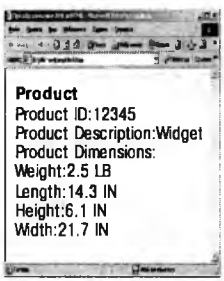


Рис. 1.4. Отличие предопределенных тегов HTML от XML-тегов

Одно из самых очевидных отличий XML от HTML – в имени, применяемом для обозначения элемента или атрибута. Как уже говорилось, большая часть HTML элементов и атрибутов является предопределенной, они выполняют заданные функции по презентации и визуализации. В XML значения имен элементов и атрибутов специфичны для каждого конкретного документа. За редким исключением (например, зарезервированные XML-атрибуты для обозначения языка, версии и кодировки символов) за этими именами не закреплена определенная смысловая нагрузка. Это является преимуществом, когда разработчик определяет, каким образом описать содержимое некоторого документа. XML-атрибуты и элементы выступают в роли именованных контейнеров для данных, а не фокусируются на виде шрифта и других презентационных характеристиках данных. Когда XML-документ используется совместно с относящейся к нему схемой, то добавляется поддержка дополнительных характеристик метаданных.

Наиболее полно возможности XML раскрываются при работе с данными. Например, в транзакциях глобальной электронной коммерции или в обмене корпоративными данными. Однако возможности XML значительно шире, чем просто использование в транзакциях. Если рассматривать ситуацию в масштабах целого предприятия, а не отдельного автономного приложения, то XML покажет себя как совместимый с различными платформами, повторноиспользуемый, гибкий и расширяемый инструмент.

Со свойствами самоописываемости XML связана кодировка символов. Большинство XML-документов кодируется с помощью какой-либо формы Unicode, обычно это "UTF-8", которая включает как подмножество набор символов ASCII. Это позволяет просмотреть содержание XML-документа в самых простых текстовых редакторах. И если проектировщик использовал осмысленные имена для элементов и атрибутов документа, существует потенциальная возможность "засветить" конфиденциальные данные. Контейнер-элемент с именем <CreditCardNumber> может заинтересовать случайного читателя. В таких случаях необходима какая-либо форма шифрования или аналогичная мера безопасности.

Межплатформенное взаимодействие

XML – универсальная технология. Если отправляющая и принимающая платформы могут читать и создавать текстовые файлы в кодировке Unicode (обычно используется UTF-8 и ASCII) и для этих платформ существует синтаксический анализатор XML, то XML может быть использован для обмена данными между ними и их обработки. Различия в операционных системах и прочие вещи, такие, как преобразования наборов символов, не имеют значения. Хотя большинство XML-документов имеют кодировку ASCII, потенциально поддерживается более широкий набор символов. Эти расширенные возможности обеспечиваются поддержкой Unicode, и при необходимости содержание документа можно создавать не только на английском, но и на других языках.

Принимая во внимание, что XML фактически не зависит от платформы, он также является и агностиком платформы. Он предоставляет возможность описывать данные, передаваемые между автономными техническими средами, а также совершенно разнородными приложениями предприятия. XML-транзакция, созданная и прошедшая первоначальную обработку на платформе Windows в базе данных SQL Server 2000, может быть также обработана на платформе Unix в базе данных ORACLE (естественно, каждая из платформ и сред должна поддерживать Unicode). Возможность межплатформенной обработки обеспечивает поддержку *интеграции предприятия (enterprise integration)* (также называемой *интеграцией корпоративных приложений – enterprise application integration – EAT*).

Факт. *XML независим от платформы (по умолчанию используется кодировка Unicode и UTF-8, поддерживающая основной набор символов ASCII).*

Общей формой интеграции предприятия является распределение, обмен или перемещение данных между автономными системами предприятия, которые были разработаны давно и эксплуатируются в течении долгого времени, на разных технических платформах, с различной архитектурой баз данных. Например, информацией об изделиях, находящейся в ведении системы закупок, можно делиться и обмениваться с системой инвентаризации, а также с системой оформления заказов. Каждая из этих систем может работать на собственном оборудовании, со своей операционной системой и СУБД. Но если все эти платформы поддерживают работу с кодировкой Unicode и имеют свой XML-синтаксический анализатор, то для описания циркулирующих между ними данных можно эффективно использовать XML (рис. 1.5).

По свидетельствам ведущих журналов по информационным технологиям, XML быстро становится важнейшей технологией для межплатформной и совместной обработки. Развитая поддержка XML обеспечивается большинством ведущих поставщиков технологий. Ко времени этой публикации поддержку XML в свои продукты и платформы включили: IBM, Microsoft, Oracle, Sybase, TIBCO, Altova и многие другие поставщики технологий. Хотя Xml можно поддерживать по-разному, наиболее распространенная форма поддержки включает XML-синтаксический анализатор, XML SDK (набор для поддержки разработки ПО с использованием XML) и расширения СУБД средствами импорта, экспорта и хранения данных в формате XML.

Другим применением XML, к которому растет всеобщее внимание, является определение переносимых моделей для обмена между инструментами проектирования и разработки. Многие из этих инструментов обладают возможностями импорта-экспорта XML-файлов. В ближайшем будущем модели, разработанные в одном инструменте моделирования и экспортированные в файловый формат с XML-описаниями, могут быть импортированы в другой аналогичный инструмент. Группа OMG[2] (Object Management Group) представила этот сценарий, предлагая использовать общий инструментальный набор словарей, базирующихся на XML, для описания моделей и моделируемых объектов в соответствии со стандартной структурой XML.

Повторное использование

Повторное использование – одна из прекрасных возможностей уменьшить стоимость разработки приложения, технические затраты, эксплуатационные издержки, повторное использование помогает создавать и продвигать стандарты данных предприятия. Проще говоря, повторное использование – это возможность

[2] Object Management Group. OMG Modeling and Metadata Specifications. XML Metadata Interchange (XMI). Находится по адресу http://www.omg.org. – *Примеч. авт.*

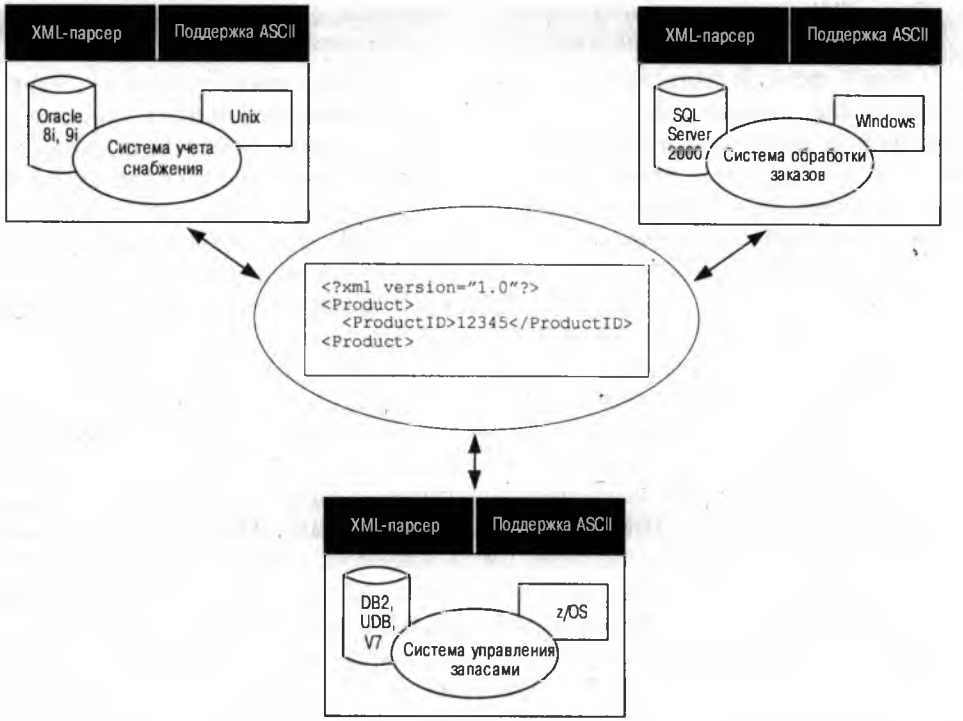


Рис. 1.5. Использование XML для описания данных, циркулирующих между разнородными системами

один раз что-то разработать и затем пользоваться этим снова и снова. Еще важнее расширенное, или широкомасштабное (broad scale), повторное использование, когда созданное однажды используется повторно, причем даже и для других целей или вне рамок того контекста, чем задумывалось при создании.

Более формальное определение повторного использования включает два основных вида деятельности. Первый – разработка с расчетом на повторное использование (т. е. разработка чего-либо с учетом того, что это будет использоваться неоднократно). Второй – признание возможности повторного использования (т. е. определение того, что что-либо может быть использовано повторно, проверка того, что повторное использование эффективно, а затем – собственно использование). Как ни удивительно, про повторное использование часто забывают или игнорируют его, принося в жертву методам быстрой разработки. Часто причиной уклонения от формализованного повторного использования являются трудности в реализации приемов практического повторного использования. Но с приходом W3C XML-Схем повторное использование может быть быстро реализовано и усилено многими способами.

Факт. *XML является повторноиспользуемым (W3C XML-Схемы могут быть спроектированы как модульные компоненты).*

В документе "May 2001 Recommendation for XML Schemas"[3] консорциума W3C приведены технологии, позволяющие структурировать метаданные для поддержки нескольких видов повторного использования. Можно определить контейнеры-элементы, наборы из контейнеров-элементов и собственные, нестандартные типы данных в отдельной W3C XML-Схеме и затем неоднократно их использовать с помощью ссылки. Так, можно один раз определить собственный тип данных (например, использовать глобально объявленный тип "simpleType", отражающий стандарт предприятия по представлению денежных сумм) и использовать этот тип данных во всех документах, ссылаясь на схему, где он объявлен (рис. 1.6).

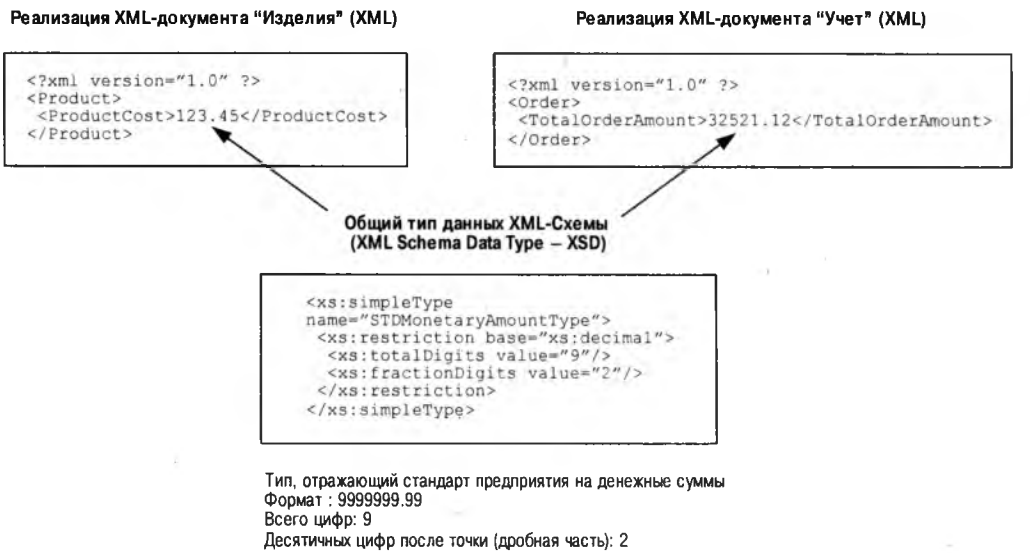


Рис. 1.6. Пример повторноиспользуемого типа данных для денежных сумм

Еще большее значение имеют модульные внешние компоненты W3C XML-Схем, содержащие элементы и нестандартные типы данных, которые можно использовать повторно во многих других W3C XML-Схемах посредством их включения или с помощью ссылки на них. Обратимся к примеру с почтовым

[3] World Wide Web Consortium (W3C). XML Schemas. W3C Recomendation, May 2, 2001. Находится по адресу http://www.w3.org/TR/2001/REC-xmlschemas-1-20010502/ (структуры); http://www.w3.org/TR/2001/REC-xmlschemas-2-20010502/ (типы данных). – *Примеч. авт.*

адресом. Посредством W3C XML-Схем можно определить метаданные со стандартом предприятия на почтовый адрес, которые бы допускали гибкость формата почтового адреса, чтобы учитывать широкое разнообразие международных форматов почтовых адресов. Такую схему (назовем ее Address) можно затем сделать стандартом и использовать неоднократно в самых разных контекстах (рис. 1.7).

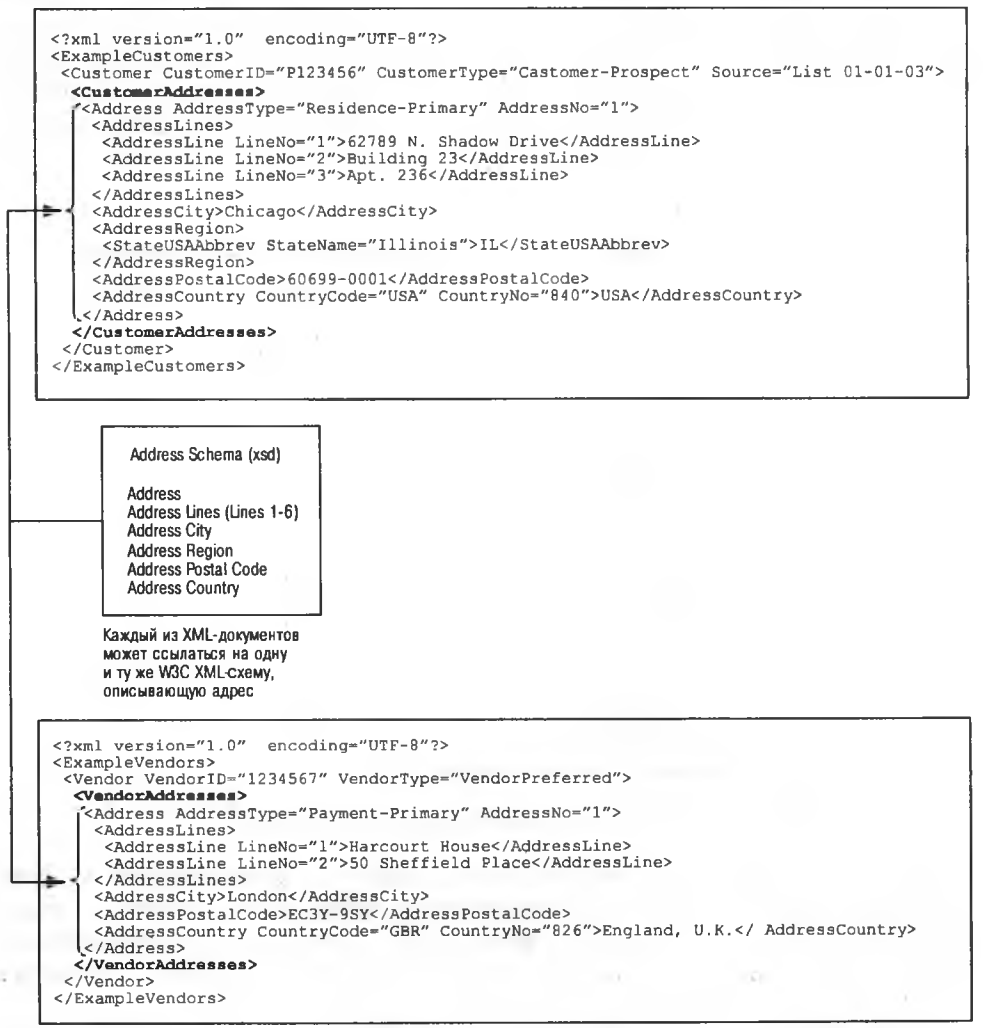


Рис. 1.7. Повторное использование схемы Address в разных контекстах

Схема-компонент с описанием формата адреса может быть сконструирована так, чтобы варианты ее использования были очевидны. Приложения, в которых не нужно иметь четыре или пять адресных строк, как это требуется во многих международных адресах, могут игнорировать эти дополнительные элементы. Контейнеры для этих адресных строк определяются как "необязательные". W3C XML-Схема для почтового адреса, отражающая стандарт предприятия, может повторно использоваться другими W3C XML-Схемами для представления адреса покупателя, адреса поставщика или адреса служащего. Если в будущем потребуется изменить структуру почтового адреса, то достаточно будет изменить только соответствующую схему-компонент с форматом адреса и не потребуется изменять все ссылающиеся на нее схемы (рис. 1.8).

Рис. 1.8. Схема со стандартом адреса, на которую ссылаются другие схемы

После определения и публикации схем, отражающих стандарты предприятия, затраты на начальную разработку, а также эксплуатационные затраты (относящиеся к XML) могут быть уменьшены или сведены к нулю. XML-схемы, которые могут потребоваться для вновь разрабатываемых приложений, можно создавать с помощью ссылок на свои наработки схем-компонентов (подобно сборочному производству из готовых частей). Таким образом, исключаются новые разработки "с нуля" или повторное воспроизведение уже сделанного силами других разработчиков.

Гибкость

Архитектура баз данных и структура интерфейсных файлов (файлов для передачи данных от одного приложения к другому) часто задаются разработчиками жестко. Вернемся к примеру с почтовыми адресами. Интерфейсные файлы, содержащие североамериканский почтовый адрес, часто определятся со специ-

фическими полями: для двух адресных строк с названием улицы, для города, штата и почтового индекса. Однако коль скоро мы продвигаемся к глобальной электронной коммерции, эти традиционные адресные структуры невозможно использовать для поддержки или описания международных форматов адресов. Чтобы в статичный (с жесткой структурой) интерфейсный файл внести изменения, учитывающие сильно различающиеся адресные данные разных стран, используют один из следующих традиционных подходов

- Пытаются запихнуть глобальные адресные данные в существующие интерфейсные файлы (верный путь к ошибкам).
- Перерабатывают и расширяют существующие интерфейсные файлы для поддержки интернациональных адресов.
- Создают новые интерфейсные файлы.
- Игнорируют глобальные адресные данные (или некоторую их часть).

Кроме интерфейсных файлов, используемых для транзакций и обмена данными, может потребоваться внести изменения в архитектуру исходной и целевой баз данных. К стоимости переработки существующих таблиц баз данных в соответствии с изменениями структуры данных могут прибавиться затраты на корректировку функций доступа к данным и логики приложения. И иногда незначительные изменения структуры данных, типа добавления двух-трех дополнительных адресных строк для улицы, могут вызвать заметные накладные расходы.

Факт. *XML – гибкий язык (XML-структуры можно разрабатывать в расчете на оперативное расширение или сокращение).*

XML предлагает большую гибкость решения проблем транзакций и обмена данными. XML-структуры, разработанные в соответствии с вертикальными моделями и архитектурными формами, являются достаточно гибкими. Как будет показано в главе 6, вертикальная модель использует группы подобных элементов. При использовании гибких архитектурных форм добавление в XML-документ новых контейнеров-элементов (например, дополнительных адресных строк) может быть выполнено с минимальными издержками или вообще без них. Определение элементов для адресных строк в качестве необязательных добавляет еще одну степень гибкости. В XML-документе могут присутствовать только элементы, необходимые для адреса любого типа. При правильном подходе к их разработке XML-структуры могут быть расширены или ужаты в соответствии с контекстом данных, в которых они используются (рис. 1.9).

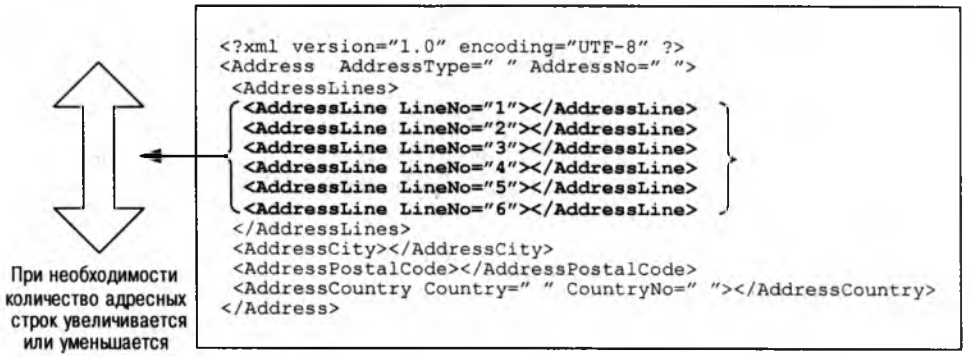


Рис. 1.9. Гибкая архитектурная форма (вертикальная)

Расширяемость

Часто приходится слышать, что единственной постоянной вещью в мире является его изменчивость. Бизнес-парадигмы, технологические платформы и подходы к разработке подвержены частым изменениям. Рассмотрим пример предприятия, занимающегося прямой доставкой своих товаров потребителю, но получающего платежи при посредничестве третьей стороны. Простая транзакция обработки заказа включает основную информацию о покупателе, заказ, а также некоторую учетную информацию по заказу. Однако информация о платежах будет отсылаться третьей стороне, со сведениями о факте оплаты, находящимися вне рамок видимости отдельной транзакции по заказу.

Если позднее будет принято решение напрямую использовать новый тип платежных средств (например, использовать расчет с помощью кредитных карт), в транзакцию по заказу потребуется внести несколько новых полей: для номера кредитной карты, срока действия и данных авторизации. Файлы транзакции необходимо будет расширить для введения новых полей. Если для определения транзакции был использован XML, расширения можно осуществить добавлением или вставкой в базовую XML-транзакцию и схему новых элементов и атрибутов или ссылкой на эти дополнительные контейнеры в составе отдельной схемы-компонента (рис. 1.10). Еще одно преимущество этой техники в том, что дополнительные элементы и атрибуты не обязано использовать каждое приложение, принимающее участие в обработке транзакции.

Факт. *XML расширяем (XML-структуры и схемы могут быть расширены сами или добавлены для расширения другого).*

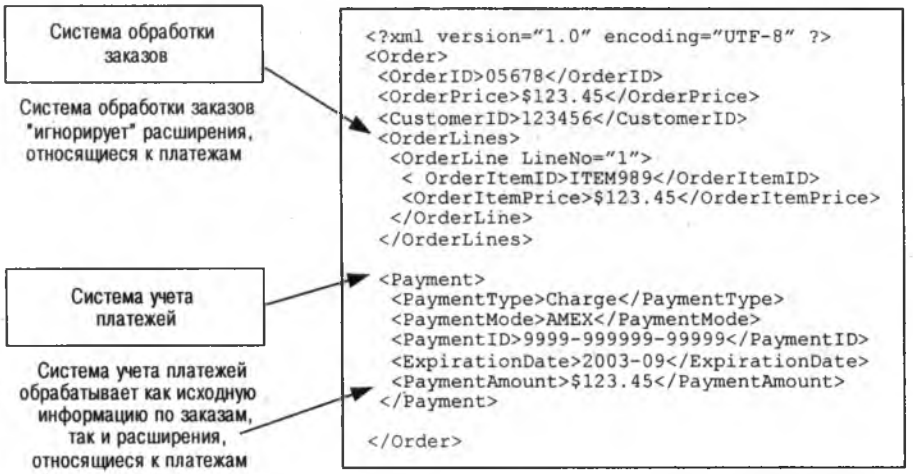


Рис. 1.10. Обработка расширенного XML-документа

В качестве другого примера рассмотрим предприятие, внедрившее у себя XML и использующее существующую отраслевую схему, описывающую транзакции по заказам. Такие предопределенные схемы мы часто будем называть словарями (например, XML-словарь по заказам). Если некоторая схема содержит исчерпывающий набор элементов данных для поддержки определенной функции, то такая схема является словарем. В рассматриваемом случае предприятие хочет добавить для каждой строки с данными о заказе свои альтернативные пакетные модули.

Если исходная схема была разработана как W3C XML-схема, ее можно расширить, добавив дополнительные элементы с уникальным квалификатором (например, используя "пространство имен"). Такие модификации в минимальной степени затронут исходную XML-схему и ее дальнейшее использование. Пространство имен обеспечивает возможность уникально идентифицировать схему и определяемые ею контейнеры и избежать коллизий с другими контейнерами, имеющими то же имя. Понимание пространств имен может оказаться немного затруднительным, однако эта концепция дает ощутимую пользу. Пространство имен обычно описывается идентификатором ресурса (например, универсальным идентификатором ресурса – URI). Назначение URI пространства имен – действовать как уникальный пул или набор входящих в него вещей, например элементов данных. Пространству имен можно присвоить префикс – разновидность аббревиатуры. Когда этот префикс используется в XML-элементах или атрибутах, это означает, что данные контейнеры входят в соответствующее адресное пространство и там же и определяются.

Рассмотрим XML-элемент с объявленным тегом <title>. Если этот элемент встречается среди данных, предназначенных для публикации, первой мыслью может быть, что этот элемент предназначен для хранения заглавия книги, журнала, статьи и т. п. Однако если в состав данных о публикации входят контейнеры-элементы с информацией об авторе, элемент с тегом <title> можно по ошибке заполнить титулом автора (например, кандидат технических наук, профессор и т. п.). Креативный проектировщик данных должен разрешить эту ситуацию добавлением к данным о книге дополнительного контейнера-элемента <title>. Синтаксически допустимо иметь два элемента с одинаковыми тегами, однако прикладной программе будет трудно отличить, какой из них содержит название книги, а какой – титул автора.

Пространства имен, поддерживаемые в XML и W3C XML-Схемах, позволяют иметь два элемента-контейнера <title>, каждый из которых связан с собственным пространством имен и уникально квалифицируется этим пространством. С каждым пространством имен может быть связан свой префикс, и этот префикс затем может присваиваться элементам. Таким образом, первый элемент <title> может быть частью пространства имен "publication" (например, " <publication:title>"), а второй – частью пространства имен "author" (например, " <author:title>"). Эта техника позволяет избегать коллизий имен объектов (когда два контейнера-элемента, имеющие одно имя, содержат данные из разных контекстов). Предназначенный для широкого распространения словарь, определенный с использованием W3C XML-Схем, может принадлежать к некоторому пространству имен. Если предприятию требуются дополнительные элементы, их можно связать с другим пространством имен и затем добавить к схеме. Таким образом, можно избежать коллизий между одинаково названными элементами и сохранить целостность исходного отраслевого словаря.

Как описывалось выше, существует много преимуществ в использовании XML и XML-Схем. Однако эффективное использование XML в качестве технологии метаданных не полностью безоблачно. Существуют также сопутствующие задачи и несколько проблем. Первая проблема – согласовать требования и характеристики данных с типом документа и определить, в каких случаях использование XML в проекте может дать выигрыш.

Глава 2

Типы XML-документов

Несмотря на все свои возможности и достоинства, XML не всегда является наилучшей технологией метаданных для проекта. Принятие решения об использовании XML требует тщательной оценки данных проекта: функциональных требований и того, какие характеристики данных должны быть описаны. Если в итоге признано, что XML – наилучший выбор, то одной из первых задач проектирования данных становится следующая задача: определить, нужно ли применять схему, поддерживающую ограничения (схема с набором определений, правил и ограничений, применяемых синтаксическим анализатором при разборе и верификации). Существует несколько типов XML-схем, из которых можно выбрать одну или несколько. Каждый тип имеет свои сильные и слабые стороны. К числу трех наиболее часто используемых типов схем относятся следующие типы:

1. определения типов документа (document type definition – DTD или dtd);
2. ограничения XML-данных (XML data reduced – XDR или xdr);
3. W3C XML-Схемы (W3C XML Schemas – XSD или xsd).

Важно отметить, что для некоторых приложений XML-документ (например, XML-файл, документ, транзакция или сообщение) может обрабатываться и без применения схем. Когда XML-документ ясно описан, имеет простую структуру и не требует применения бизнес правил и ограничений, а также невелик по размерам, то верификация его синтаксическим анализатором на соответствие какой-либо схеме не требуется (рис. 2.1). Однако для XML-документов, не ссылающихся на схему, нельзя провести проверку того, что содержащиеся в контейнерах данные имеют допустимый тип данных или что порядок расположения элементов данных внутри документа соответствует ожидаемому.

Схема обеспечивает поддержку дополнительных характеристик, относящихся к метаданным, таких, как взаимное расположение, количественные характеристики, допустимые значения и в некоторых случаях – типы данных. Каждый тип схемы действует как метод описания характеристик данных и применения правил

```
<?xml version="1.0" encoding="UTF-8"?>
<ExampleHRTransaction>
  <EmployeeIdentifier>12345</EmployeeIdentifier >
  <EmployeeName>Sally M. Johnson</EmployeeName >
  <EmployeeHireData>2002-05-27</EmployeeHireData >
</ExampleHRTransaction>
```

Рис. 2.1. Простая реализация XML-документа

и ограничений к соответствующему XML-документу. Хотя для достижения требуемого результата можно использовать несколько типов XML-схем, одним из определяющих факторов выбора наиболее подходящего типа является ориентация и организация данных в рассматриваемой реализации XML-документа. Внутренняя структура XML-документа может быть организована по-разному, для представления разных видов содержимого, в том числе документо- или текстоориентированного, ориентированного на транзакции и ориентированного на сообщения. У каждого типа XML-схем есть свои преимущества и ограничения.

Применение DTD-типа схемы ограничено степенью детализации[1] информации и поддержкой типизации данных. Кроме того, синтаксис DTD-схем порожден от синтаксиса SGML, отсюда – его использование не столь интуитивно понятно, как использование других типов схем (рис. 2.2). Использование DTD для верификации типичной транзакции электронной коммерции позволит лишь ввести поименованные контейнеры и задать их взаимное расположение. Однако с помощью DTD нельзя описать, должен ли данных контейнер (элемент или атрибут) содержать дату, количество или какой-либо другой, отличный от символов тип данных. Естественно, применение DTD-схем для детализации транзакции электронной коммерции не очень эффективное решение.

```
<?xml version="1.0" encoding="UTF-8"?>
<! ELEMENT ExampleHRTransaction (EmployeeIdentifier, EmployeeName,
 EmployeeHireData)>
<! ELEMENT EmployeeIdentifier (#PCDATA)>
<! ELEMENT EmployeeName (#PCDATA)>
<! ELEMENT EmployeeHireData (#PCDATA)>
```

Рис. 2.2. Пример определения типа документа

[1] Здесь и далее под степенью детализации подразумевается прежде всего гранулярность, т. е. использование дискретно определяемых элементов данных (высокая гранулярность) или же строк текста (низкая гранулярность). – *Примеч. ред.*

XDR-типы схем были более ранней попыткой обеспечить необходимый уровень детализации данных и возможность описания типов данных для содержимого XML-документа, что было недоступно при использовании DTD. Как отмечалось ранее, XML – прекрасная технология для описания содержимого транзакций электронной коммерции или обмена данных. Однако использование XML для этих целей совместно с DTD не позволяет описывать такие характеристики, как типы данных. Чтобы повысить описательные возможности XML и преодолеть ограничения, присущие DTD, создали XDR. Первоначально XDR-типы схем поддерживались компанией Microsoft и рядом других технологических компаний. Однако затем поддержка XDR-схем ослабла. Ряд участников консорциума W3C предложили более строгую альтернативу, ставшую известной как "XML Schemas Recommendation" (рекомендации по XML-схемам). W3C XML-Схемы перекрывают возможности XDR-схем, и они были поддержаны еще большим количеством поставщиков технологий. В результате, применение XDR-схем сошло на нет, при росте использования W3C XML-Схем. Наше обсуждение DTD и XDR завершается констатацией того, что большинство поставщиков технологий ввели поддержку W3C XML-Схем во многие свои продукты.

W3C XML-Схемы обеспечивают расширенные возможности метаданных и именно этот тип схем рекомендуется к применению в большинстве приложений. W3C XML-Схемы обеспечивают сильно детализированный метод описания содержимого XML-документа. Как и DTD, W3C XML-Схема может описательно именовать или идентифицировать содержимое транзакции. W3C XML-Схемы также предоставляют расширенные возможности в следующих областях: типы данных, настройка и повторное использование. С их помощью на использование каждого контейнера (элемента или атрибута) XML-документа можно наложить ограничения: тип данных, который контейнер может содержать, а также некоторые дополнительные характеристики, например наибольшая и наименьшая длина данных, формат представления десятичных цифр и другие ограничительные фасеты. W3C XML-Схемы также обеспечивают возможность представления и включения повторноиспользуемых структур, контейнеров и нестандартных типов данных. Дополнительным плюсом является то, что в W3C XML-Схемах используется синтаксис XML (они самоописывающие и обычно менее запутанные, чем DTD). Синтаксически W3C XML-Схема представляет собой набор элементов и атрибутов, как и XML-документ. Местоположение, взаимосвязь и значения элементов и атрибутов схемы представляют серии правил, которые описывает эта схема.

Как и XML-документ, W3C XML-Схема содержит один "корневой" элемент (известный как элемент "схема"). Если схема определяется в некотором пространстве имен, она также может содержать ссылку на пространство имен. Хотя обычное назначение пространства имен – обеспечить уникальность и избежать коллизий среди именованных объектов, с помощью ссылки на базовое простран-

ство имен (или пространство имен, принятое по умолчанию) W3C XML-Схемы, можно проинформировать синтаксический анализатор о версии схемы. Подобно другим технологиям, W3C XML-Схемы непрерывно совершенствуются, и версия схемы отражается, в частности, в пространстве имен. Пространству имен очередной версии схемы назначается свой префикс, и все относящиеся к этой схеме элементы и атрибуты должны включать этот префикс. Самым общим префиксом, относящимся к W3C XML-Схемам, является префикс "xs:" или "xsd:".

За корневым элементом схемы следуют другие элементы, атрибуты и конструкции, с различными именами и типами. Каждый из них служит для описания некоторого контейнера данных или характеристики, ссылающейся на схему реализации XML-документа. Таким образом можно определять как отдельные элементы, так и объединять элементы в группы, используя ключевое слово "complexType" или "group". В отличие от элементов, атрибуты не могут определяться отдельно, они обязательно должны быть определены в составе какого-либо элемента. И элементы, и атрибуты могут иметь как применяемый к ним тип данных, так и другие ограничения и характеристики. Как будет показано в главах 4 и 5, возможность задать тип данных и затем определить дополнительные ограничения на этот тип является не только мощной возможностью метаданных, но и согласуется с традиционной практикой проектировщиков данных (рис. 2.3).

Вместо того чтобы выбрать единственный тип схемы и применять его для всех и каждой из форм данных, которые могут содержаться в XML-документах, более эффективно делать следующее: в каждом конкретном случае анализировать ориентацию, организацию и способ использования данных и затем использовать наиболее подходящий в этом случае тип схемы. Большинство прикладных программ (особенно те, что связаны с глобальной электронной коммерцией и интеграцией предприятия) реализуют логику по презентации, экспорту, импорту, чтению, записи, изменению, совместному использованию или обмену данных. Такие данные обычно относятся к одному из трех базовых типов:

1. документоориентированные;
2. ориентированные на транзакцию;
3. ориентированные на сообщение.

Теоретически, можно применять любую схему данных. Однако первичными факторами являются степень детализации, структура и назначение данных. Выбор типа схемы, подходящей под характеристики данных и требования проекта, имеет большое значение. Пробелы или недостаточная функциональность выбранного типа схемы могут дорого обойтись.

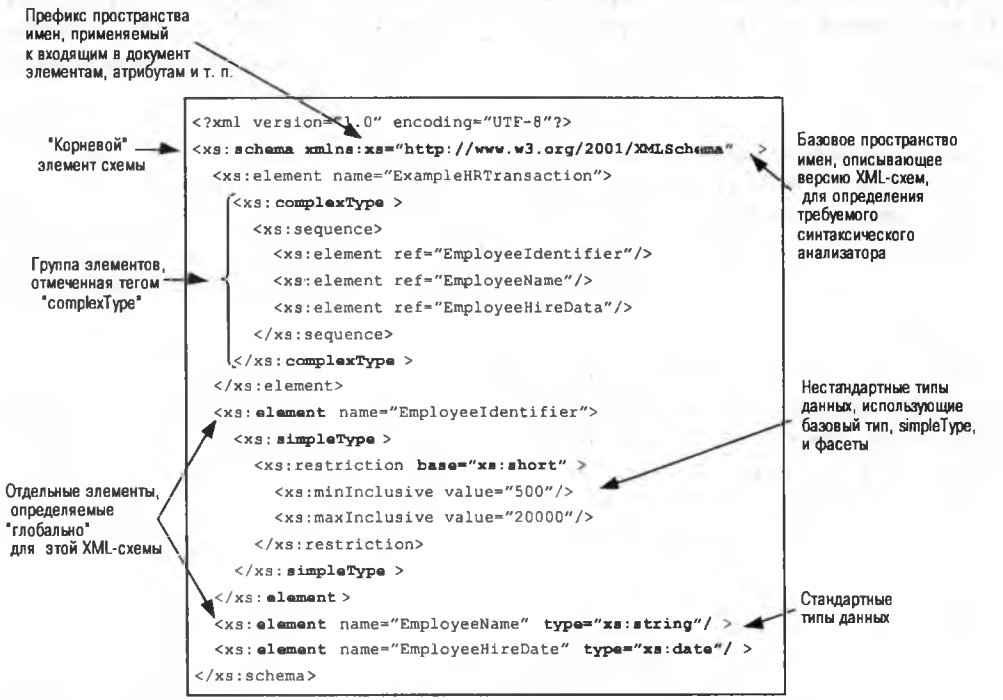


Рис. 2.3. Пример XML-Схемы

Документоориентированное содержание

Исторически документоориентированное содержание является наиболее распространенным и узнаваемым типом данных, доступных в Web. Упрощенно, документоориентированное содержание – это набор текстовых строк, часто визуализируемых с помощью браузера в качестве информации, в качестве ссылок или в качестве развлечения (например, для просмотра и иногда прослушивания человеком).

Документоориентированное содержание, например, имеют:

- книги, журналы, периодика и статьи;
- бюллетени, списки и повестки дня;
- корреспонденция и документация;
- брошюры, рекламные и маркетинговые материалы;
- академические, учебные и другие справочники;
- каталоги продукции и описания изделий.

Документоориентированное содержание структурировано в соответствии с фразами, предложениями, параграфами, заголовками, оглавлениями и потоками информации. Дополнительно в содержание могут включаться характеристики форматирования, например стиль и размер шрифта. Хотя документоориентированное содержание, предназначенное для визуального потребления, может быть эффективно выражено с помощью XML, чаще его записывают в виде HTML-документов.

При использовании XML для описания документоориентированного содержания обычно формируют структуру XML, напоминающую структуру книги с оглавлением и создают ссылку на таблицу стилей, предназначенную для презентации этих данных и задания характеристик формата. Таблица стилей – это еще одна разновидность XML-технологии, она указывает пути (paths) и места в XML-документе и представляет правила преобразования XML-документа в другую форму, например в другой XML-документ или в HTML-документ (рис. 2.4). Преимущество такого подхода в том, что значения данных, содержащихся в XML-документе, отделены от итоговой формы их презентации. При этом, наряду с сохранением структурной целостности исходного XML-документа, существует возможность изменения или расширения данных с минимальным влиянием на таблицу стилей. Несмотря на изменение содержимого, формат и внешний вид документа в браузере останется прежним.

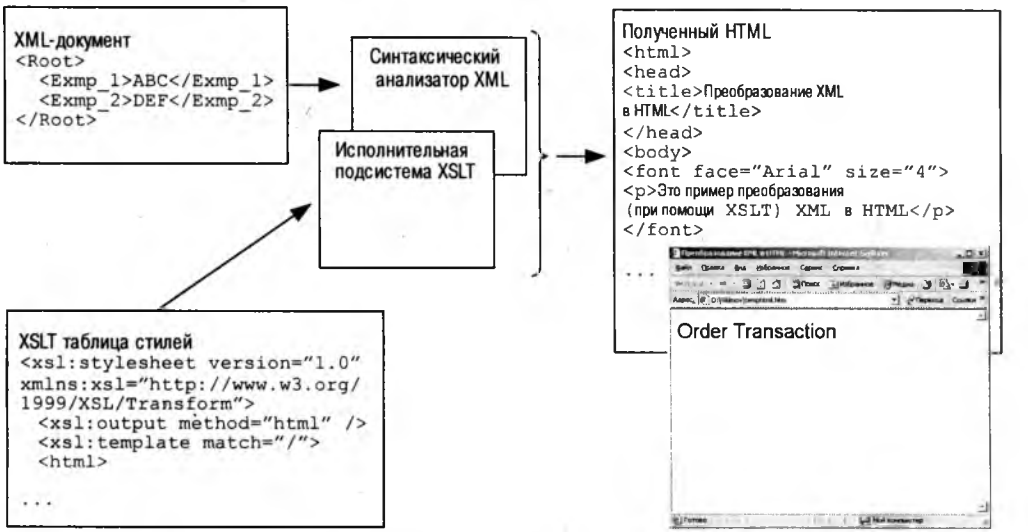


Рис. 2.4. Применение таблицы стилей к XML-документу для его визуализации

Эту технику используют и на интернациональных Web-сайтах для информации, отображаемой на разных языках. Переводы на разные языки части информации, требующей перевода, можно хранить в отдельных XML-документах, а визуализировать ее с помощью одной таблицы стилей XML. Получается согласованный внешний вид, независимо от языка. Альтернативный подход – описать содержимое данных, используя один базовый язык, а затем применять разные таблицы стилей, в зависимости от языка. Заглавия и заголовки описываются на разных языках в каждой из таблиц стилей и применяются к содержимому XML-документа (рис. 2.5). Использование этих подходов дает положительный результат, однако это не единственно возможное применение XML в глобальных или интернациональных целях.

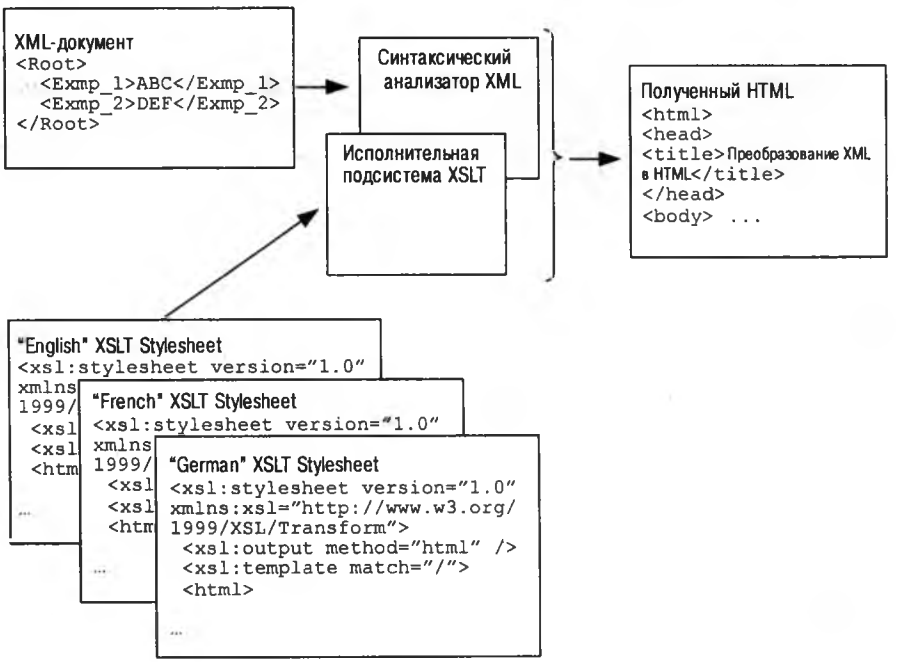


Рис. 2.5. Ориентированные на разные языки таблицы стилей XML применяются к одному XML-документу

Документоориентированное содержание можно описать с помощью большинства типов XML-схем. Однако иногда для документоориентированного содержания требуется задать характеристики отдельных частей в составе метаданных, например тип данных или допустимые значения. В результате, предпочтительнее использовать W3C XML-Схемы. Если же документоориентированное содержание – просто строки

текста и их основное назначение – презентация или визуализация браузером, приемлемым выбором будут DTD-схемы. W3C XML-схемы также иногда используются для поддержки сложной организации документа, навигации и для описания таких структур документа, которые динамически расширяются или уменьшаются (т. е. гибких).

Рекомендация. *DTD-схемы могут эффективно использоваться как метод описания и введения ограничений для простого документоориентированного содержания. Однако во многих случаях вместо них с тем же успехом можно использовать и W3C XML-Схемы.*

Содержание, ориентированное на транзакции

XML очень хорошо подходит как способ описания транзакций. Содержимое, ориентированное на транзакции, может служить многим целям, но чаще всего это данные, которыми обмениваются приложения, или данные, полученные от Web-формы. Данные, содержащиеся в транзакции, обычно складываются из информации, необходимой для выполнения одного или более прикладных процессов предприятия. Как правило, транзакции содержат сильно детализированные наборы элементов данных. К примеру, простая транзакция, представляющая заказанные покупателем товары, может включать следующие элементы данных:

- номер заказа;
- дату заказа;
- номер покупателя;
- метод платежа;
- строки заказа;
- номер строки заказа:
  - элемент заказа или код товара;
  - заказанное количество;
  - единицы измерения товара;
  - цена;
  - вид валюты.

Из этого списка очевидно, что содержание транзакции по заказу сильно детализировано, в отличие от документоориентированного содержания, состоящего просто из длинных строк текста. За редким исключением, данные из транзакции обычно обрабатываются программами и не ориентированы на чтение человеком с помощью браузера. Когда содержимое транзакции предназначено для обработки в прикладной программе, презентация и форматирование, которые требуются для документоориентированного содержания, необходимы редко. Однако стоит отметить, что благодаря самоописывающей природе XML и при условии, что

используются описательные (интуитивно понятные) имена элементов, человек потенциально способен сориентироваться в структуре транзакции и опознать определенный элемент данных. Это может потребоваться при проектировании, тестировании и отладке (листинг 2.1).

Листинг 2.1. Пример ориентированного на транзакции содержимого – информация по заказу

```
<?xml version="1.0" encoding="UTF-8"?>
<Order>
  <OdrerNumber>12345678</OdrerNumber>
  <OdrerDate>2003-02-15</OdrerDate>
  <CustomNumber>P1234567</CustomNumber>
  <PaymentMethod>Cash</PaymentMethod>
  <OrderLines>
    <OrderLineNumber>1</OrderLineNumber>
    <OrderItem>GT-AL-1256</OrderItem>
    <OrderItemQuantity>10</OrderItemQuantity>
    <OrderItemUnitofMeasure>EA
    </OrderItemUnitofMeasure>
    <OrderItemPrice>123.45</OrderItemPrice>
    <OrderItemPriceCurrency>USD
    </OrderItemPriceCurrency>
  </OrderLines>
</Order>
```

Ориентированное на транзакции содержание чаще всего предназначено для обработки в прикладных программах. Возможность извлечь необходимые данные, проверить правильность и обработать ориентированное на транзакции содержание обеспечивается характеристиками из метаданных, такими, как имена элементов, местоположение элементов и их взаимосвязь. Типичная прикладная программа нуждается в возможности проверить тип данных, длину и допустимые значения для каждого из элементов данных (рис 2.6).

XML-транзакции, предназначенные для непосредственного просмотра человеком, обычно требуют применения *преобразования на основе XML-таблицы стилей (XML stylesheet trasformation – XSLT)* или какой-либо формы связывания. XSLT-обработка может преобразовать структуру XML-документа в технологию, предназначенную для отображения, например в HTML (или в другие формы и структуры). Альтернатива преобразованию – *связывание (binding)* – заключается в установлении ассоциации между XML-элементами и элементами некоторой презентационной технологии, например HTML. Существуют и другие формы связывания, например ассоциация элементов базы данных с XML-элементами или презентационными технологиями.

W3C XML-схемы позволяют эффективно описать характеристики метаданных для детализированного содержания, ориентированного на транзакции.

Элемент данных	Тип данных	Длина	Целая часть	Дробная часть	Допустимые значения	Количество повторений
Номер заказа	Numeric	8	8	0		1
Дата заказа	Date (ISO 8601)	10				1
Номер покупателя	String	8				1
Метод платежа	String	20				1
Строки заказа						1..n (повторяющаяся группа)
Номер строки заказа	Integer	5	5	0	+1..+32766	1
Элемент заказа или код продукта	String	15				1
Заказанное количество	Integer	5	5	0	+1..+32766	1
Единица измерения	String	2			EA, DZ, GR	1
Цена за единицу	Decimal	11	8	3		1
Вид валюты	String	3			USD, CAD, GBR	1

Рис. 2.6. Пример характеристик метаданных для транзакции по заказу

За пределами предприятия (Web)

XML-транзакции часто используются для перемещения или обмена данных между предприятием и внешними субъектами, например, покупателями, производствами, торговыми партнерами и совместно работающими группами. Наиболее общими примерами внешних приложений предприятия являются приложения электронной коммерции двух основных типов:

1. Бизнес-Потребитель (Business-to-Consumer, B2C);
2. Бизнес-Бизнес (Business-to-Business, B2B).

Приложения B2C представляют собой Интернет- или Web-приложения, предназначенные для обеспечения ведения бизнеса между предприятием и покупателем (обычно человеком). Приложения B2C часто называют "прилавок". Эти приложения представляют на потребительском рынке сам продукт и сервисную информацию по нему. В зависимости от уровня интереса покупатель изучает информацию о продукте и сервисную информацию и делает выбор: посмотреть дополнительную информацию, сделать покупку или прекратить сеанс работы с программой. Если покупатель решил приобрести товар, приложение должно собрать информацию о выбранном продукте, о его доставке и о платеже. Как правило, эти данные структурируются в виде одной или нескольких транзакций.

Обычно для презентации продукта и представления сервисной информации покупателю используется HTML. XML в комбинации с таблицей стилей также можно использовать для этих целей. В этом случае содержимое XML-документа (информация о продукте и сервисная информация) преобразуется в форму, пригодную для отображения браузером. В приложениях B2C информация, первоначально предоставляемая покупателю (каталог, описание продукта, сервисная информация и т. п.), часто носит описательный характер, т. е. является документоориентированной. И, следовательно, как было сказано ранее, здесь можно успешно использовать DTD-схему.

Транзакция с информацией по заказу, генерируемая в ходе взаимодействия покупателя с B2C Web-сайтом, может создаваться как результат работы HTML-формы (HTML-форма получает данные от покупателя и передает их на сервер в форме транзакции). Эта транзакция также может быть описана посредством XML. После того как транзакция по заказу получена Web-сервером, ее необходимо переработать, чтобы можно было доставить заказ покупателю и получить платеж. Для этого может потребоваться несколько "back end"[2] программ. Если транзакция от сервера была форматирована посредством XML, эту транзакцию можно передать другим программам и на другие платформы (рис. 2.7).

Данные, ориентированные на транзакцию, обычно включают определяемые по отдельности элементы данных, каждый из которых имеет собственный тип данных и специфичные характеристики. В этом случае W3C XML-Схема – предпочтительнее других типов схем.

В отличие от B2C-приложений, в приложениях Бизнес-Бизнес (B2B) в качестве клиентов выступают не отдельные покупатели, а другие предприятия. Такие приложения обычно взаимодействуют с другими бизнес-приложениями, обмениваясь с ними данными ориентированного на транзакции вида или обрабатывая такие данные. Иногда B2B-приложения также предоставляют пользовательский интерфейс, позволяющий сотруднику взаимодействовать с представителями бизнес-партнера.

[2] Под этим термином понимают различные программные системы или их серверные части, занимающиеся непосредственно хранением данных и управлением доступом к ним (т. е. компоненты, реализующие логику постоянства в архитектуре клиент-сервер), – СУБД, ERP-системы, системы управления контентом и т. д. – *Примеч. ред.*

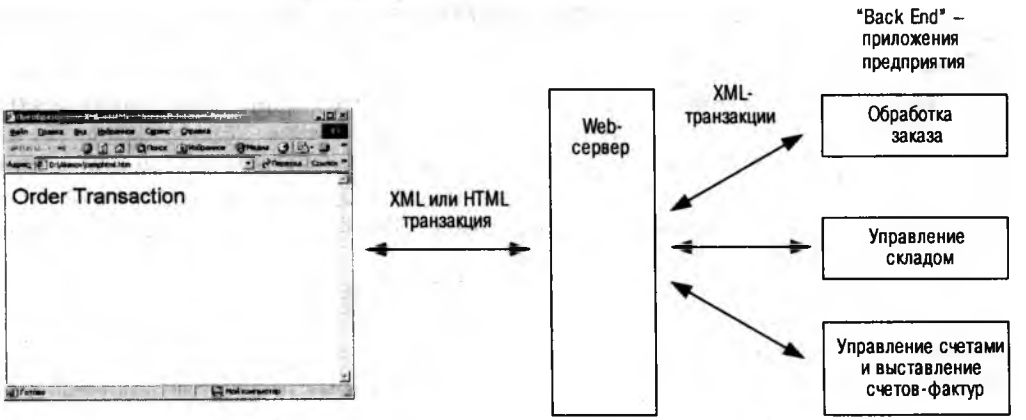


Рис. 2.7. Сценарий B2C-транзакции, использующий XML

Часто группа предприятий, работающих в одной отрасли, договаривается о совместном использовании общего словаря для описания данных, участвующих в обмене данных между предприятиями. Исторически эти отраслевые словари определяются посредством EDI (electronic data interchange – обмен электронными данными) или транзакций, защищенных патентами. Однако в последнее время, преобладающим методом становится определение отраслевых словарей с помощью DTD-схем или W3C XML-Схем (рис. 2.8).

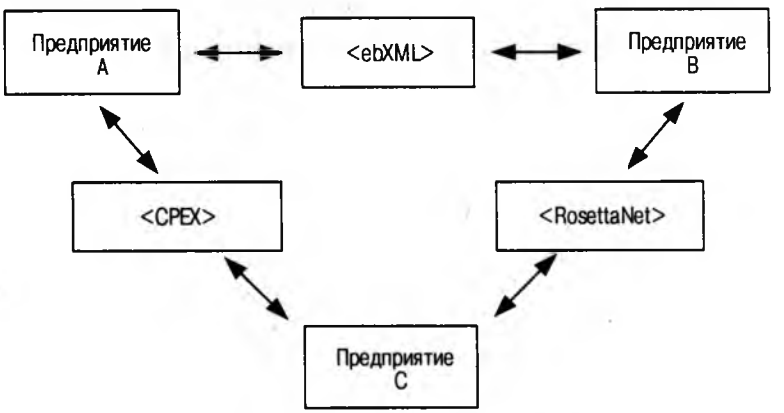


Рис. 2.8. Пример B2B сценария транзакций с использованием XML-словарей

При использовании отраслевых словарей данные транзакции размещаются в одном или нескольких XML-документах. Затем эти XML-транзакции описываются посредством соответствующих схем-словарей. Такой подход гарантирует, что стороны, участвующие в обмене, смогут эффективно интерпретировать и обрабатывать данные транзакции. В настоящее время используется много отраслевых словарей. В некоторых отраслях один и тот же тип транзакции может быть описан в нескольких схемах-словарях. И через некоторое время в этих отраслях необходимо будет прийти к соглашению об использовании общего набора словарей, тем самым сократив их количество.

Так же, как и B2C-транзакции, B2B-транзакции содержат набор детализированных элементов данных. Характеристики метаданных для B2B-данных играют критическую роль. Предприятия, обменивающиеся данными транзакций, должны быть уверены, что каждый из элементов данных имеет необходимый тип данных, длину и допустимое значение. Некоторые отраслевые словари в настоящее время определены посредством DTD. Однако использование в B2B-сценариях DTD-схем менее эффективно. Очевидный выбор здесь – W3C XML-Схемы, предоставляющие возможности типизации данных. Это понимают и многие индустриальные группы и прилагают усилия по переводу словарей, ранее определенных посредством DTD, в W3C XML-Схемы.

Внутри предприятия (EAI и Интранет)

В дополнение к использованию XML во внешних транзакциях и обменах существует возможность применять XML в транзакциях, участвующих в обработке и обмене внутри предприятия. Вот наиболее общие типы внутренних транзакций предприятия:

- Приложение-Потребитель (Application-to-Consumer, A2C);
- Приложение-Приложение (A2A).

Приложения A2C примерно аналогичны B2C-приложениям, но в роли потребителя выступает служащий или представитель данного предприятия. Как и B2C-приложение, A2C-приложение предоставляет информацию человеку. Однако транзакции создаются для обработки "в стенах предприятия", а не для передачи в открытую сеть, например в Web. A2C-приложения часто обслуживают традиционные системы предприятия, такие, как управление складом, сервисная или финансовая.

Одна из задач, стоящих сегодня перед предприятиями, – обеспечить возможность совместного использования данных и обмена данными для систем и приложений разных типов. В большинстве традиционных бизнесов существует долгая история разработки вертикальных приложений и реализации обособленного программного обеспечения, функционирующего на разных технологических платформах, с разными операционными системами и базами данных. Некоторые называют эти вертикальные системы *автономными системами* или "силосными башнями".

Как правило, каждая из этих систем разрабатывалась для поддержки работы определенного подразделения или для некоторой бизнес-функции. Зачастую эти системы сохраняют свои ограниченные определения данных и собственные базы данных. И хотя лежащие в их основе технологии и метаданные могут отличаться, на концептуальном уровне между этими системами есть схожие моменты и общая информация, которой было бы хорошо обмениваться или использовать совместно. Данные о продукте (например, коды продуктов, их описания и размер складского запаса) важны и для приложений управления складом, и для приложений обработки заказа. Для обеспечения взаимосвязанного функционирования этих систем требуется, чтобы информация о продукте, используемая в каждой из них, была определена одинаковым образом или ее можно было преобразовать в общий формат. И менеджер склада, и представитель службы обработки заказов покупателя должны иметь возможность просмотреть информацию о продукте, пользуясь своими системами (рис. 2.9).

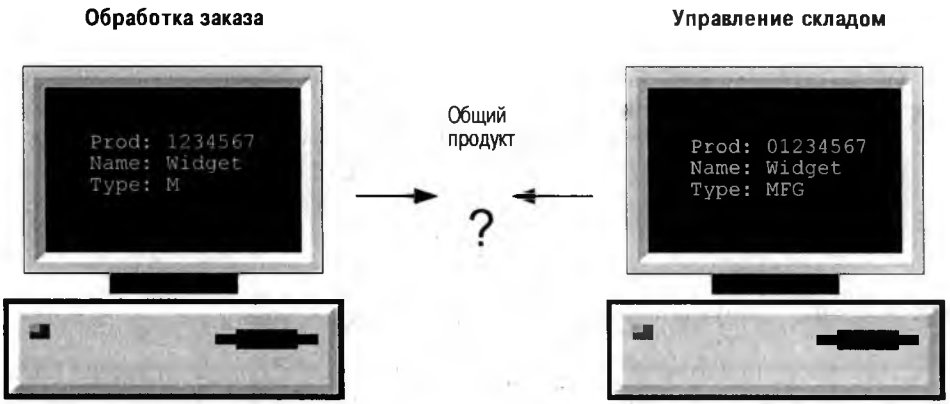


Рис. 2.9. Информация о продукте, используемая в разных системах

Проблема в том, что система обработки заказов и складская система могут пользоваться разными технологиями и окружениями. Каждая из систем может быть разработана со своим собственным определением данных о продукте. Коды продуктов и подобные данные могут иметь разные типы данных, длину и допустимые значения. Обмен, совместное использование или трансляция данных о продукте в форму, которая была бы общей и пригодной для использования в обеих системах, может быть трудным делом. XML здесь можно использовать для описания метаданных об информации по продукту в каждой из систем, а также для создания общего словаря по продукту, предназначенного для обмена данными (рис. 2.10).

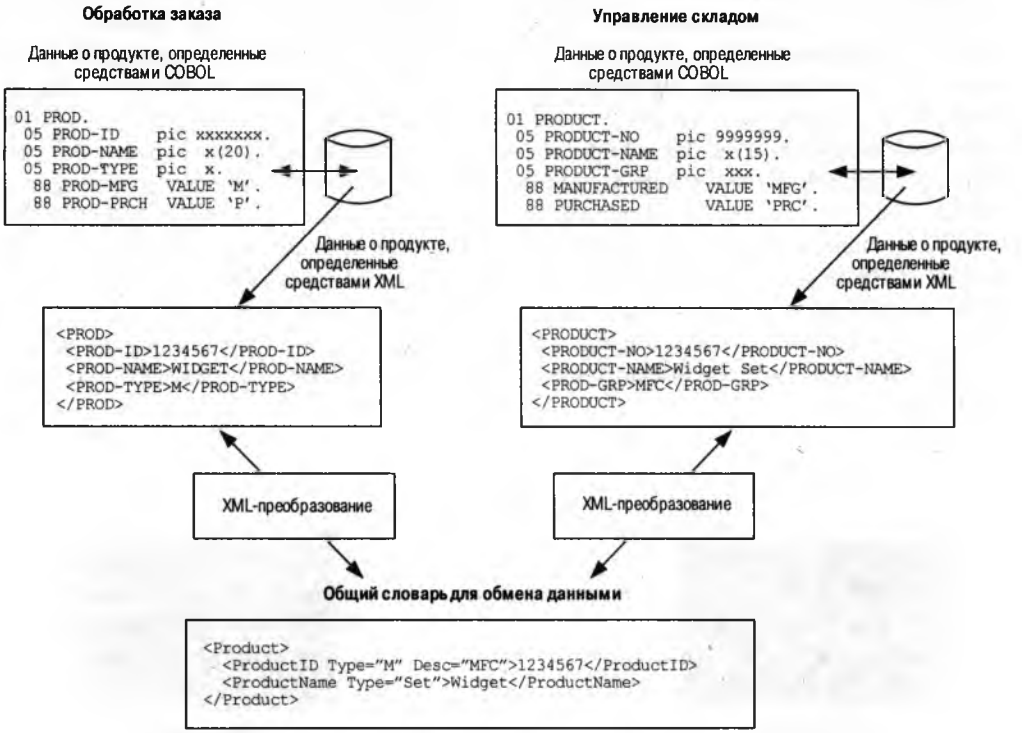


Рис. 2.10. Данные о продукте, преобразованные в общий XML-словарь

В отличие от приложений A2C, приложения A2A обычно не являются интерактивными. Помимо отсутствия пользовательского интерфейса, для них характерен более высокий объем транзакций. В некоторых аспектах A2A-процессы аналогичны традиционным интерфейсным файлам. Данные передаются от одной системы к другой, или происходит обмен данными между системами. Как и в сценариях A2C, формат и характеристики метаданных для информации могут отличаться от одной системы к другой. К наиболее общим формам отличий в метаданных относятся: совокупность типов данных, длина, количество знаков после запятой, формат, допустимые значения и структура. Когда данные транзакции состоят из детализированных элементов данных, наиболее подходящим типом схемы для описания транзакции (и для A2C, и для A2A) являются W3C XML-Схемы. Возможности определить содержимое, наложить ограничения на возможные значения и преобразовать данное содержимое внутренних транзакций предприятия критичны для эффективной интеграции предприятия. Кроме того, стандартизированные W3C XML-Схемы можно использовать

в качестве внутренних словарей (примерная аналогия с отраслевыми словарями) для обеспечения уверенности в том, что обмен данными предприятия происходит в согласованном, общем для всех виде. Интеграция предприятия предоставляет замечательную возможность использовать метаинформационные возможности W3C XML-Схем и их независимость от платформы. Однако стоит отметить, что ни XML, ни W3C XML-Схемы не являются панацеей для интеграции. Интеграция предприятия требует жесткой стратегии, включающей анализ, план движения от исходной точки к цели, оценку различий в метаданных и разработку правил преобразования. Здесь могут встретиться и примеры данных, определенных для разных систем предприятия, но настолько несовместимых, что преобразование и обмен или неосуществимы, или не принесут пользы.

Рекомендация. *W3C XML-Схемы, в общем случае, прекрасно подходят для описания содержимого, ориентированного на транзакции (как вне, так и внутри предприятия).*

Содержание, ориентированное на сообщения

Модель клиент-сервер имеет в технологическом сообществе долгую историю, включающую ее принятие, эволюцию и пору зрелости. Она лежит в основе архитектуры многих приложений, в частности Web-приложений. На фундаментальном уровне, в приложении клиент-сервер разделены интерфейс, прикладная логика и данные. Вот пример клиент-серверной обработки: клиент запрашивает сервисы у одного или нескольких серверов, серверы эти запросы обрабатывают, затем ответ возвращается клиенту (рис. 2.11)

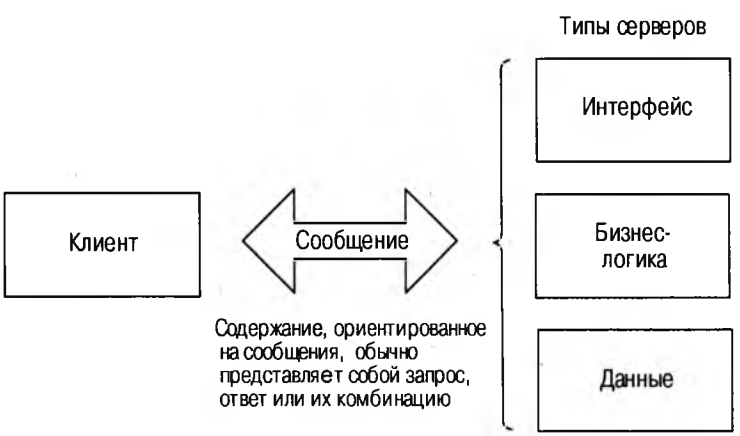


Рис. 2.11. Модель клиент-сервер

Наиболее общая модель обработки сообщений является сочетанием запроса на сервис, определения требуемого сервиса и ответа от сервиса. Эта модель может быть расширена добавлением к функциональному ответу сервиса возможности появления сбоя в его работе или сообщений об ошибке. В простейшем сценарии клиент и сервер обмениваются сообщениями, работая на одной и той же платформе и пользуясь одной технологией. С приходом Интернета и WWW отпали ограничения в виде необходимости использовать единую технологию обработки и платформу. Web-приложения и сервисы пользуются самыми разными технологиями и платформами. В результате, существует необходимость "замаскировать" используемую технологию от клиента.

В типичном сценарии Web-приложения клиенты сервисов и провайдеры сервисов идентифицируются своими универсальными идентификаторами ресурса или IP-адресами и могут находиться в любой точке мира, а также могут использовать почти любую платформу и технологию. В этом случае клиент часто не знает ни физического местоположения сервиса, ни используемой им платформы, ни способа передачи ответа от сервиса. Чем более глобальными становятся сервисы, тем важнее для них независимость от платформы и т. п. Также важна семантика сообщения. Клиент должен говорить на "языке" интерфейса сервиса. Если структура запроса клиента не понятна серверу (провайдеру требуемого сервиса), запрос не попадет к нужному сервису.

Как и содержимое транзакций, содержимое большинства сообщений состоит из детализированных элементов данных. Большинство сообщений узкоспециализированны для той функции или области, для которой они определены, и в результате – очень компактны (ограничены в размере). Содержание сообщений – критическая часть связи между клиентами и сервисами. Примером сервисов, работающих на основе сообщений, являются Web-Сервисы. Более подробную информацию о Web-Сервисах можно найти в главе 10. К общим примерам Web-Сервисов на основе сообщений относятся сервисы, обслуживающие следующие запросы:

- запросы по уровням текущих запасов продуктов;
- запросы по текущему состоянию сборочной линии и производства;
- запросы биржевых котировок;
- запросы погодных условий в разных точках Земли;
- запросы на пересчет сумм из одной валюты в другую.

В каждом случае клиент создает сообщение, включающее данные и параметры запроса, посылает сообщение Web-сервису, в какой-либо форме обрабатывающему этот запрос, и выполняет какие-либо действия над полученным ответом. На более технологическом уровне клиент, пославший запрос сервису, может приостановить свою работу в ожидании ответа или переключиться на другую работу, пока не придет ответ. Web-Сервис получает запрос, проверяет полномочия клиента

и корректность запроса, выполняет запрошенное обслуживание и возвращает сообщение-ответ. Простым примером запроса является запрос у Web-Сервиса текущей температуры в некотором пункте (рис. 2.12).

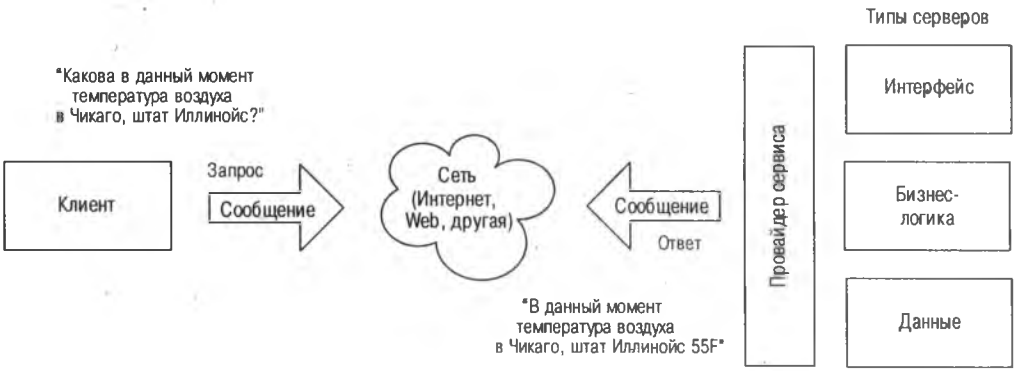


Рис. 2.12. Простой пример работы Web-сервиса

Для описания содержания сообщений (одного и более), как и для описания содержания транзакций, XML является достаточно эффективным решением. Высокая степень детализации сообщения и важность проверки правильности данных из сообщения (например, тип данных или длина) позволяют для описания содержимого сообщения эффективно использовать W3C XML-Схемы. XML становится объединительным языком для описания содержания сообщений между клиентами и Web-Сервисами. Протокол, основанный на XML, известный под названием *простой протокол доступа к объектам (Simple Object Access Protocol – SOAP)* в настоящее время является общепринятым способом описания содержимого сообщений-запросов к Web-Сервисам. Другая форма XML, известная как *язык определения web-сервисов (Web Services Definition Language – WSDL)*, также становится стандартом в описании интерфейса Web-Сервисов и структуры сообщений.

Рекомендация. *W3C XML-Схемы прекрасно подходят для описания содержимого, ориентированного на сообщения.*

Выбор типа схемы

На выбор схемы влияет не только назначение XML-документа, но и ряд других характеристик. Во-первых, вид обработки или использования содержимого. Данные, предназначенные для обработки несколькими способами или приложениями нескольких типов, могут потребовать выбора и использования сразу

нескольких типов схем. Например, документоориентированное содержание, которое предполагается визуализировать в браузере для прочтения человеком, может потребовать использования DTD-схемы. Если этот же документ будет также использоваться в сценарии B2B, где получатель данных осуществляет разбор документа с целью построения индекса ключевых слов, то дополнительно может потребоваться тип схемы, предназначенный для более детального описания данных, например W3C XML-Схемы.

Для определения наиболее подходящего типа (или типов) схемы, проектировщик должен учитывать следующие обстоятельства:

- ориентация данных (документ, транзакция или сообщения);
- структурная организация (иерархическая, реляционная или свободная форма);
- гранулярность (дискретно определяемые элементы данных или строки текста);
- предполагаемый вид обработки (презентация или приложение бизнес-логики);
- предполагаемый потребитель (человек или автоматизированное приложение);
- независимость от платформы (конкретная платформа или платформнонезависимое приложение);
- стандартизация и повторное использование (типы данных или повторноиспользуемые конструкции);
- соображения производительности.

Ориентация или вид данных, которые требуется представить средствами XML, – очень важный фактор в выборе типа схемы. Документоориентированным данным может соответствовать иной тип схемы, нежели данным транзакции или сообщения. Также важна структурная организация данных. Чаще всего данные можно организовывать иерархически (структура типа отец-сын-брат, как оглавление книги), или как набор реляционных данных (наподобие реляционных таблиц, столбцов и строк), или в свободной форме (неупорядоченные или слабо упорядоченные элементы данных, фрагменты или строки текста).

Важно учитывать и степень детализации данных. Если содержимое состоит из множества отдельных элементов данных, то схема, поддерживающая типы данных, просто необходима. Если те же данные предназначены для чтения в браузере человеком, может потребоваться более документоориентированный тип схемы.

Важную роль играет и предполагаемый потребитель данных. Если это – человек, необходимость строгой типизации и верификации данных может отсутствовать (но необязательно). Если же потребитель данных – автоматизированное приложение, следует использовать W3C XML-Схему. Поскольку XML-документ и его содержимое независимы от платформы, верификация с использованием некоторых типов схем может представлять проблему. Не все типы схем поддерживаются во всех техноло-

гических продуктах. Например, XDR-схемы. Поддержка XDR-схем со стороны браузеров в основном ограничивается браузерами, включающими "MSXML" – синтаксический анализатор от Microsoft. Если заранее не известен провайдер или тип браузера, возможно, XML, ссылающийся на XDR-схему, не везде будет иметь поддержку. Напротив, W3C XML-Схемы получают все более широкое распространение. Фирма Microsoft заявила о том, что она будет поддерживать XML-схемы (о чем свидетельствует и недавний "MSXML4" – синтаксический анализатор от Microsof).

Характеристики данных	DTD	XDR[a]	XSD
Ориентация данных			
Документоориентированные данные	X	X	X
Данные, ориентированные на транзакцию		X	X
Данные, ориентированные на сообщение		X	X
Структурная организация			
Иерархическая	X	X	X
Реляционная (косвенно используемая с помощью идентификаторов, указателей и структуры)			X
Свободная форма (в основном неструктурированная)	X		
Степень детализации			
Детализированные или дискретно определенные элементы данных		X	X
Длинные строки текста	X		X
Предполагаемый потребитель			
Человек	X	X	X
Автоматизированное приложение		X	X
Вид обработки или потребления			
Визуальный (презентация или визуализация)	X	X	X
Приложение с бизнес-логикой		X	X

Рис. 2.13. Сравнение общеупотребимых типов XML-схем

Характеристики данных	DTD	XDR[a]	XSD
Независимость от платформы			
Определенная платформа		_[b]	
Обработка на любой из платформ	X		X
Типы, стандартизация и повторное использование			
Строго типизированные данные		X	X
Нестандартные или производные типы данных		X	X
Обширная поддержка фасетов типов данных			X
Повторноиспользуемые внутренние конструкции		X	X
Повторноиспользуемые внешние конструкции	X	_[c]	X
Возможность переопределения повторноиспользуемых конструкций			X

Рис. 2.13. Сравнение общеупотребимых типов XML-схем (Окончание)

[a] Предполагается, что в случае их поддержки XSD-типы схем более предпочтительны для замены XDR-типов схем.

[b] Поддержка XDR-типов схем обеспечивается в основном в продуктах фирмы Microsoft и ее партнеров. Дополнительно Microsoft включила поддержку XML-схемы (XSD) во многие свои продукты.

[c] Пространство имен, ссылающееся на внешние схемы, описывается в XML Data Reduced "Note". Однако поддержка этой возможности синтаксическими анализаторами варьируется.

Стандартизация и повторное использование также важны. Среди перечисленных ранее типов схем строгие типы данных поддерживаются только в XDR и W3C XML-Схемах. XDR-схемы допускают ограниченную настройку типа данных, и они поддерживаются не всеми синтаксическими анализаторами. В отличие от этого, W3C XML-Схемы обеспечивают расширенные возможности в части нестандартных и производных типов данных. До определенной степени все три типа схем поддерживают формы повторного использования. Однако W3C XML-Схемы обеспечивают наиболее прочную поддержку повторного использования.

Факт. *В зависимости от характеристик данных, их назначения и количества обрабатывающих их приложений может потребоваться использовать более одного типа схемы.*

Рекомендация. *Если необходимо использовать только один тип схемы, рассмотрите в качестве первого претендента W3C XML-Схемы.*

В качестве простого метода выбора схемы можно принять процесс, при котором характеристики данных и необходимых описательных метаданных сравниваются с возможностями каждого типа схемы. Некоторые характеристики данных поддерживаются более чем одним типом схем, и степень этой поддержки различна. Хорошее соответствие достигнуто, когда характеристики выбранного типа схемы наилучшим образом соответствуют требуемому характеру использования XML (рис. 2.13). Этот способ прост в использовании, но он может не дать окончательный результат. Однако он обеспечивает общий подход к определению схем – лучших кандидатов. Если оказалось, что для содержимого транзакций или сообщений применимы несколько типов схем, велика вероятность того, что лучшим выбором в этом случае окажутся W3C XML-Схемы.

К этому моменту должно быть очевидно, что в качестве формы метаданных XML и особенно W3C XML-Схемы предоставляют проектировщикам данных дополнительные возможности по применению, расширению и углублению их мастерства. Использование XML как метода описания содержимого документа, транзакции или сообщения будет продолжать нарастать, и потенциальные выгоды этого использования очевидны. Однако здесь существуют и некоторые риски. Без эффективного проектирования, разработки и техники практического использования существует значительная опасность разрастания числа нестандартных и несопоставимых определений информационных активов предприятия. К наиболее очевидным областям приложения к XML практического опыта проектирования архитектуры данных относятся: именование (таксономия), типы и фасеты данных, моделирование, архитектура и техника повторного использования.

Глава 3

Необходимость стандартов именования

Создание описательных имен для элементов данных, полей и традиционных структур данных исторически является прерогативой проектировщиков данных. Имена элементов данных представляют собой форму описательной классификации и идентификации, обеспечивающей контекст. Поскольку XML – самоописывающий метаинформационный язык, назначение имен элементам и атрибутам XML относится к процессу построения архитектуры данных. Традиционные подходы к присвоению имен элементам данных имеют долгую историю и зачастую относятся, в основном, к области технологий и продуктов, связанных с базами данных. Применение этих подходов к элементам и атрибутам XML может привести к появлению запутанных, сложных, насыщенных абревиатурами, чрезмерно многословных имен.

Чтобы раскрыть описательные возможности XML и использовать преимущества строгой стандартизации именования, следует придерживаться ряда правил. Во-первых, все имена элементов и атрибутов должны быть:

- содержательными (информативными);
- краткими;
- без аббревиатур или акронимов (за исключением общеизвестных);
- разумной длины.

Некоторые из проектировщиков данных имеют на вооружении адаптированные подходы к именованию, являющиеся описательными, интуитивными и не нарушающие основные синтаксические правила XML. Когда эти подходы применяются последовательно и хорошо принимаются и технологическим сообществом, и бизнес-сообществом, их вполне можно адаптировать для именования элементов и атрибутов XML.

Возможность. *Когда существующие на предприятии стандарты и процессы именования данных являются описательными, интуитивными, общепризнанными и не нарушают синтаксических правил XML, они могут быть адаптированы для присвоения имен атрибутам и элементам XML.*

Рекомендация. *Имя элемента или атрибута XML должно быть интуитивно понятным ("правило интуитивности").*

И процесс именования, и части имени элемента или атрибута могут быть разными. Однако должно неизменно выполняться правило: имя элемента или атрибута должно быть интуитивно понятным. Даже не технарь должен быть в состоянии прочитать образец XML-документа и уяснить себе, что за данные в нем содержаться. Многие из традиционных подходов к именованию используют жестко заданную структуру имени и жесткие физические ограничения (например, на длину имени), что может отрицательно сказаться на возможности их применения для XML.

Помимо жестких структурированных форм, традиционные стандарты и подходы к именованию часто включают использование стандартных аббревиатур, стандартных акронимов и слов – признаков принадлежности к определенному классу (class word). В некоторых стандартах именования слово класса также используется как метод описания типа данных. Этот подход может дать положительный результат при его использовании для баз данных и физических компонентов баз данных (например, столбцов, элементов и полей). Однако подобные подходы могут привести к созданию имен элементов данных с очень конкретным контекстом, которые невозможно повторно использовать в широких масштабах. Таким образом, ограничивается возможность совместного использования данных разными приложениями или возможность повторного использования стандартных структур данных.

Вместо того чтобы пытаться насаждать подобные подходы к именованию в XML, лучше адаптировать некоторые из этих подходов, избегая тех, которые приводят к нарушению правила интуитивной понятности имени. Но для того чтобы проектировщики данных смогли отличить приемлемые подходы от неприемлемых и сумели адаптировать их, им необходимо получить базовые знания по синтаксису, именованию и ограничениям XML.

Характеристики стандарта именования

Стандарт именования – это форма именования или идентификации. В наиболее общем виде стандарты именования также включают иерархическую классификацию и группирование. Имя, присвоенное в соответствии с определенным стандартом именования, не только идентифицирует некоторый объект, но также

позволяет классифицировать его или отнести его к группе объектов, имеющих схожие характеристики. Например, человек может иметь дополнительные характеристики в виде пола, роста и веса, а также может относиться к прочим подобным группам, например к млекопитающим или двуногим.

Стандарт именования в проектировании данных классифицирует, идентифицирует и описывает содержимое (например, значения) элемента данных. Человек, читающий имя элемента данных, должен понять, что содержится внутри этого элемента, а также определить базовые характеристики относящихся к нему метаданных. В качестве формы именования некоторые стандарты именования данных также описывают типы данных, допустимые значения и прочие характеристики метаданных.

Подобно элементам данных, XML-контейнеры (элементы и атрибуты) описываются именованными тегами. Синтаксически тег XML-элемента состоит из открывающей и закрывающей угловых скобок, в которые заключено имя элемента (например, "<ABC/>"). Синтаксис XML-атрибута похож на синтаксис HTML-атрибутов. Они задаются в форме пары "атрибут-значение" (например, xyz=" "). Существует несколько других синтаксических ограничений на XML-имена. В основном к ним относятся следующие.

- Имя элемента или атрибута может состоять из символов верхнего регистра, нижнего регистра или их комбинации (например, "<ABC/>", "<abc/>", "<Abc/>" – допустимые имена элементов).
- Имена элементов и атрибутов чувствительны к регистру (т. е. "<ABC/>" и "<abc/>" обозначают совершенно разные элементы).
- Пробельные символы (white space) в составе имени недопустимы (т. е. "<ABC DEF GHI/>"– некорректное имя, а "<ABCDEFGHI/>"–корректное).
- Имя не должно начинаться с цифры (т. е. "<9ABC/>"-неправильно, а "<ABC9/>"–правильно).
- Существует несколько зарезервированных слов, которые нельзя использовать как часть имени (например, "XML", "xml", "LANG", "lang" не могут быть частью имени элемента или атрибута).
- Не существует ограничения на длину имени (только здравый смысл и, возможно, ограничение конкретного синтаксического анализатора).

При использовании этих правил совместно с продуманным подходом к стандарту именования получаются имена элементов и атрибутов, ясно описывающие содержание этих контейнеров.

Составные части имени

Имя XML-контейнера (элемента или атрибута) складывается из одной или нескольких составных частей. Составные части имени (particles) – это слова, аббревиатуры или акронимы, которые вместе и составляют полное имя. В качестве частей имени могут выступать существительные, прилагательные, наречия и прочие описательные модификаторы (рис. 3.1).

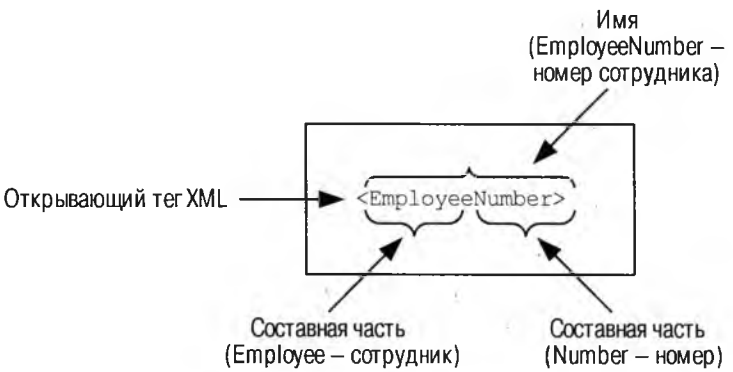


Рис. 3.1. Составные части имени

Традиционные стандарты именования предпочитают не использовать в составе имени артикли (“a”, “ah”, “the”), предлоги (“with”, “to”, “by”, “of”), местоимения (“I”, “we”, “they”) и имена собственные (“James”, “Sally”, “Washington”, “Chicago”). И я рекомендую вам избегать использования артиклей, предлогов, местоимений, и имен собственных в составе имен элементов и атрибутов XML.

Рекомендация. *Избегайте использования артиклей, предлогов, местоимений и имен собственных в составе имен XML.*

Рекомендация. *Избегайте использования акронимов и аббревиатур, за исключением легко узнаваемых и общераспространенных.*

Развитие бизнеса и прогресс в области технологии породили огромное количество акронимов и аббревиатур. Из-за их явного переизбытка многие отрасли вынуждены были опубликовать списки аббревиатур и акронимов. Частое использование аббревиатур и акронимов в качестве составных частей имен приводят к тому, что эти имена перестают быть интуитивно понятными (это не касается широко известных аббревиатур и акронимов, например FBI, CIA, UN). Чтобы интерпретировать такое имя, вам потребуется обратиться к списку аббревиатур.

Во многих практикуемых в проектировании данных подходах к именованию предписывается в качестве добавочного классификатора, включаемого в состав имени, использовать *слово класса (class word)*. Слово класса служит для дополнительного описания именуемого домена данных (с точки зрения допустимых значений, типа данных или аналогичных ограничений). Хотя слово класса, характеризующее допустимые значения или тип данных, может применяться при именовании элементов или столбцов базы данных, его присутствие в составе имени реально никак не влияет и не накладывает никаких ограничений на именуемые данные. То же самое можно сказать и про добавление слова класса в имена элементов и атрибутов XML. Ограничения в виде строгого типа данных и допустимых значений здесь накладываются использованием схемы. Верификация XML-документа на основании W3C XML-Схемы позволяет проверить тип, допустимые значения и т. п., а внешний вид имен элементов и атрибутов никак не влияет на этот процесс. Если в состав имени некоторого элемента было добавлено слово класса (например, свидетельствующее об определенном типе данных или допустимых значениях) и затем содержимое этого элемента было изменено при перемещении документа в другую систему, возникнет расхождение между именем и реальным содержанием. Таким образом, добавление слова класса в имена элементов и атрибутов XML мало что дает, а в отдельных случаях может ввести в заблуждение разработчика приложения.

Рассмотрим элемент данных, содержащий значение даты "2003-03-15". На платформе одной базы данных это значение может быть описано как тип данных "Date" (дата), а на другой – как тип "Character" (символьный). Включение в имя данного элемента слова класса, свидетельствующего о типе "Character", может привести к ошибочной интерпретации значения, содержащегося в элементе.

Кроме того, многие из слов, которые используются как слова класса, часто применяются в другом значении. Слово класса "Number" (число) часто некорректно используется для представления концепции идентификатора (например, "Custom-Number" [номер заказа]), тогда как фактически элемент с таким именем может содержать символы или смесь из символов и чисел. К примеру, во многих организациях используется элемент данных с именем "Part Number" (номер подразделения), в то время как содержимым этого элемента может быть значение "A-123456".

В этом же ряду – неоднозначность, связанная с различием между порядковыми и количественными числами. Порядковые номера обозначают порядок расположения или последовательность и обычно не участвуют в вычислениях, связанных с бизнес-логикой (например, бессмысленно умножать один порядковый номер на другой). В отличие от них, количественные числа в вычислениях используются. Слово класса, обозначающее числовой тип данных, но не делающее различий между порядковыми и количественными числами, также может ввести в заблуждение разработчика приложения.

Рекомендация. *Используйте слова класса только тогда, когда они коротки, представляют понятное и исчерпывающе определяют ограничение.*

Если слово класса – содержательное, краткое и однозначное, его можно включить в состав имени XML-элемента. В случае с именем элемента <Employee-BirthDate/>(дата рождения сотрудника) использование слова "Date"(дата) вполне уместно. При одном взгляде на этот элемент становится ясно, что он содержит. Однако будьте осторожны, поскольку такое слово класса, как "Date", не подразумевает, не указывает и не ограничивает формат даты. И хотя ясно, что содержимым упомянутого выше элемента должна быть дата, непонятно, какой из следующих форматов она должна иметь:

- ГГГГ-ММ-ДД;
- ММ-ДД-ГГГГ;
- Месяц ДД, ГГГГ;
- ДД Месяц ГГГГ;
- другие варианты формата.

Постороннее замечание: в соответствии с типами данных, поддерживаемыми W3C XML-Схемами[1], все данные типа "Date", определяемые в XML, должны придерживаться стандарта на дату и время ISO 8601[2] или должны быть описаны так, чтобы ясно информировать читателя о формате даты. Однако не существует никаких определяющих или синтаксических ограничений, налагаемых использованием слова класса "Date".

Регистр символов в имени

Другой важной характеристикой имени элемента данных и в частности XML-имени является допустимый регистр символов. В нашем обсуждении предполагается, что для записи XML-тегов используется английский язык (но точно так же для этого можно использовать и другие языки). XML допускает несколько вариантов использования регистра символов. В традиционных именах элементов данных использование регистра символов часто ограничивается репозитарием метаданных, инструментом моделирования данных или инфраструктурой базы данных. В некоторых базах данных имена элементов должны состоять только из

---

[1] World Wide Web Consortium (W3C). XML Schemas. W3C Recommendation May 2, 2001. Доступен по адресу http:/www.w3.org/TR/2001/REC-xmlschema-1-20010502/(структуры), http:/www.w3.org/TR/2001/REC-xmlschema-2-20010502/(типы данных). – *Примеч. авт.*

[2] International Standards Organization (ISO). ISO 8601, Date and Time TC154 Technical Committee. http:/www.iso.ch/iso/en/stdsdevelopment/tc/tclist/TechnicalCommitteeDetailPage.TechnicalCommitteeDetail?COMMID=3827. – *Примеч. авт.*

символов верхнего регистра. В других – только из символов нижнего регистра. XML не ограничивает нас в выборе регистра, здесь допустима любая из форм имени, показанных в табл. 3.1.

Таблица 3.1. Регистр символов в именах тегов XML

Имя элемента	Регистр символов
<EMPLOYEENUMBER/>	Все символы в верхнем регистре
<employeenumber/>	Все символы в нижнем регистре
<EmployeeNumber/>	Смесь регистров. Слова, составляющие имя начинаются с заглавных букв
<employeeNumber/>	Смесь регистров. Первое слово – с прописной, второе с заглавной буквы

Каждый из приведенных в табл. 3.1 вариантов использования символов разного регистра в имени является допустимым в XML. Однако проектировщикам данных всегда следует помнить о самоописывающих возможностях XML и тщательно выбирать из возможных вариантов имен наиболее интуитивно понятные. Из приведенных примеров видно, что использование в имени только одного регистра символов (т. е. только верхний регистр или только нижний регистр), без какого-либо визуального выделения его составных частей, затрудняет интерпретацию имени (мы помним, что включение пробелов для наглядного разделения XML-тега на составные части не допускается). Таким образом, если к противному не обязывают существующие на предприятии стандарты именования или иные ограничения, следует избегать XML-имен, состоящих из символов только верхнего или только нижнего регистра.

Рекомендация. *Если не существует обязательных ограничений (например, в виде уже существующих технологий и инструментальных средств, поддерживающих лишь символы одного регистра), старайтесь в имени элемента или атрибута XML использовать символы обоих регистров.*

В частности, представляет интерес такой вариант совместного использования символов разных регистров в имени, при котором первый символ каждой из частей имени принадлежит верхнему регистру, а остальные – нижнему (например, <EmployeeNumber/>). Такой вариант использования символов разного регистра обеспечивает в большинстве случаев простую и понятную форму наглядного разделения составных частей имени.

Разделители составных частей имени

Большое значение для имени элемента имеет возможность визуально разделить и логически собрать вместе отдельные составные части имени (подобно тому, как читается предложение или фраза). В английской и многих других письменностях для разделения слов в предложении используется пропуск или пробел. Однако использование пробелов в именах элементов и атрибутов XML запрещено. Но без той или иной формы визуального деления имени на составные части длинные имена, складывающиеся из нескольких слов и акронимов, очень трудно воспринимать. Без визуального деления имени на составные части оно воспринимается как одно "долгоиграющее" слово. На примере (рис. 3.2) составные части имени "EMPLOYEE" и "NAME" легко различимы, поскольку для разделения используется пробел, но это недопустимо в XML.

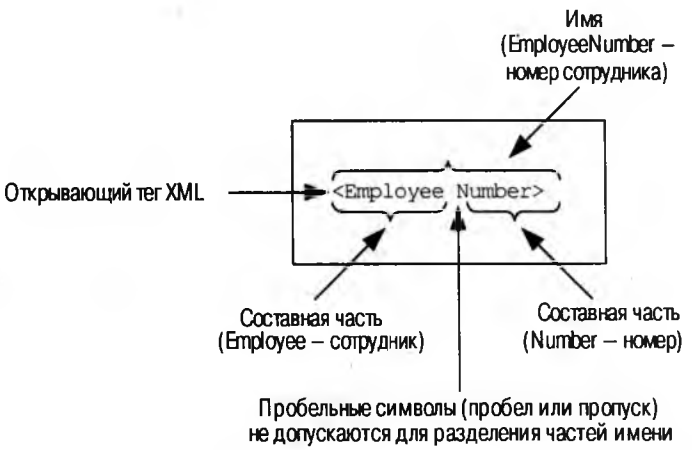


Рис. 3.2. Пример недопустимого разделителя составных частей имени

Рекомендация. *Разделители между составными частями имени используются только для обеспечения читабельности имен элементов и атрибутов.*

Некоторые традиционные подходы к именованию элементов данных включают ту или иную форму символа-разделителя между составными частями имени. Для именования физической базы данных, например столбцов таблицы базы данных, в качестве символа-разделителя часто используется подчеркивание ("_"). Некоторые процедурные языки программирования, например COBOL, используют в качестве разделителя тире ("–"). В других, таких, как Visual Basic, для этой цели используется точка (".") XML позволяет использовать в качестве символа-разделителя имени элемента или атрибута и подчеркивание, и тире, а также двоеточие (":"). Однако, несмотря на формальное разрешение, использовать в качестве разделителя

двоеточие не рекомендуется, поскольку у двоеточия есть и другая смысловая нагрузка – оно используется при указании префикса пространства имен. В любом случае, символ-разделитель предназначен для визуального выделения составных частей имени и улучшения восприятия этого имени.

Недостаток использования для разделения подчеркивания, тире или точки в том, что каждое их вхождение добавляет одну дополнительную позицию в итоговое имя. Хотя это не имеет прямого отношения к XML, многие технологии ограничивают максимальный размер столбца или имени элемента данных. Проблема в том, чтобы совместить визуальное разбиение имени элемента данных с ограничениями технологии. В этом смысле более приемлемым может быть использование для разделения символов верхнего регистра в начале каждой из частей имени, вместо добавления в имя дополнительного символа-разделителя.

Факт. *Для визуального деления имени элемента или атрибута в XML можно использовать символы подчеркивания ("_"),тире ("–"), и точки (". "). Однако каждое вхождение в имя символа разделителя увеличивает его длину еще на одну позицию.*

Как было сказано, альтернативный способ визуального деления имени заключается в использовании символов разного регистра. При этом отпадает необходимость введения в имя символов-разделителей, но обеспечивается такой же эффект, как при их использовании. Применение техники "camel case" (когда первый символ каждой из составных частей имени находится в верхнем регистре, а все остальные – в нижнем) наглядно отделяет части имени друг от друга. Хотя реально составные части сцеплены и вместе составляют единое имя элемента или атрибута, визуально они выделяются и имя становится более читабельным.

В табл. 3.2 приведены разные формы описания элемента данных "EMPLOYEE NAME" (фамилия сотрудника) на языке XML.

Таблица 3.2. Примеры имен элемента данных и разделителей

Имя XML-элемента	Используемый способ разделения
<EMPLOYEENAME/>	Нет
<EMPLOYEE_NAME/>	Подчеркивание
<EMPLOYEE-NAME/>	Тире
<EMPLOYEE.NAME/>	Точка
<employeename/>	Нет
<employee_name/>	Подчеркивание
<employee-name/>	Тире
<employee.name/>	Точка
<EmployeeName/>	"camel case"

При использовании XML каждое из приведенных в таблице имен элементов является синтаксически правильным. Однако в первом примере составные части имени плохо различимы. В остальных примерах показаны эффективные способы визуального разделения имени на составные части. Но в тех из них, где для разделения используется дополнительный символ, длина имени увеличивается, а следовательно, возрастает объем XML-документа, в котором это имя используется. Если длина имени – существенный фактор или если действуют директивы, ограничивающие эту длину, проектировщик данных может использовать в имени какие-либо варианты сокращений или другие ограничивающие приемы.

Кроме того, если XML-файл является транзакцией, включающей множество элементов с именами значительной длины, суммарный размер транзакции также будет велик. Например, если XML-транзакция включает элементы "<employee_name>" для всех служащих предприятия, может получиться, что несколько тысяч символов в этом документе будут занимать только символы-разделители. И хотя здесь нет прямой зависимости от размера документа, производительность обмена транзакциями в сети может понизиться. Заметим однако, что из этого не вытекает необходимость испоъльзования "шифрованных" имен элементов. Напротив, это должно стать дополнительным аргументом в пользу создания эффективных имен элементов и атрибутов и рачительного использования каждого символа имени.

Рекомендация. *Использование в именовании подхода "camel case", при котором каждый начальный символ каждой составной части имени находится в верхнем регистре, а остальные символы – в нижнем, визуально разделяет составные части. Отпадает необходимость введения в имя дополнительных символов-разделителей.*

Пример "<EmployeeName/>" использует подход "camel case", при котором начальный символ каждой составной части имени находится в верхнем регистре, а остальные символы – в нижнем. Это дает эффект визуального отделения друг от друга составных частей имени. Кроме того, отсутствие символов-разделителей сокращает длину имени.

Длина имени

Спецификация на XML не оговаривает точный верхний предел длины для имени элемента и атрибута. Теоретически эта длина может составлять сотни символов. Однако чрезмерная длина имен элементов и атрибутов может понизить эффективность обработки данных. Слишком большая многословность имени не рекомендуется, но и избыток сокращений в имени элемента или атрибута также снижает эффективность использования XML. Имена, изобилующие сокращениями, тяжело читать и интерпретировать, так же, как и очень многословные имена. Говоря о длине имени элемента или атрибута XML, следует иметь в виду ряд важных характеристик. Имя элемента или атрибута XML должно:

- иметь оправданную длину;
- не быть чрезмерно многословным;
- не включать большого количества сокращений или "шифров".

Рекомендация. *Имена атрибутов и элементов XML должны иметь оправданную длину (быть не слишком многословными, но и не слишком сокращенными).*

Рекомендация. *Длина имен атрибутов и элементов XML должна быть оптимизирована и согласована с большинством общих источников данных и получателей данных.*

При определении наиболее эффективного подхода к именованию элементов и атрибутов XML очень важно учитывать то, каким образом XML будет использоваться или обрабатываться. Если предполагается использовать XML для обмена транзакциями или сообщениями, можно сделать вывод, что будет существовать один или несколько обрабатывающих процессов. И возможно (но не обязательно), данные для этих процессов будут извлекаться из XML-документа и помещаться в базу данных. Нельзя сказать, что все XML-транзакции будут напрямую сохраняться в базе данных. Однако такое вполне возможно. Общий подход в определении максимально допустимой длины имен элементов и атрибутов XML заключается, в таком случае, в сопоставлении, оптимизации и выравнивании ограничений на имена исходной и конечной платформ баз данных. Один из последних документов правительства США на эту тему устанавливает, что длина XML-элементов не должна превышать 30 символов[3].

Уровень абстрактности имени

Уровень абстрактности имени является еще одним важным аспектом стандарта именования элементов и атрибутов XML. Традиционные подходы к именованию элементов данных порождают очень конкретизированные имена. Часто эти имена отражают конкретную реализацию, а не контекст описания. Рассмотрим следующие примеры.

- CORPORATE_EMPLOYEE_IDENTIFIER_NUMBER
  (идентификационный номер сотрудника).
- PAYROLL_EMPLOYEEIDENTIFIER_NUMBER
  (идентификационный номер сотрудника в платежной ведомости).
- VEHICLE_PARKING_TAG_EMPLOYEE_IDENTIFIER_NUMBER
  (идентификационный номер сотрудника на автостоянке).

[3] Crawford M, Egan D, Jackson A. Federal Tag Standarts for Extensible Markup Language. GS018T1, p 27. Logistics Management Institute (LMI), June 2001. Находится по адресу http://xml.gov/. – *Примеч. авт.*

Если в каждом из этих примеров представлены схожие данные "EMPLOYEE IDENTIFIER" (идентификатор сотрудника), которые описываются одинаковыми метаданными (например, типом данных и длиной), мы можем считать, что каждый из этих экземпляров содержит данные одного и того же типа. Однако каждое из этих имен элементов данных описывает собственное приложение или использование понятия "EMPLOYEE NUMBER" (номер сотрудника). Этот уровень конкретности мог стать результатом, унаследованным от соответствующей базы данных, требующей наличия уникального имени столбца в пределах одной таблицы, или мог стать следствием применяемых стандартов именования. В любом случае, возможности повторного использования таких данных об идентификационном номере в других контекстах сильно ограничены.

Принимая во внимание, что XML-транзакция может участвовать в обмене данными в пределах предприятия и может содержать данные, имеющие одно общее определение, эффективнее использовать более абстрактное имя элемента. Можно определить элемент для хранения "EMPLOYEE IDENTIFIER" (идентификатора сотрудника) в качестве стандарта предприятия и затем использовать его в различных контекстах. Если отдельная XML-транзакция, содержащая данные о сотруднике, участвует в обмене между несколькими системами, она может содержать, ссылаясь на соответствующее описание, элемент "EMPLOYEE IDENTIFIER" как дочерний элемент в составе разных родительских элементов. При этом родительский элемент будет обеспечивать конкретный контекст использования идентификатора сотрудника, как бы назначая ему определенную "роль" (листинг 3.1).

Листинг 3.1. Пример транзакции по служащему, демонстрирующей множественное использование "EMPLOYEE IDENTIFIER" (идентификатора сотрудника)

```
<?xml version = "1.0" encoding = "UTF-8"?>
<ExampleTransaction>

  <CorporateHumanResourcesData>
    <EmployeeIdentifier>12345</EmployeeIdentifier>
  </CorporateHumanResourcesData>

  <PayrollData>
    <EmployeeIdentifier>12345</EmployeeIdentifier>
  </PayrollData>

  <VehicleParkingTagData>
    <EmployeeIdentifier>12345</EniployeeIdentifier>
  </VehicleParkingTagData>

</ExampleTransaction>
```

Существует несколько разных подходов к применению родительских элементов в качестве ролей или классификаций, и у каждого свои достоинства и недостатки, (см. главу 8). Когда тип роли элемента определен (путем вложения в родительский эле-

мент или как классифицирующий атрибут), этот дочерний элемент может получить абстрактное имя. Вместо того чтобы определять XML-элементы с уникальными именами для каждого из потенциальных применений понятия "идентификатор сотрудника" (в списке сотрудников, в платежной ведомости, на стоянке), можно определить один общий элемент и затем использовать его повторно в разных контекстах.

Возможность. *Абстрактные элементы и атрибуты XML, из имени которых исключены внутренние "роли" или "классификации" и для которых уточнение их текущего назначения производится с помощью их родительских элементов или описательных атрибутов, хорошо подходят для повторного использования.*

Как видно из предыдущего примера, с транзакцией по сотруднику, имена абстрактных XML-элементов иногда полезны. Такой подход может использоваться для поддержки стандартов метаданных предприятия и повторного использования. Однако здесь следует быть осторожным. Одна из наиболее сильных сторон XML – его самоописывающая природа. И даже в случае имен абстрактных элементов и атрибутов необходимо придерживаться правила интуитивности. Если имя элемента настолько неопределенное, что вызывает двусмысленность или ставит в тупик, то на пути его использования существует значительные опасности. Чрезмерно абстрактные имена элементов могут быть неверно истолкованы или применены в контексте, не имеющем смысла.

Факт. *Слишком абстрактные имена элементов и атрибутов XML могут привнести излишнюю сложность и даже стать причиной неверного толкования.*

Расширяя предыдущий пример, мы можем назвать наш элемент просто "Identifier" (идентификатор), например "<Identifier />", и позднее описать новый родительский ролевой элемент, чтобы добавить "Employee" (сотрудника). В этом случае элемент "Identifier" претендует стать стандартом предприятия и потенциально будет использоваться в различных контекстах. Проблема в том, что элемент "Identifier" слишком абстрактен и его описание и применение вызывает вопросы. Даже при корректном использовании этого элемента, для обеспечения необходимого в конкретном случае контекста его использования потребуется слишком большой уровень вложенности (листинг 3.2).

Листинг 3.2. Дополнительные уровни вложенности в родительские элементы, необходимые излишне абстрактному элементу

```
<?xml version = "1.0" encoding = "UTF-8"?>
<ExampleTransaction>

  <CorporateHumanResourcesData>
    <Employee>
      <Identifier>12345</Identifier>
    </Employee>
  </CorporateHumanResourcesData>
```

```
    <PayrollData>
      <Employee>
        <Identifier>12345</Identifier>
      </Employee>
    </PayrollData>

    <VehicleParkingTagData>
      <Employee>
        <Identifier>12345</Identifier>
      </Employee>
    </VehicleParkingTagData>

</ExampleTransaction>
```

Сила стандарта именования заключается и в определенности, и в абстрактности. Здесь должен достигаться необходимый баланс. Следующий список характеристик может помочь в определении того, в каких случаях имеет смысл рассмотреть вопрос об абстрактном именовании элемента.

- Элемент представляет общеизвестную и легко узнаваемую концепцию данных.
- Элемент – кандидат на широкомасштабное повторное использование.
- Получающееся имя элемента ясно описывает внутреннее содержимое.
- Получающееся имя элемента интуитивно понятно.
- Получающееся имя элемента недвусмысленно.
- Получающееся имя элемента не допускает неверного толкования и представления данных другой формы или значения.

Возможность. *Проектировщик данных обычно хорошо разбирается в стандартах именования и процессах именования и прекрасно знает информационные активы предприятия (их определение и использование). Несмотря на уровень абстракции или определенности и прочие процессы стандарта именования в XML, проектировщики данных имеют реальную возможность принять в них участие и применить все свои навыки.*

Не только элементы с абстрактными именами, но конкретизировано названные элементы можно использовать с большой пользой. Однако элементы с конкретизированными именами не очень подходят для широкомасштабного повторного использования (могут быть веские основания на исключение такого элемента из списка кандидатов на повторное использование). Элемент может быть ограничен определенным контекстом или процессом. Также могут существовать некоторые характеристики данных, требующие уникального имени. Независимо от уровня конкретности или абстрактности применяемого к имени XML-элемента, проектировщик данных, как правило, хорошо разбирается в практическом использовании стандартов именования. Возможность определить и применить описывающий и строгий стандарт именования к элементам и атрибутам XML открывает широкие возможности для предприятия.

Традиционные подходы к именованию элементов данных

Существует довольно много подходов к присвоению имени элементам данных. Большинство из них, несмотря на возраст, имеют очевидные достоинства и приносят пользу всему технологическому сообществу. Однако некоторым из них присущи ограничения и зависимость от конкретных реализаций баз данных того времени, когда они разрабатывались. К примеру, в некоторых базах данных максимально допустимая длина имени элемента или столбца ограничена относительно небольшим значением и к тому же в имени допускается использование только символов верхнего регистра. В результате многие имена элементов данных и столбцов баз данных представляют собой сплошные сокращения и часто выглядят для непосвященных как шифр.

Коль скоро процесс информационного проектирования становится преобладающим методом для идентификации, описания и классификации информационных активов предприятия, прочие формы именования элементов данных были адаптированы и применены к атрибутам логических моделей данных. Поскольку логическая модель данных подразумевает отсутствие ориентации на какую-либо конкретную реализацию, именование также становится более абстрактным. У проектировщиков данных стало больше свободы в использовании описательных имен для информационных активов предприятия. В процессе нисходящего проектирования эти атрибуты позднее будут транслированы для их адаптации к более низкому уровню ("физическому"), и в этот момент ограничения, диктуемые конкретной базой данных, опять вступят в игру. В попытке применить к имени строгий стандарт именования и структурирование родились несколько техник именования.

Одни из наиболее известных базируются на компоненте Data Entity Naming Convention (Соглашение об именовании сущностей данных) стандарта Information Resource Dictionary System[4] (IRDS – системы словаря информационных ресурсов). Техники, полученные в результате адаптации этого стандарта, широко используются в информационной индустрии, они внедряют высоко структурированный подход в именование элементов данных. В числе очевидных выгод – то, что все имена элементов данных следуют согласованной структуре и форме, и то, что эти имена ограничивающим образом описывают допустимые значения элементов данных (определяются по слову класса или домена). Наиболее общее применение этой техники приводит к конструированию имени элемента данных из следующих составных частей:

[4] American National Standarts Institute (ANSI), X3 Working Group. X3.138 Information Resource Dictionary System. Data Entity Naming Conventions, Special Publication 500-149; Manual for Data Administration, Special Publication 500-208 [InterNational Committee for Information Technology Standarts (INCITS)]. Находится по адресу http://www.x3.org/incits/. – *Примеч. авт.*

- главное слово;
- модификатор (или модификаторы);
- слово класса.

Главное слово, чаще всего имя существительное, может определять роль или классификацию. Имя элемента данных "PAYROLL EMPLOYEE IDENTIFIER NUMBER" (идентификационный номер сотрудника в платежной ведомости) – это, возможно, экстремальный пример традиционного именования элементов данных (некоторые могут возразить, что на их предприятии такая техника именования не используется, однако позвольте оставить этот пример для обсуждения). В этом примере "PAYROLL" – главное слово. За главным словом могут следовать один или несколько модификаторов. В данном случае это "EMPLOYEE" (сотрудник) и "IDENTIFIER" (идентификатор). Двусмысленность часто проявляется именно в области модификаторов, поскольку они могут быть прилагательными (что предвещает и появление модификаторов-существительных), а также они могут быть другими существительными, глаголами или наречиями. При использовании дополнительных модификаторов имя становится более конкретизированным. В некоторых случаях эта конкретизированность необходима. Но она может ограничить возможность повторного использования элемента данных в других контекстах. Если некоторый элемент данных, представляющий общий идентификатор сотрудника предприятия, мог использоваться в контексте платежной системы, специализация имени этого элемента ограничит возможности его использования в других контекстах (например, VEHICLE_PARKING_TAG_EMPLOYEE_IDENTIFIER_NUMBER – идентификатор сотрудника на стоянке автомобилей).

В рассматриваемом примере "NUMBER" является словом класса, дополняющим имя элемента данных. Проблема в том, что в некоторых сценариях слово класса может сбивать с толку. При рассмотрении элемента данных, действующего как некая форма идентификатора, четверо из пяти проектировщиков данных назовут или опишут его как "number". Однако в действительности, часто такие идентификаторы состоят из букв или букв и цифр. В этом случае (и тому есть множество примеров) значение слова класса ограничивает использование элемента. Рассмотрим общеизвестные примеры, например номер водительских прав, номер лицензии на плате, номер счета покупателя и т. п. Физический тип данных в каждом из этих примеров сильно отличается от остальных, тогда как слово класса повсюду подразумевает числовое значение.

Давно практикующие проектировщики данных знакомы с этой проблемой в именовании, особенно характерной для слов класса. Справедливости ради отметим множество примеров, когда слово класса очень уместно и помогает специалисту, читающему имя элемента данных. Проблема в том, чтобы отличить, когда использование слов класса полезно, а когда нет. Строгие приложения слов класса, похожие на подходы к именованию IRDS, могут устанавливать де-факто правило обязательного употребления слов класса, невзирая на уместность и краткость.

Как уже отмечалось, другой проблемой, связанной с традиционными практиками стандартов именования, является то, что имена физических элементов и столбцов базы данных также являются производными или результатом трансляции от подобных форм именования. Снова рассмотрим сценарий идентифицирующего номера сотрудника в платежной ведомости для новой системы учета личного состава. Даже с сокращениями, в соответствии с ограничениями базы данных на длину имени, появится следующее имя физического элемента данных, достаточно описательное: "PAY-EMP-IDENT-NO".

В этом примере имя физического элемента данных все также базируется на стандарте именования и имеет главное слово, модификатор, и слово класса. Оно состоит из четырех частей (PAY, EMP, IDENT и NO). Можно догадаться, что первая часть – сокращение от "PAYROLL"(платежная ведомость), вторая – сокращение от "EMPLOYEE" (сотрудник), третья – от "IDENTIFIER" (идентификатор), а последняя часть – слово класса – сокращение от "NUMBER" (номер). Хотя замысел в том, что значение, хранящееся в этом элементе, представляет "PAYROLL EMPLOYEE IDENTIFIER NUMBER" (идентифицирующий номер сотрудника в платежной ведомости), возникает ряд закономерных вопросов.

- Действительно ли идентифицирующий номер – число?
- Является ли идентифицирующий номер уникальным (неповторяющимся)?
- Идентифицирующий номер генерируется как произвольное число или этот номер имеет какую-то внутреннюю смысловую нагрузку?
- Существует ли ограничение его наибольшего и наименьшего значения или длины?
- Значение идентификатора служащего является общим по всему предприятию или используется в этом качестве только в платежной системе?
- Идентификатор служащего базируется или является производным от других форм идентификаторов (например, номера социального страхования или налогового идентификатора)?
- Возможно ли, что контекст разрегулирован (например, идентификатор служащего в платежной ведомости в действительности – другая форма идентификации, существующей вне предприятия, но захваченной в систему учета личного состава в качестве формы проверки гражданства).

Суть в том, что приведенный пример имени элемента данных – предмет для потенциального неверного толкования. В зависимости от ответа на приведенные вопросы, соответствующие бизнес-правила и логика приложения могут меняться. Неверные предположения, построенные на имени, могут привести к ошибкам в обработке и потенциально – к переделке прикладной программы. Также данный пример нарушает правило интуитивности. Непосвященный пользователь может разобраться, а может и не разобраться в том, что это изобилующее сокращениями имя физического элемента данных имеет отношение к идентификатору сотруд-

ника в платежной ведомости. За этой заботой об именовании стоят перспективы проектирования. Приведенное в качестве примера имя слишком конкретизировано, чтобы поддерживать повторное использование в других контекстах. Альтернативой проектирования может выступать определение стандартного элемента "Employee Identifier" (идентификатор сотрудника), используемого повсюду на предприятии. Предполагается, что определение и характеристики метаданных этого идентификатора будут общими, независимо от используемого контекста.

Хотя традиционные практики именования можно использовать и в отношении имен и атрибутов XML в неизменном виде, проектировщик данных должен тщательно адаптировать эти практики с целью расширения их возможностей с учетом мощи XML.

Альтернативные стандарты именования для XML

Может существовать много альтернативных методов именования для XML. К сожалению, многочисленные публикации по XML и словари не представляют руководства или понимания общего подхода. Благодаря своей гибкости, самоописывающим возможностям и отсутствию ограничения на длину имени, именование элементов и атрибутов представляет вызов для промышленных стандартов именования. Разрешение разных имен для одного и того же элемента данных может внести такую же форму несоответствия данных, которая может уже существовать на предприятии.

Рассмотрим тот факт, что XML можно использовать для описания содержания транзакции для обмена, совместного использования или перемещения данных между разными системами. Очевидно, наилучшим решением было бы иметь имя элемента, которое было бы общим для взаимодействующих систем. Однако для автономных систем предприятия, имеющих свои собственные определения данных, такая ситуация редка. Другой полезный подход – выработать для обмена общий словарь, включающий стандартные имена и определения элементов данных. Если передающая или принимающая системы не опознает эти имена, необходимо выполнить соответствующее преобразование.

Применение строгих стандартов именования XML предполагает также ограничение количества преобразований и продвижение использования общего словаря для всех передающих и принимающих систем. Это – благородная цель, и применение эффективного подхода к именованию элементов и атрибутов XML может обеспечить необходимую основу. Однако держите за правило всегда предполагать, что обмен транзакциями между автономными системами потребует, по крайней мере, одного преобразования. Целью должно стать сокращение числа преобразований, и эта цель может быть достигнута, если все взаимодействующие системы начнут применять строгий стандарт именования и будут использовать общий словарь обмена. Прочие применения XML, такие, как отображение или визуализация содержимого для пользователя, обеспечивают еще большую мотива-

цию для использования описательных имен. Проектирование таблицы стилей намного эффективнее, если исходные данные имеют интуитивные имена.

Эффективный подход к стандарту именования XML сочетает аспекты, описанные до этого, с простым и понятным методом сборки составных частей имени. Вместо использования высоко структурированных и следующих в определенном порядке главного слова, модификатора и слова класса, предлагается формировать имена в соответствии с естественным языком (для английского – слева направо). Кроме того, следует избегать аббревиатур, а слово класса можно использовать тогда и только тогда, когда оно краткое, меткое и добавляет смысл имени. Конкретизация имени до степени полной невозможности повторного использования приемлемо только тогда, когда использование соответствующего элемента намеренно ограничено определенным контекстом. И наоборот, имена элементов должны иметь ту степень абстракции, чтобы быть точными, краткими, недвусмысленными и интуитивно понятными и при этом все еще допускать возможность повторного использования (рис. 3.3).

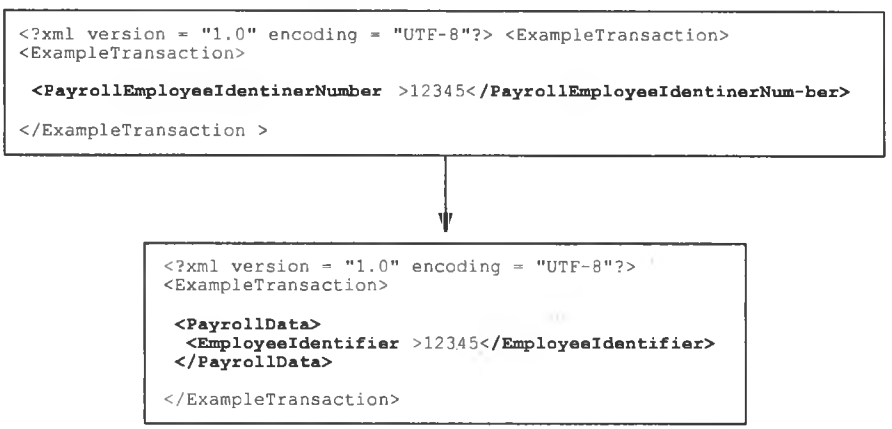


Рис. 3.3. Пересмотр традиционного имени элемента с учетом мощных возможностей XML

Следует отметить, что проектировщики данных не обязаны просто отбрасывать свои традиционные стандарты именования. Во многих ситуациях эти подходы могут быть адаптированы или усилены за счет мощи XML. Также не стоит полностью отказываться от слова класса. Многие слова класса могут быть использованы очень уместно и с пользой. Перед проектировщиками данных стоит задача определять и применять полезные практики и подходы к именованию и избегать остальных. Выработка строгих практик именования – фундаментальная задача проектировщика данных. Как мы увидим в главе 4, другой важной их функцией является согласование поддерживаемых типов данных.

Глава 4

Сравнение типов W3C XML-схемы с типами баз данных

Как уже говорилось, W3C XML-Схемы (W3C Recommendation, May 2001) обеспечивают четкий и сильно структурированный метод для описания строго типизированных данных. Синтаксически это описание совершается посредством применения типов данных и фасет типов данных. W3C XML-Схемы также предоставляют возможность определения собственных (нестандартных типов данных), производных от предопределенных типов. К примеру, вместо того чтобы определять многочисленные элементы данных для разных форматов денежных сумм, проектировщик данных может определить набор стандартных денежных типов данных, которые затем, по необходимости, можно применять к соответствующим элементам и атрибутам XML. По мере роста использования XML-транзакций для обмена, совместного использования и перемещения данных между системами предприятия такой подход также способствует внедрению и распространению стандартов данных.

Применение типа данных схемы, по сути, – форма редакторской проверки. Когда XML-документ ссылается на соответствующую W3C XML-Схему и эта схема содержит описания типов данных, применяемых в определениях элементов и атрибутов, синтаксический анализатор в процессе верификации документа сверяет значения данных, содержащиеся в этих элементах и атрибутах, с описаниями типов данных из схемы. Если значения данных не согласуются с ограничениями, диктуемыми типами данных, определенными в W3C XML-Схеме, в обрабатывающем документ приложении возбуждается ошибка. Важно отметить, что сам синтаксический анализатор не изменяет данные в XML-документе. Принятие решения о возможных действиях возлагается на обрабатывающее приложение. К примеру, при нарушении типа данных, обрабатывающее приложение может прервать только обработку содержимого ошибочного элемента, а может прервать весь процесс обработки целиком.

Следует отметить, что у разных баз данных и связанных с ними технологий поддержка типов данных различается. Но существует несколько базовых типов данных, определения которых имеют схожие характеристики в разных базах данных. К отличиям в определениях типов данных могут относиться: максимальное и минимальное возможные значения, наибольшая длина, поддержка дробных чисел, внутренняя кодировка и внешний формат (например, форматы представления даты и времени). Поначалу может показаться, что типы данных "integer" или "date", независимо от платформы базы данных, описывают одни и те же допустимые значения, количество цифр, символьную длину и формат. В действительности, такое случается редко. Многие базы данных поддерживают различающиеся реализации этих, да и других типов. Разнообразие в типах данных напоминает разнообразие диалектов и расширений SQL. Несмотря на то что SQL базируется на хорошо определенных промышленных стандартах и основные его функции являются общими для большинства реляционных платформ баз данных, существует множество синтаксических расширений SQL, делающих каждую его реализацию по существу уникальной (SQL-запрос, созданный для Sybase Transact SQL может не работать в IBM DB2).

Подобно примеру с SQL, типы данных, используемые для описания содержимого столбцов базы данных или элементов данных, могут отличаться от продукта к продукту. По моему мнению, W3C XML-Схемы обеспечивают гораздо более надежное и точное представление типов данных, чем то, что в настоящее время поддерживается большинством платформ баз данных и языков программирования. С одной стороны, поддержка строго типизированных данных W3C XML-Схемами является исключительно мощной. Но с другой стороны, ошибочно предполагать, что использование W3C XML-Схем "волшебным образом" решит проблему несоответствия данных между автономными системами. В действительности в отображении возможностей и характеристик метаданных исходного и целевого хранилища данных первоочередную роль играет проектировщик данных.

Факт. *Несмотря на возможное совпадение названий типов данных в различных продуктах баз данных, типы данных не всегда реализовываются и поддерживаются одинаковым образом.*

Некоторые могут задаться вопросом – так ли необходимо исследовать соотношение типов данных W3C XML-Схем и типов данных баз данных. Реальность такова, что данные, содержащиеся в большинстве XML-транзакций, скорее всего, берутся из одного или двух возможных источников. Содержимое транзакций электронной коммерции, связанных с покупателем (т. е. B2C), – это либо данные, введенные с клавиатуры, либо данные, полученные из Web-сеанса с пользователем (рис. 4.1). Данные для обмена между приложениями предприятия обычно извлекаются из баз данных (рис. 4.2). В любом случае пунктом назначения данных из таких транзакций чаще всего является база данных предприятия.

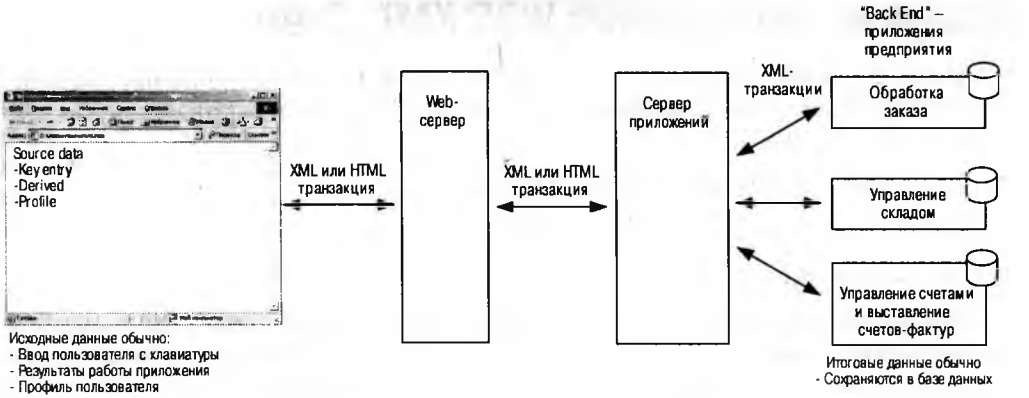


Рис. 4.1. Web-транзакция, ориентированная на покупателя

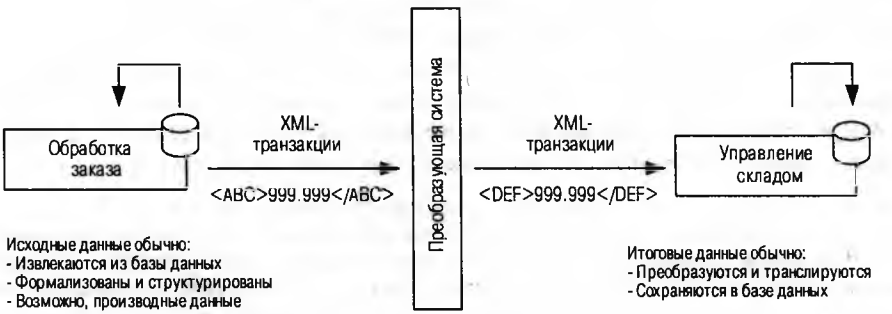


Рис. 4.2. Транзакция для интеграции корпоративных приложений

Факт. *Наиболее часто данные, переносимые посредством транзакций или интерфейсных файлов, в той или иной форме, берутся из базы данных и, так или иначе, сохраняются в базе данных.*

Из рассмотренных примеров видно, что отображение и согласование типов данных – исключительно важный вид деятельности. Это справедливо независимо от того, используется для взаимодействия традиционный "плоский"(flat) файл, файл, использующий разделители-запятые, XML-транзакция или иной формат. Богатые возможности W3C XML-Схем по типизации данных обеспечивают существенное преимущество при обмене или совместном использовании данных автономными системами.

Базовые типы данных W3C XML-Схем

Майские рекомендации по W3C XML-Схемам[1] содержат обширное описание поддержки строго типизированных данных. Поддерживаются три основные категории типов данных:

1. предопределенные примитивные типы;
2. предопределенные производные типы;
3. нестандартные типы.

Предопределенные (встроенные) примитивные типы являются фундаментальными типами данных, на которые можно ссылаться и которые можно применять к элементам и атрибутам XML в их обычной форме (как правило, используя атрибуты-ссылки "type"или "base" и соответствующий "simpleType" W3C XML-Схемы). Большинство из этих типов также поддерживает фасеты в качестве метода введения дополнительных ограничений на допустимое содержание (например, для задания длины, наименьшей допустимой длины, наибольшей допустимой длины и т. п.). Большинство поддерживаемых в W3C XML-Схемах типов данных базируются или согласованы с известными промышленными и международными стандартами. К примеру, типы даты, времени и даты-времени согласуются с "расширенным форматом" ISO 8601 (например, ввгг-мм-дд и ввгг-мм-ддТчч:мм:сс).

> **Факт.** *W3C XML-Схемы обеспечивают обширную поддержку типов данных для даты и времени (например, даты, времени, продолжительности, часового пояса и отдельных частей даты). Однако платформы баз данных могут не обеспечивать поддержку типов данных для даты и времени в соответствии с ISO 8601. Поэтому применение некоторых типов данных даты и времени в W3C XML-Схемах может потребовать использования символьных или числовых типов баз данных для хранения таких данных в базе данных, а также применения соответствующей логики преобразования.*

Нестандартные типы данных могут быть произведены от предопределенных и производных типов путем использования ограничивающих фасет. К примеру, можно определить нестандартный тип данных, представляющий целое число из диапазона от 1000 до 2000. В качестве базового типа будет выступать тип "integer", кроме того, будут применены ограничивающие фасеты для задания минимального и максимального значения. Список предопределенных примитивных типов, определяемых в рекомендациях по W3C XML-Схемам весьма обширен. Типы, представляющие интерес для проектировщиков, применительно к содержимому, ориентированному на транзакции или сообщения, перечислены в табл. 4.1.

---

[1] World Wide Web Consortium (W3C). XML Schemas. W3C Recommendation May 2, 2001. Доступен по адресу http:/www.w3.org/TR/2001/REC-xmlschema-1-20010502/(структуры), http:/www.w3.org/TR/2001/REC-xmlschema-2-20010502/(типы данных). – *Примеч. авт.*

Таблица 4.1. Предопределенные примитивные типы W3C XML-Схемы (наиболее часто используемые) (подмножество, взятое из майских 2001 года рекомендаций по W3C XML-Схемам)

Предопределенные примитивные типы W3C XML-схемы	Примеры значений и формат
duration	Пример значения: P1Y2M3DT1H2M3S Формат: P9Y9M9DT9H9M9S Описание: P = литерал (расшифровывается как период ["period"]) Y= литерал (лет) M = литерал (месяцев) D = литерал (дней) T= литерал (расшифровывается как время["time"]) H = литерал (часов) M = литерал (минут) S = литерал (секунд)
dateTime	Пример значения: 2003-03-15T11:37:02.123 Формат: CCYY-MM-DDTHH:MM:SS.sss Описание: CCYY = год (включая век) MM = месяц DD = день T= литерал (расшифровывается как время["time"]) HH= часы MM = минуты SS = секунды sss = дробная часть секунд sss...n (где требуется)
date	Пример значения: 2003-03-15 Формат: CCYY-MM-DD Описание: CCYY = год (включая век) MM = месяц DD = день

Таблица 4.1. Предопределенные примитивные типы W3C XML-Схемы (наиболее часто используемые) (подмножество, взятое из майских 2001 года рекомендаций по W3C XML-Схемам) (Продолжение)

Предопределенные примитивные типы W3C XML-схемы	Примеры значений и формат
time	Пример значения: 11:37:02.123 Форматы: HH:MM:SS.sss HH:MM:SSZ (обозначает часовой пояс по UTC) HH:MM:SS+HH:MM (положительное смещение от UTC) HH:MM:SS-HH:MM (отрицательное смещение от UTC) Описание: HH = часы MM = минуты SS = секунды sss = дробная часть секунд sss...n (где требуется) Z = литерал (обозначает часовой пояс по UTC)
gYearMonth	Пример значения: 2003-03 Формат: CCYY-MM Описание: CCYY=год(включая век) MM=месяц
gYear	Пример значения: 2003 Формат: CCYY= год(включая век)
gMonthDay	Пример значения: 03-15 Формат: MM-DD Описание: MM=месяц DD=день

Таблица 4.1. Предопределенные примитивные типы W3C XML-Схемы (наиболее часто используемые) (подмножество, взятое из майских 2001 года рекомендаций по W3C XML-Схемам) (Продолжение)

Предопределенные примитивные типы W3C XML-схемы	Примеры значений и формат
gMonth	Пример значения: 03 Формат: MM Описание: MM=месяц
gDay	Пример значения: 15 Формат: DD Описание: DD=день
string	Пример значения: AbcdefgABCDEFG123456#$%* Формат: X (любые, допустимые в XML символы, общая длина не ограничена)
boolean	Пример значения: "0" (значение ложь) "1" (значение истина) "false" (значение ложь) "true" (значение истина)
base64Binary	Формат: Данные, закодированные по алгоритму Base 64 (см. RFC 2045)
hexBinary	Пример значения: 3FA8 Формат: X (любое количество шестнадцатеричных цифр) Описание: Каждый символ должен быть из диапазона 0..9 или A..F (шестнадцатеричные цифры)

Таблица 4.1. Предопределенные примитивные типы W3C XML-Схемы (наиболее часто используемые) (подмножество, взятое из майских 2001 года рекомендаций по W3C XML-Схемам) (Продолжение)

Предопределенные примитивные типы W3C XML-схемы	Примеры значений и формат
float	Пример значения: 15E31 Формат: s9Es9 (с плавающей точкой) Описание: s=знак (+,-,абсолютное значение, не объявлен) 9=мантисса (в диапазоне от –2E24 до 2E24) E=литерал (для обозначения степени [exponent]) 9=степень (в диапазоне от –149 до 104)
decimal	Пример значения: 12345678.12345678 Формат: s9.9 (любое количество цифр) Описание: s=знак (+,-,абсолютное значение, не объявлен) В рекомендациях по W3C XML-Схемам установлена необходимость поддержки не менее 18 цифр
double	Пример значения: 100E231 Формат: s9Es9 (с плавающей точкой, двойной точности) Описание: s= знак (+,-,абсолютное значение, не объявлен) 9=мантисса(в диапазоне от –2E53 до 2E53) E=литерал (для обозначения степени [exponent]) 9=степень (в диапазоне от –1075 до 970)
anyURI	Пример значения: www.abcdefgXXX123.com Формат: строка URI (универсального идентификатора ресурсов) (см. RFC 2396, RFC 2732)

Факт. *По умолчанию, десятичные (decimal) типы данных и некоторые из целочисленных типов W3C XML-Схем не имеют ограничений в виде минимального и максимального значений (они "неограниченные"). Однако в большинстве баз данных десятичные и целочисленные типы имеют определенные верхние и нижние пределы. При обмене числовыми данными следует быть осторожными и следить за тем, чтобы не выйти за пределы ограничений базы данных.*

В дополнение к предопределенным примитивным типам данных, W3C XML-Схемы также поддерживают производные типы. Наиболее часто производные типы являются вариациями двух предопределенных примитивных типов ("string" и "decimal"). Причем, если от "string" напрямую произведено несколько строковых типов, то от "decimal" напрямую произведен только один тип ("integer"). И уже от "integer" произведен ряд других типов, а от этих других – третьи. В табл. 4.2 приведены производные типы, представляющие интерес для описания элементов данных транзакций.

Таблица 4.2. Предопределенные производные типы W3C XML-Схемы (наиболее часто используемые) (подмножество, взятое из майских 2001 года рекомендаций по W3C XML-Схемам)

Предопределенный производный тип W3C XML-Схемы		Примеры значений и формат
Произведен от	**Производный тип**	
string	normalizedString	Пример значения: AbcdefgABCDEFG123456#$%* Формат: X (любой символ, допустимый в XML, кроме символов возврата каретки, перехода на следующую строку и табуляции; длина не ограничена)
string	token	Пример значения: AbcdefgABCDEFG123456#$%* Формат: X (любой символ, допустимый в XML, кроме символов возврата каретки, перехода на следующую строку, табуляции а также пробельных символов (white space) в начале и конце; длина не ограничена)

Таблица 4.2. Предопределенные производные типы W3C XML-Схемы (наиболее часто используемые) (подмножество, взятое из майских 2001 года рекомендаций по W3C XML-Схемам) (Продолжение)

Предопределенный производный тип W3C XML-Схемы		Примеры значений и формат
Произведен от	**Производный тип**	
decimal	integer	Пример значения: 123456789 Формат: s9(n) Границы значений: min значение: -бесконечность max значение: +бесконечность Описание: s= знак (+,-,абсолютное значение, не объявлен) n=неограниченная длина
integer	nonPositiveInteger	Пример значения: -12345 Формат: s9(n) Границы значений: min значение: -бесконечность max значение: 0
integer	long	Пример значения: 123456789 Формат: s9(n) Границы значений: min значение: -9223372036854775808 max значение:+9223372036854775807 Описание: s= знак (+,-,абсолютное значение, не объявлен) n=ограниченная длина – диктуется границами значений

Таблица 4.2. Предопределенные производные типы W3C XML-Схемы (наиболее часто используемые) (подмножество, взятое из майских 2001 года рекомендаций по W3C XML-Схемам) (Продолжение)

Предопределенный производный тип W3C XML-Схемы		**Примеры значений и формат**
Произведен от	**Производный тип**	
integer	nonNegativeInteger	Пример значения: 12345 Формат: s9(n) Границы значений: min значение: 0 max значение:+бесконечность Описание: s=знак (+,-,абсолютное значение, не объявлен) n= ограниченная длина – диктуется границами значений
nonPositiveInteger	negativeInteger	Пример значения: -12345 Формат: s9(n) Границы значений: min значение: -бесконечность max значение:-1 Описание: s= знак (+[a],-,абсолютное значение, не объявлен) n= ограниченная длина – диктуется границами значений
long	int	Пример значения: 123456789 Формат: s9(n) Границы значений: min значение: -2147483648 max значение:+2147483647 Описание: s= знак (+,-,абсолютное значение, не объявлен) n= ограниченная длина – диктуется границами значений

Таблица 4.2. Предопределенные производные типы W3C XML-Схемы (наиболее часто используемые) (подмножество, взятое из майских 2001 года рекомендаций по W3C XML-Схемам) (Продолжение)

Предопределенный производный тип W3C XML-Схемы		Примеры значений и формат
Произведен от	**Производный тип**	
long	short	Пример значения: 12345 Формат: s9(n) Границы значений: min значение: -32768 max значение:+32767 Описание: s= знак (+,-,абсолютное значение, не объявлен) n= ограниченная длина – диктуется границами значений
long	byte	Пример значения: 127 Формат: s9(n) Границы значений: min значение: -128 max значение:+127 Описание: s= знак (+,-,абсолютное значение, не объявлен) n= ограниченная длина – диктуется границами значений
nonNegativeInteger	positiveInteger	Пример значения: 12345 Формат: s9(n) Границы значений: min значение: +1 max значение:+бесконечность Описание: s= знак (+,-,абсолютное значение, не объявлен) n= ограниченная длина – диктуется границами значений

Таблица 4.2. Предопределенные производные типы W3C XML-Схемы (наиболее часто используемые) (подмножество, взятое из майских 2001 года рекомендаций по W3C XML-Схемам) (Продолжение)

Предопределенный производный тип W3C XML-Схемы		Примеры значений и формат
Произведен от	**Производный тип**	
nonNegativeInteger	unsignedLong	Пример значения: 12345 Формат: 9(n) Границы значений: min значение:0 max значение:18446744073709551615 Описание: n= ограниченная длина – диктуется границами значений
nonNegativeInteger	unsignedInt	Пример значения: 12345 Формат: 9(n) Границы значений: min значение: 0 max значение:4294967295 Описание: n= ограниченная длина – диктуется границами значений
nonNegativeInteger	unsignedShort	Пример значения: 12345 Формат: 9(n) Границы значений: min значение: 0 max значение:65535 Описание: n= ограниченная длина – диктуется границами значений

Таблица 4.2. Предопределенные производные типы W3C XML-Схемы (наиболее часто используемые) (подмножество, взятое из майских 2001 года рекомендаций по W3C XML-Схемам) (Продолжение)

Предопределенный производный тип W3C XML-Схемы		Примеры значений и формат
Произведен от	**Производный тип**	
nonNegativeInteger	unsignedByte	Пример значения: 127 Формат: 9(n) Границы значений: . min значение: 0 max значение:255 Описание: n= ограниченная длина – диктуется границами значений

a. Етественно, здесь не может быть знак "+". Простим автору его маленькую неточность. – *Примеч. ред.*

b. Аналогично, здесь не может быть знак "-". – *Примеч. ред.*

Как отмечалось, W3C XML-Схемы поддерживают большое число предопределенных типов данных (обладающих внутренней поддержкой), а также дополнительные производные типы данных. W3C XML-Схемы также позволяют создавать собственные, нестандартные типы данных путем применения ограничивающих фасет. Поддержка нестандартных типов данных – очень полезная возможность при работе с бизнес-приложениями, требующими разнообразных форматов, шаблонов, длин и форм представления десятичных чисел. Подробнее использование ограничивающих фасет рассматривается в главе 5.

Согласование с типами данных СУБД

Многие организации приобрели и используют разные технологии баз данных. И хотя XML-расширения доступны во многих продуктах реляционных баз данных, XML сам по себе не является базой данных. Но XML часто используется для описания фрагментов данных, извлекаемых из базы данных и помещаемых в базу данных прикладными системами. Перед проектировщиками данных часто встает проблема обмена данными между разными системами, поддерживающими базы данных с различающимися типами данных. Тип данных, поддерживаемый одной базой данных,

может отличаться от аналогичного типа, определенного в другой базе данных. Часто решение по преодолению различий в поддержке типов данных включает процесс отображения типов данных исходной базы данных на типы данных базы данных назначения и преобразование данных на основе этого отображения.

В табл. 4.3 – 4.7 описаны наиболее общие типы данных, поддерживаемые рядом продуктов баз данных, и их отображение на типы данных, поддерживаемые часто используемыми W3C XML-Схемами. В ряде случаев тип данных W3C XML-Схемы может не иметь прямого соответствия с аналогичным типом базы данных. В других случаях, тип данных W3C XML-Схемы может поддерживаться базой данных, но наибольшее и наименьшее значения и формат по умолчанию могут быть другими.

При использовании этих таблиц для сравнения типов данных, помните, что внутренние форматы базы данных могут отличаться от внешних форматов извлеченных данных (например это касается даты, времени и временных отметок (timestamp)). Некоторые примеры числовых значений являются значениями со знаком – для описания допустимого диапазона значений (например, только неотрицательные или только отрицательные числа). Также, хотя некоторые из числовых примеров представляют собой данные со знаком, ряд числовых типов данных не разрешает употребление символа знака ("+" или "–") или признака знака в качестве части хранимого значения данных (наподобие "абсолютного" значения). Чтобы определить, требуется или разрешается ли знак для числовых значений, обратитесь к программной документации по соответствующей базе данных или по SQL.

В тех случаях, когда типы данных W3C XML-Схем не имеют прямой поддержки, может потребоваться некоторая форма преобразования. Вид и сложность логики преобразования зависит от степени различий между типами данных. Будем надеяться, что перечисленные ниже продукты баз данных продолжат свое развитие в направлении поддержки XML и типов данных.

Типы данных W3C XML-Схем и типы данных IBM DB2 UDB7

Универсальная база данных DB2 (Universal Database – UDB) (версии 7 и 7.2) является продуктом фирмы IBM, основанным на реляционной модели. Продукт DB2 имеет в сфере информационных технологий долгую историю и широко распространен. DB2 UDB обеспечивает широкую поддержку прикладных данных предприятия, распределенных прикладных данных, и данных транзакций электронной коммерции. Также обеспечена поддержка XML. Однако могут потребоваться расширения или дополнительные компоненты баз данных. Линейка продуктов DB2 UDB доступна на нескольких разных платформах и для разных операционных систем.

Хотя DB2 UDB поддерживает многие базовые типы данных, если сравнить эти типы с типами W3C XML-Схем, увидим несколько отличий и ограничений. Заметны различия в поддержке типов данных в разных версиях продукта и для разных операционных систем. Информацию о специфике поддержки типов данных и XML в разных версиях можно получить у фирмы IBM[2]. В табл. 4.3 сравниваются наиболее часто используемые типы данных W3C XML-Схемы с типами данных, поддерживаемыми в DB2 UDB.

Таблица 4.3. Сравнение наиболее часто используемых типов данных W3C XML-Схемы с типами данных IBM DB2

Наиболее часто используемые типы данных W3C XML-Схемы	IBM DB2 UDB версии 7 и 7.2
duration	Тип "duration" не имеет прямой поддержки. Можно работать со значениями этого типа (форматировать, преобразовывать и извлекать) используя строковый тип, например "char". Но этот подход требует применения в приложении нестандартной логики
dateTime	Тип " dateTime " не имеет прямой поддержки. Можно попытаться использовать тип "TIMESTAMP", предоставляющий схожие возможности. Однако и внутреннее представление, и формат данных "TIMESTAMP" отличается от типов данных "dateTime". Одно из очевидных отличий – тип "TIMESTAMP" не включает литерал T для отделения даты от времени
date	Тип "date" в DB2 поддерживается типом "DATE". Однако внутреннее представление данных типа "DATE" может отличаться от типа "date" W3C XML-Схемы. Также DB2 поддерживает параметр установки, позволяющий варьировать формат
time	Тип "time" в DB2 поддерживается типом "TIME". Однако внутреннее представление данных типа "TIME" может отличаться от "time" W3C XML-Схемы. В частности, формат "time" имеет вид: "HH:MM:SS", тогда как формат времени в DB2 имеет вид: "HH.MM.SS". В некоторых случаях отображаемый формат для времени может быть настроен пользователем или реализацией. Кроме того, представление времени в DB2 не включает обозначения часового пояса и не поддерживает долей секунды (Доли секунды поддерживаются в DB2 типом "TIMESTAMP")

[2] IBM DB2 Universal Database, SQL Reference, Version 7.1, SC09-2974-01. IBM, 1993, 2001. DB2 Universal Database for OS/390, z/OS, SQL Reference, Version 7, SC26-9944-01. IBM, 1982, 2001. Доступен по адресу http://www.ibm.com/. – *Примеч. авт.*

Таблица 4.3. Сравнение наиболее часто используемых типов данных W3C XML-Схемы с типами данных IBM DB2 (Продолжение)

Наиболее часто используемые типы данных W3C XML-Схемы	**IBM DB2 UDB версии 7 и 7.2**
gYearMonth gYear gMonthDay gMonth gDay	Все эти типы W3C XML-Схемы не имеют прямой поддержки в DB2. Их можно представить с помощью строкового типа, например, "CHAR". Но этот подход требует применения в приложении нестандартной логики
string	Тип "string" поддерживается в DB2 UDB одним из нескольких строковых типов. Однако тип "string" не имеет ограничения в виде фиксированной длины или наибольшей длины. Символьные типы DB2 UDB имеет ограничение на максимально возможную длину и обеспечивает разную поддержку в зависимости от типа символов (например, двухбайтовые символы). Также следует принимать во внимание внутреннее представление строки в базе данных. Для хранения могут использоваться разные кодировки (Unicode, UTF-8, ASCII или EBCDIC). Тип DB2 UDB — Ориентировочное наибольшее количество символов CHAR — 254 VARCHAR — 32672 LONGVARCHAR — 32700 CLOB — 2147483647 NCLOB — 4294967295 LONG — 2147483647
boolean	Тип "boolean" не имеет прямой поддержки. Для хранения сходных логических значений ("0", "1", "false", "true") в DB2 можно использовать типы данных "SMALLINT" или "CHAR". Однако использование этих типов потребует введения нестандартной логики в приложение
base64Binary	Тип "base64Binary" не имеет прямой поддержки. Однако, в зависимости от типа, приложения и кодировки, можно использовать другие, графические и двухбайтовые, типы DB2. Но тип base64Binary обычно используется для описания MIME-типов данных: графики, рисунков, иллюстраций и т. п., которые могут поддерживаться или не поддерживаться в DB2. Использование типов DB2 может потребовать введения нестандартной логики в приложение. Тип DB2 UDB — Ориентировочное наибольшее количество символов GRAPHIC — 127 VARGRAPHIC — 16336 LONGVARGRAPHIC — 16350 BLOB — 2147483647

Таблица 4.3. Сравнение наиболее часто используемых типов данных W3C XML-Схемы с типами данных IBM DB2 (Продолжение)

Наиболее часто используемые типы данных W3C XML-Схемы	IBM DB2 UDB версии 7 и 7.2
hexBinary	Тип " hexBinary " не имеет прямой поддержки. Его можно представить с помощью строкового типа, например, "CHAR". Однако такой подход может потребовать введения нестандартной логики в приложение
float	Тип "float" в DB2 поддерживается, до некоторой степени, типами REAL, FLOAT и DOUBLE. Однако диапазоны допустимых значений (и мантиссы и экспоненты) отличаются от аналогичных значений типа "float" из W3C XML-Схемы
decimal	Этот тип в DB2 напрямую поддерживается типами DECIMAL и NUMERIC. Однако диапазоны допустимых значений (и общее количество цифр и длина дробной части) отличаются от аналогичных значений типа "decimal" из W3C XML-Схемы
integer long	Типы "integer"и "long" напрямую не поддерживаются. Однако в некоторых случаях можно использовать типы BIGINT, INTEGER и SMALL INT. Важно отметить, что для типа "integer" W3C XML-Схемы не существует ограничений в виде max и min значений (его значения бесконечны). Тип "long" соответствует типу BIGINT из DB2 и по допустимым значениям. Все типы DB2 имеют верхнюю и нижнюю границу. Тип DB2 UDB Ориентировочные пределы BIGINT(min) -9223372036854775808 BIGINT(max) +9223372036854775807 INTEGER(min) -2147483648 INTEGER(max) +2147483647 SMALLINT(min) -32768 SMALLINT(max) +32767
int	Тип "int" в DB2 напрямую поддерживается типом INTEGER.
short	Тип " short " в DB2 напрямую поддерживается типом SMALLINT.

Таблица 4.3. Сравнение наиболее часто используемых типов данных W3C XML-Схемы с типами данных IBM DB2 (Продолжение)

Наиболее часто используемые типы данных W3C XML-Схемы	IBM DB2 UDB версии 7 и 7.2
byte	Тип "byte" не имеет прямой поддержки. В некоторых случаях для его поддержки можно использовать типы BIGINT, INTEGER или SMALLINT. При этом следует помнить, что значения типа "byte" W3C XML-Схемы находятся в границах от –128 до +127. Перечисленные типы DB2 имеют иные границы значений. Тип DB2 UDB Ориентировочные пределы BIGINT(min) -9223372036854775808 BIGINT(max) +9223372036854775807 INTEGER(min) -2147483648 INTEGER(max) +2147483647 SMALLINT(min) -32768 SMALLINT(max) +32767
double	Тип "double" до некоторой степени поддерживается в DB2 типами REAL, FLOAT и DOUBLE. Однако их наибольшая и наименьшая границы (и мантиссы и экспоненты) иные, нежели у типа "double"
anyURI	Тип anyURI не имеет прямой поддержки в DB2. Однако в некоторых случаях его могут заменить другие типы DB2. Однако для этого потребуется дополнительная логика приложения. Возможные кандидаты на замену: DATALINK (внешний файл или ссылка на объект); CHAR (обычная строка)

Типы данных W3C XML-Схем и типы данных Oracle 9*i*

Oracle (версия 9*i*) – реляционная база данных от корпорации Oracle. Как и DB2, базы данных Oracle основываются на реляционной модели и широко распространены. Их часто используют для поддержки прикладных данных предприятия, распределенных прикладных данных, и сохранения транзакций электронной коммерции. Oracle 8*i* и 9*i* также обеспечивают поддержку XML. Но все равно могут потребоваться расширения или дополнительные компоненты базы данных.

Линейка продуктов Oracle доступна на нескольких аппаратных платформах и для разных операционных систем. Базовые типы данных поддерживаются продуктами Oracle 8*i* и 9*i*. Однако могут быть различия в характеристиках и поддержке типов в базах данных Oracle и в W3C XML-Схемах. Для некоторых данных и типов

данных XML потребуются функции преобразования. Следует отметить, что в разных версиях Oracle и для разных операционных систем существуют некоторые отличия в поддержке типов данных. Дополнительную информацию, касающуюся типов данных и поддержки XML, можно получить у корпорации Oracle[3].

В табл. 4.4 сравниваются наиболее часто используемые типы данных W3C XML-Схемы с типами данных, поддерживаемыми в продуктах Oracle.

Таблица 4.4. Сравнение наиболее часто используемых типов данных W3C XML-Схемы с типами данных Oracle 9*i*

Наиболее часто используемые типы данных W3C XML-Схемы	Oracle 9*i*
duration	Тип "duration" не имеет прямой поддержки. Можно работать со значениями этого типа с помощью типов INTERVAL YEAR TO MONTH и INTERVAL DAY TO SECOND. Использование этих типов может зависеть от параметров установки Oracle, формата и допустимых значений. В приложении может потребоваться преобразование и дополнительная логика
dateTime	Тип " dateTime " не имеет прямой поддержки. Можно попытаться использовать типы TIMESTAMP, TIMESTAMP WITH TIMEZONE и TIMESTAMP WITH LOCAL TIMEZONE. Использование этих типов может зависеть от параметров установки Oracle, формата и допустимых значений. В приложении может потребоваться преобразование и дополнительная логика. Также стандартный формат даты в Oracle имеет вид: "DD-mmm-YYYY"
date	Тип "date" в Oracle косвенно поддерживается типом "DATE". Использование этого типа может зависеть от параметров установки Oracle, формата и допустимых значений. В приложении может потребоваться преобразование и дополнительная логика. Ряд дополнительных возможностей предоставляет функция "TO_DATE". Также стандартный формат даты в Oracle имеет вид: "DD-mmm-YYYY"
time	Тип "time" может косвенно поддерживаться типами TIMESTAMP, TIMESTAMP WITH TIMEZONE и TIMESTAMP WITH LOCAL TIMEZONE. Использование этих типов может зависеть от параметров установки Oracle, формата и допустимых значений. В приложении может потребоваться преобразование и дополнительная логика. Ряд дополнительных возможностей предоставляют функции "TO_DATE" и "TRUNC". Также, стандартный формат даты в Oracle имеет вид: "DD-mmm-YYYY".

[3] Oracle 9*i* Application Developer Guide, release 1 (9.0.1), part A88876-02. Oracle Corporation, 1996–2001. Доступно по адресу http://www.oracle.com/. – *Примеч. авт.*

Таблица 4.4. Сравнение наиболее часто используемых типов данных W3C XML-Схемы с типами данных Oracle 9*i* (Продолжение)

Наиболее часто используемые типы данных W3C XML-Схемы	**Oracle 9*i***
gYearMonth gYear gMonthDay gMonth gDay	Все эти типы W3C XML-Схемы не имеют прямой поддержки в Oracle. Их можно представить с помощью строкового типа, например, "CHAR". Но этот подход требует применения в приложении нестандартной логики
string	Тип "string" поддерживается в Oracle одним из нескольких строковых типов. Однако тип "string" не имеет ограничения в виде фиксированной длины или наибольшей длины. Символьные типы Oracle 9*i* имеет ограничение на максимально возможную длину и обеспечивает разную поддержку в зависимости от типа символов (например, двухбайтовые символы). Также следует принимать во внимание внутреннее представление строки в базе данных. Для хранения могут использоваться разные кодировки. Тип Oracle 9*i* — Ориентировочное наибольшее количество символов CHAR — 2000 VARCHAR2 — 4000 NCHAR — 2000 NCHAR2 — 4000 CLOB — 4294967295 NCLOB — 4294967295 LONG — 2147483647
boolean	Тип "boolean" не имеет прямой поддержки. Для хранения сходных логических значений ("0", "1", "false", "true") в Oracle можно использовать типы данных NUMBER или CHAR. Однако использование этих типов потребует введения нестандартной логики в приложение

Таблица 4.4. Сравнение наиболее часто используемых типов данных W3C XML-Схемы с типами данных Oracle 9*i* (Продолжение)

Наиболее часто используемые типы данных W3C XML-Схемы	Oracle 9*i*
base64Binary	Тип "base64Binary" не имеет прямой поддержки. Однако, в зависимости от типа, приложения и кодировки, можно использовать другие, графические, типы Oracle. Но тип base64Binary обычно используется для описания MIME-типов данных: графики, рисунков, иллюстраций и т. п., которые могут поддерживаться или не поддерживаться в Oracle. Использование типов Oracle может потребовать введения нестандартной логики в приложение. Тип Oracle 9*i* — Ориентировочное наибольшее количество символов BLOB — 4 Гбайт BFILE — 4 Гбайт (внешняя ссылка) RAW — 2000 LONG ROW — 2 Гбайт
hexBinary	Тип " hexBinary " не имеет прямой поддержки. Его можно представить с помощью строкового типа, например, "CHAR". Однако такой подход может потребовать введения нестандартной логики в приложение
float	Этот тип в Oracle не имеет прямой поддержки. Его значения можно попытаться представить с помощью строкового типа "CHAR" или с помощью "NUMBER". Однако такой подход потребует введения нестандартной логики в приложение
decimal	Этот тип в Oracle напрямую поддерживается типом "NUMBER" Однако диапазоны допустимых значений (и общее количество цифр и длина дробной части) отличаются от аналогичных значений типа "decimal" из W3C XML-Схемы
integer long	Типы "integer"и "long" в некоторых случаях можно заменить типом "NUMBER". Важно отметить, что для типа "integer" W3C XML-Схемы не существует ограничений в виде max и min значений (его значения бесконечны). Тип "NUMBER" имеет верхнюю и нижнюю границы. Тип Oracle 9*i* — Ориентировочные пределы NUMBER(min) — $-1x10^{-130}$ NUMBER(max) — $+9.99x10^{125}$

Таблица 4.4. Сравнение наиболее часто используемых типов данных W3C XML-Схемы с типами данных Oracle 9*i* (Продолжение)

Наиболее часто используемые типы данных W3C XML-Схемы	Oracle 9*i*
int	Тип "int" не имеет прямой поддержки. В некоторых случаях можно использовать тип "NUMBER". Следует отметить, что значения типа "int" ограничены в пределах от -2147483648 до +2147483647. Тип "NUMBER" имеет иные границы. Тип Oracle 9*i* Ориентировочные пределы NUMBER(min) $-1x10^{-130}$ NUMBER(max) $+9.99x10^{125}$
short	Тип " short " не имеет прямой поддержки. В некоторых случаях можно использовать тип "NUMBER". Следует отметить, что значения типа "short" ограничены в пределах от -32768 до +32767. Тип "NUMBER" имеет иные границы. Тип Oracle 9*i* Ориентировочные пределы NUMBER(min) $-1x10^{-130}$ NUMBER(max) $+9.99x10^{125}$
byte	Тип "byte" не имеет прямой поддержки. В некоторых случаях можно использовать тип "NUMBER". Следует отметить, что значения типа "short" ограничены в пределах от -128 до +127. Тип "NUMBER" имеет иные границы. Тип Oracle 9*i* Ориентировочные пределы NUMBER(min) $-1x10^{-130}$ NUMBER(max) $+9.99x10^{125}$
double	Тип "double" не имеет прямой поддержки. Его значения можно попытаться представить с помощью строкового типа "CHAR" или с помощью "NUMBER". Однако такой подход потребует введения нестандартной логики в приложение
anyURI	Тип anyURI не имеет прямой поддержки в Oracle. Однако в некоторых случаях его могут заменить другие типы Oracle, что может потребовать дополнительной логики приложения. Возможные кандидаты на замену: BFILE (внешний файл или ссылка на объект). CHAR (обычная строка)

Типы данных W3C XML-Схем и типы данных Sybase ASE 12.5

Продукт баз данных Sybase Adaptive Server Enterprise (ASE) также предоставляет поддержку XML (могут потребоваться расширения или дополнительные компоненты). Линейка продуктов Sybase ASE также доступна на нескольких аппаратных платформах и для разных операционных систем. Продукт ASE поддерживает несколько базовых типов данных. Однако, по сравнению с типами данных W3C XML-Схемы, также имеется ряд отличий и ограничений. Важно отметить, что в разных версиях баз данных и для разных операционных систем существуют отличия в поддержке типов данных. Фирма Sybase также представляет продукт Adaptive Server Anywhere (ASA), поддерживающий многие из тех типов данных, что поддерживаются в ASE. Однако и здесь существуют отличия в поддержке. Дополнительную информацию можно получить от фирмы Sybase[4]. В табл. 4.5 сравниваются наиболее часто используемые типы данных W3C XML-Схемы с типами данных, поддерживаемыми в продуктах Sybase ASE.

Таблица 4.5. Сравнение наиболее часто используемых типов данных W3C XML-Схемы с типами данных Sybase ASE 12.5

Наиболее часто используемые типы данных W3C XML-Схемы	Sybase ASE 12.5
duration	Тип "duration" не имеет прямой поддержки. Можно работать со значениями этого типа с помощью строкового типа, например, "char". Однако в приложении может потребоваться дополнительная логика
dateTime	Тип " dateTime " можно косвенно поддерживать типами "datetime" и "smalldatetime" . Однако их наибольшее и наименьшее допустимое значения не соответствуют ограничениям типа "dateTime" . Также отсутствует литерал T для разграничения даты и времени
date	Тип "date" можно косвенно поддерживать типами "datetime" и "smalldatetime" . Однако их наибольшее и наименьшее допустимое значения не соответствуют ограничениям типа "date". Формат и допустимые значения могут зависеть от параметров установки (сначала установите формат даты и дату). Также, стандартный формат даты имеет вид: "MMM DD YYYY"

[4] Sybase Adaptive Server Enterprise 12.5. Transact-SQL, Content ID 1009196, revised Feb. 1, 2002. Sybase Inc., 2002. Доступно по адресу http://www.sybase.com/. – *Примеч. авт.*

Таблица 4.5. Сравнение наиболее часто используемых типов данных W3C XML-Схемы с типами данных Sybase ASE 12.5 (Продолжение)

Наиболее часто используемые типы данных W3C XML-Схемы	Sybase ASE 12.5
time	Тип "time" можно косвенно поддерживать типами "datetime" и "smalldatetime". Однако не существует документированной поддержки часового пояса и обозначения часового пояса. Кроме того, поддержка долей секунды в этих двух типах различается
gYearMonth gYear gMonthDay gMonth gDay	Все эти типы W3C XML-Схемы не имеют прямой поддержки в Sybase. Их можно представить с помощью строкового типа, например, "char", "datetime" и "smalldatetime". Но этот подход требует применения в приложении нестандартной логики
string	Тип "string" поддерживается в Sybase одним из нескольких строковых типов. Однако тип "string" W3C XML-Схемы не имеет ограничения в виде фиксированной длины или наибольшей длины. Символьные типы Sybase имеет ограничение на максимально возможную длину и обеспечивает разную поддержку в зависимости от типа символов (например, двухбайтовые символы). Также следует принимать во внимание внутреннее представление строки в базе данных. Для хранения могут использоваться разные кодировки. Тип Sybase ASE 12.5 — Ориентировочная наибольшая длина char — 255 varchar — 255 nchar — 255 nvarchar — 255 text — 2147483647
boolean	Тип "boolean" не имеет прямой поддержки. Для хранения сходных логических значений ("0", "1", "false", "true") в Sybase можно использовать битовые или символьные типы. Однако использование этих типов потребует введения нестандартной логики в приложение

Таблица 4.5. Сравнение наиболее часто используемых типов данных W3C XML-Схемы с типами данных Sybase ASE 12.5 (Продолжение)

Наиболее часто используемые типы данных W3C XML-Схемы	Sybase ASE 12.5
base64Binary	Тип "base64Binary" не имеет прямой поддержки. Однако, в зависимости от типа, приложения и кодировки, можно использовать другие, графические, типы Sybase. Но тип base64Binary обычно используется для описания MIME-типов данных: графики, рисунков, иллюстраций и т. п., которые могут поддерживаться или не поддерживаться в Sybase. Использование типов Sybase может потребовать введения нестандартной логики в приложение. Sybase ASE 12.5 — Примерное наибольшее количество символов binary — 255 varbinary — 255 image — 2147483647
hexBinary	Тип " hexBinary " не имеет прямой поддержки. Его можно представить с помощью строкового типа, например, "char". Однако такой подход может потребовать введения нестандартной логики в приложение
float	Этот тип в Sybase до некоторой степени можно представить с помощью типов "float", "double", и "real". Однако формат и границы значений (и мантиссы и экспоненты) могут отличаться от аналогичных значений типа "float" W3C XML-Схемы
decimal	Этот тип в Sybase напрямую поддерживается типами "decimal", "numeric", и "dec"(синоним). Однако диапазоны допустимых значений (и общее количество цифр и длина дробной части) отличаются от аналогичных значений типа "decimal" из W3C XML-Схемы
integer long	Типы "integer"и "long" косвенно, в некоторых приложениях, поддерживаются типами "decimal", "numeric", "int" и "integer"(синоним). Важно отметить, что для типа "integer" W3C XML-Схемы не существует ограничений в виде max и min значений (его значения бесконечны). Типы Sybase имеют верхнюю и нижнюю границы. Тип Sybase ASE 12.5 — Ориентировочные пределы numeric(min) — -10^{38} numeric(max) — $+10^{38}(-1)$ decimal(min) — -10^{38} decimal(max) — $+10^{38}(-1)$ int(min) — - 2147483648 int(max) — +2147483647

Таблица 4.5. Сравнение наиболее часто используемых типов данных W3C XML-Схемы с типами данных Sybase ASE 12.5 (Продолжение)

Наиболее часто используемые типы данных W3C XML-Схемы	Sybase ASE 12.5
int	Тип "int" поддерживаются напрямую
short	Тип " short " поддерживаются в Sybase напрямую типом "smallint".
byte	Тип "byte" косвенно поддерживается типом "smallint" и, возможно, типом "tinyint". Однако, тип "smallint" имеет более широкие границы значений, чем тип "byte" W3C XML-Схемы. Значения типа "byte" лежат в диапазоне от –128 до +127. Значения же типа "tinyint" ограничены неотрицательными целыми числами (от 0 до 255). Тип Sybase ASE 12.5 Ориентировочные пределы smallint (min) -32768 smallint (max) +32767 tinyint(min) 0 tinyint(max) 255
double	Тип "double" до некоторой степени поддерживается в Sybase типами "float", "double" и "real". Однако формат и наибольшие и наименьшие значения (и мантиссы и экспоненты) могут отличаться от аналогичных величин типа "double" W3C XML-Схемы
anyURI	Тип anyURI не имеет прямой поддержки в Sybase. Однако в некоторых случаях его могут заменить строковые типы Sybase, что может потребовать дополнительной логики приложения. Возможные кандидаты на замену: Тип Sybase 12.5 Ориентировочная наибольшая длинна char 255 varchar 255 nchar 255 nvarchar 255 text 2147483647

Типы данных W3C XML-Схемы и типы данных SQL Server

Завершаем этот раздел популярным продуктом баз данных – Microsoft SQL Server. SQL Server также основывается на реляционной модели и обеспечивает широкую поддержку приложений предприятия и приложений электронной коммерции. Как и другие продукты, SQL Server поддерживает XML (в зависимости от версии). Опять

таки, могут потребоваться расширения или дополнительные компоненты базы данных. SQL Server поддерживает большинство базовых типов данных, хотя и существуют некоторые отличия от типов данных W3C XML-Схемы. В разных версиях и для разных операционных систем присутствуют отличия в типах данных и характеристиках данных. Дополнительную информацию можно получить у фирмы Microsoft[5]. В Табл. 4.6 сравниваются наиболее часто используемые типы данных W3C XML-Схемы с типами данных, поддерживаемыми в SQL Server.

Таблица 4.6. Сравнение наиболее часто используемых типов данных W3C XML-Схемы с типами данных SQL Server

Наиболее часто используемые типы данных W3C XML-Схемы	SQL Server
duration	Тип "duration" не имеет прямой поддержки. Можно работать со значениями этого типа с помощью строкового типа, например, "char". Однако в приложении может потребоваться дополнительная логика
dateTime	Тип " dateTime " можно в SQL Server косвенно поддерживать типами "datetime" и "smalldatetime" . В некоторых приложениях для этого подойдет также тип "timestamp". Однако их наибольшее и наименьшее допустимые значения не соответствуют ограничениям типа " dateTime ". Также отсутствует литерал T для разграничения даты и времени
date	Тип "date" можно косвенно поддерживать в SQL Server типами "datetime" и "smalldatetime". Однако их наибольшее и наименьшее допустимое значения не соответствуют ограничениям типа "date". Формат и допустимые значения могут зависеть от параметров установки (сначала установите формат даты и дату)
time	Тип "time" можно косвенно поддерживаться типами "datetime" и "smalldatetime" . Однако не существует документированной поддержки часового пояса и обозначения часового пояса
gYearMonth gYear gMonthDay gMonth gDay	Все эти типы W3C XML-Схемы не имеют прямой поддержки в SQL Server. Их можно представить с помощью строкового типа, например, "char", "datetime" и "smalldatetime". Но этот подход требует применения в приложении нестандартной логики

[5] Microsoft SQL Server, Transact-SQL Reference (on-line), Data Types, Updated September 2001. Доступно по адресу http://msdn.microsoft.com/library/default.asp?url=/library/en-us/tsqlref/ts_da-db_7msw.asp. – *Примеч. авт.*

Таблица 4.6. Сравнение наиболее часто используемых типов данных W3C XML-Схемы с типами данных SQL Server (Продолжение)

Наиболее часто используемые типы данных W3C XML-Схемы	SQL Server
string	Тип "string" поддерживается в SQL Server одним из нескольких строковых типов. Однако тип "string" W3C XML-Схемы не имеет ограничения в виде фиксированной длины или наибольшей длины. Символьные типы SQL Server имеет ограничение на максимально возможную длину и обеспечивает разную поддержку в зависимости от типа символов (например, двухбайтовые символы). Также следует принимать во внимание внутреннее представление строки в базе данных. Для хранения могут использоваться разные кодировки. Тип SQL Server — Ориентировочная наибольшая длина char — 8000 varchar — 8000 text — 2147483647 nchar (поддержка Unicode) — 4000 nvarchar (поддержка Unicode) — 4000 ntext (поддержка Unicode) — 1073741823
boolean	Тип "boolean" не имеет прямой поддержки. Для хранения сходных логических значений ("0", "1", "false", "true") в SQL Server можно использовать битовые или символьные типы. Однако использование этих типов потребует введения нестандартной логики в приложение
base64Binary	Тип "base64Binary" не имеет прямой поддержки. Однако, в зависимости от типа, приложения и кодировки, можно использовать другие, графические, типы SQL Server. Но тип base64Binary обычно используется для описания MIME-типов данных: графики, рисунков, иллюстраций и т. п., которые могут поддерживаться или не поддерживаться в SQL Server. Использование типов SQL Server может потребовать введения нестандартной логики в приложение. SQL Server — Примерное наибольшее количество символов binary — 8000 varbinary — 8000 image — 2147483647
hexBinary	Тип " hexBinary " не имеет прямой поддержки. Его можно представить с помощью строкового типа, например, "char". Однако такой подход может потребовать введения нестандартной логики в приложение

Таблица 4.6. Сравнение наиболее часто используемых типов данных W3C XML-Схемы с типами данных SQL Server (Продолжение)

Наиболее часто используемые типы данных W3C XML-Схемы	SQL Server
float	Этот тип в SQL Server до некоторой степени можно представить с помощью типов "float" и "real". Однако формат и границы значений (и мантиссы и экспоненты) могут отличаться от аналогичных значений типа "float" W3C XML-Схемы
decimal	Этот тип в SQL Server напрямую поддерживается типами "decimal" и "numeric". Однако диапазоны допустимых значений (и общее количество цифр, и длина дробной части) отличаются от аналогичных значений типа "decimal" из W3C XML-Схемы
integer long	Типы "integer"и "long" косвенно, в некоторых приложениях, поддерживаются типами "decimal", "numeric", "bigint", "int". Важно отметить, что для типа "integer" W3C XML-Схемы не существует ограничений в виде max и min значений (его значения бесконечны). Типы SQL Server имеют верхнюю и нижнюю границы. Тип SQL Server — Ориентировочные пределы numeric(min) — -10^{38} numeric(max) — $+10^{38}(-1)$ decimal(min) — -10^{38} decimal(max) — $+10^{38}(-1)$ bigint(min) — -9,223,372,036,854,775,808 bigint(max) — +9,223,372,036,854,775,807 int(min) — - 2147483648 int(max) — +2147483647
int	Тип "int" поддерживаются напрямую
short	Тип " short " поддерживаются в SQL Server напрямую типом "smallint"
byte	Тип "byte" косвенно поддерживается типом "smallint" и, возможно, типом "tinyint". Однако, тип "smallint" имеет более широкие границы значений, чем тип "byte" W3C XML-Схемы. Значения типа "byte" лежат в диапазоне от –128 до +127. Значения типа "tinyint" ограничены неотрицательными целыми числами (от 0 до 255). Тип SQL Server — Ориентировочные пределы smallint (min) — -32768 smallint (max) — +32767 tinyint(min) — 0 tinyint(max) — 255

Таблица 4.6. Сравнение наиболее часто используемых типов данных W3C XML-Схемы с типами данных SQL Server (Продолжение)

Наиболее часто используемые типы данных W3C XML-Схемы	SQL Server
double	Тип "double" до некоторой степени поддерживается в SQL Server типами "float" и "real". Однако формат и наибольшие и наименьшие значения (и мантиссы, и экспоненты) могут отличаться от аналогичных величин типа "double" W3C XML-Схемы
anyURI	Тип anyURI не имеет прямой поддержки в SQL Server. Однако в некоторых случаях его могут заменить строковые типы SQL Server, что может потребовать дополнительной логики приложения. Возможные кандидаты на замену: Тип SQL Server — Ориентировочная наибольшая длинна char — 8000 varchar — 8000 text — 2147483647 nchar (поддержка Unicode) — 4000 nvarchar (поддержка Unicode) — 4000 ntext (поддержка Unicode) — 1073741823

Синтаксис W3C XML-Схемы для назначения типа данных

Когда происходит обмен данными между платформами, прикладными системами и продуктами баз данных, особое значение приобретает отображение типов данных. Использование языка XML и W3C XML-Схем для описания транзакций обмена или интеграции может обеспечить для этого необходимую поддержку. Как мы помним, экземпляр XML-документа содержит значения данных. Схема описывает характеристики метаданных, правила и ограничения, которые применяются к XML-документу в процессе верификации его синтаксическим анализатором. Одной из проверок, выполняемых в ходе верификации, является проверка соответствия значений данных из XML-документа и объявлений указанных для них типов данных, найденных в схеме.

ных" – один из поддерживаемых W3C XML-Схемой типов данных (т. е. "integer", "date", "dateTime", "decimal", "boolean" и т. д.). Эта основная форма записи может быть расширена с тем, чтобы установить дополнительные ограничения, например на допустимую длину. Простое назначение элементу или атрибуту XML какого-либо предопределенного типа – это только один из возможных подходов. Можно в составе W3C XML-Схемы объявить нестандартный тип данных, ужесточающий предопределенный тип (примитивный или производный) одним или несколькими ограничениями. Эти ограничения (длины, наибольшего или наименьшего значений, списка допустимых значений, по шаблону) задаются с помощью фасет типа данных (листинг 4.1).

Листинг 4.1 Нестандартный тип данных, описанный в W3C XML-Схеме с помощью "simpleType"

```
<xs:simpleType name="STDMonetaryType">
  <xs:restriction base="xs:decimal">
    <xs:totalDigits value="9"/>
    <xs:fractionDigits value="2"/>
  </xs:restriction>
</xs:simpleType>
```

Техника. *Синтаксическая форма назначения типа данных W3C XML-Схемы такова: "type="тип_данных"", здесь "тип_данных" – один из поддерживаемых в W3C XML-Схеме типов, например "integer", "date", "dateTime", "decimal", или "boolean".*

Ответственность за разработку W3C XML-Схем зависит от организации предприятия и организации технологических процессов на предприятии. Как мы увидим в главе 9, определения и характеристики метаданных предприятия обычно находятся в ведении проектировщика данных. Однако синтаксическое применение ограничений метаданных в W3C XML-Схеме может быть возложено на проектировщика данных, разработчика или другой технологический персонал. Несмотря на очерченную зону ответственности, проектировщик данных должен иметь достаточный багаж знаний в отношении синтаксиса и типов данных W3C XML-Схемы.

Ниже приведены примеры простых элементов в XML-документе и фрагменты синтаксиса W3C XML-Схемы, описывающего применение типов данных. Примеры логически сгруппированы по типам данных (дата и время, целочисленные типы, десятичные, логические, URI и строковые).

Основные типы даты и времени W3C XML-Схемы

```
<!-  ============================================    ->
<!- Фрагмент документа XML - элементы для даты и времени     ->
<!-  ============================================    ->
<date>
  <dateyyyy-mm-dd>2001-12-21</dateyyyy-mm-dd>
  <dateyyyy-mm>2001-12</dateyyyy-mm>
```

```
    <datemm-dd>12-21</datemm-dd>
    <dateyyyy>2001</dateyyyy>
    <datemm>12</datemm>
    <datedd>21</datedd>
    <datetime>2001-12-21T00:00:00</datetime>
    <duration>P1Y2M3DT02H03M04S</duration>
  </date>

  <time>
    <timebase>23:01:01</timebase>
    <timezone-offset>23:01:01-01:00</timezone-offset>
  </time>

  <!-  ============================================   ->
  <!- Фрагмент W3C XML-схемы - элементы для даты ->
  <!-  ============================================   ->
  <xs:element name="date">
    <xs:complexType>
      <xs:sequence>
        <xs element ref="dateyyyy-mm-dd"/>
        <xs element ref="dateyyyy-mm"/>
        <xs element ref="datemm-dd"/>
        <xs element ref="dateyyyy"/>
        <xs element ref="datemm"/>
        <xs element ref="datedd"/>
        <xs element ref="datetime"/>
        <xs element ref="duration"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

        <xs element name="dateyyyy-mm-dd" type="xs:date"/>
        <xs element name="dateyyyy-mm" type="xs:gYearMonth"/>
        <xs element name="datemm-dd" type="xs:gMonthDay"/>
        <xs element name="dateyyyy" type="xs:gYear"/>
        <xs element name="datemm" type="xs:gMonth"/>
        <xs element name="datedd" type="xs:gDay"/>
        <xs element name="datetime" type="xs:duration"/>
        <xs element name="duration" type="xs:dateTime"/>

  <!-  ============================================  ->
  <!- Фрагмент W3C XML-схемы - элементы для времени ->
  <!-  ============================================ ->
  <xs:element name="time">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="timebase"/>
        <xs:element ref="timezone-offset"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="timebase" type="xs:time"/>
  <xs:element name="timezone-ofJset" type="xs:time"/>
```

Основные целочисленные типы данных W3C XML-Схемы

```
<!-  ============================================  ->
<!- Фрагмент XML-документа - элементы для целых чисел->
<!-  ============================================  ->
<integer>
  <integerbase>99999999999999999</integerbase>

  <integerlongmin>-9223372036854775808</integerlongmin>
  <integerlongmax>+9223372036854775807</integerlongmax>

  <integerintmin>-2147483648</integerintmin>
  <integerintmax>+2147483647</integerintmax>

  <integershortmin>-32768</integershortmin>
  <integershortmax>+32767</integershortmax>

  <integerbytemin>-128</integerbytemin>
  <integerbytemax>+127</integerbytemax>
</integer>

<!-  ============================================  ->
<!- Фрагмент W3C XML-схемы -элементы для целых чисел->
<!-  ============================================  ->
<xs:element name="integer">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="integerbase"/>
      <xs:element ref="integerlongmin"/>
      <xs:element ref="integerlongmax"/>
      <xs:element ref="integerintmin"/>
      <xs:element ref="integerintmax"/>
      <xs:element ref="integershortmin"/>
      <xs:element ref="integershortmax"/>
      <xs:element ref="integerbytemin"/>
      <xs element ref="integerbytemax"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="integerbase" type="xs:integer"/>
<xs:element name="integerlongmin" type="xs:long"/>
<xs:element name="integerlongmax" type="xs:long"/>
<xs:element name="integerintmin" type="xs:int"/>
<xs:element name="integerintmax" type="xs:int"/>
<xs:element name="integershortmin" type="xs:short"/>
<xs:element name="integershortmax" type="xs:short"/>
<xs:element name="integerbytemin" type="xs:byte"/>
<xs:element name="integerbytemax" type="xs:byte"/>
```

Основные десятичные типы данных W3C XML-Схемы

```
<!-  ============================================  ->
<!-  Фрагмент XML-документа - элемент для десятичных чисел ->
<!-  ============================================  ->
<decimal>
  <decimalbase>99999999999999999999999999999.99999999
  </decimalbase>
</decimal>

<!-============================================  ->
<!-Фрагмент W3C XML-схемы -элемент для десятичных чисел ->
<!-============================================  ->
<xs:element name="decimal">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="decimalbase"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="decimalbase" type="xs:decimal"/>
```

Основные логические типы данных W3C XML-Схемы

```
<!-============================================  ->
<!-      Фрагмент XML-документа - элементы для логических значений->
<!-============================================  ->
<boolean>
  <booleantrueliteral>true</booleant rueliteral>
  <booleanfalseliteral>false</booleanfalseliteral>
  <booleantrue>1</booleantrue>
  <booleanfalse>0</booleanfals>
</boolean>

<!-  ============================================  ->
<!-  Фрагмент W3C XML-схемы - элементы для логических значений ->
<!-  ============================================  ->
<xs:element name="boolean">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="booleantrueliteral"/>
      <xs:element ref="booleanfalseliteral"/>
      <xs:element ref="booleantrue"/>
      <xs:element ref="booleanfalse"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="booleantrueliteral" type="xs:boolean"/>
```

```
<xs:element name="booleanfalseliteral" type="xs:boolean"/>
<xs:element name="booleantrue" type="xs:boolean"/>
<xs:element name="booleanfalse" type="xs:boolean"/>
```

Основные типы данных URI W3C XML-Схемы

```
<!-  ============================================  ->
<!-  Фрагмент XML-документа - элемент для URI->
<!-  ============================================  ->
<URL>
  <anyURI>www.xyz-abc.com</anyURI>
</URL>

<!-  ============================================  ->
<!-  Фрагмент XML-схемы - элемент для URI ->
<!-  ============================================  ->
<xs:element name="URL">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="anyURI"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="anyURI" type="xs:anyURI"/>
```

Основные строковые типы данных W3C XML-Схемы

```
<!-  ============================================  ->
<!-  Фрагмент XML-документа - элементы для строк ->
<!-  ============================================  ->
<string>
  <stringbase>abcdefg 12345678 CR LF TAB</stringbase>
  <stringnormalized>abcdefg</stringnormalized>
  <stringtoken>abcdefg</stringtoken>
</string>

<!-  ============================================  ->
<!-  Фрагмент XML-схемы -элементы для строк ->
<!-  ============================================  ->
<xs:element name="string">
  <xs:complexType>
    <xs:sequences>
      <xs:element ref="stringbase"/>
      <xs:element ref="stringnormalized"/>
      <xs:element ref="stringtoken"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
<xs:element name="stringbase" type="xs:string"/>
<xs:element name="stringnormalized" type="xs:normalizedstring"/>
<xs:element name="stringtoken" type="xs:token"/>
```

Как будет видно из главы 5, каждый из предопределенных типов данных W3C XML-Схемы (примитивных и производных) можно в дальнейшем дополнительно ограничить применением фасет типов данных. В результате этого образуются нестандартные типы данных.

Использование идентификаторов

Хотя это и не относится напрямую к предыдущему разделу о типах данных, W3C XML-Схемы (так же, как ряд других типов схем) обеспечивают поддержку типа "ID". В зависимости от использования и определения идентифицирующих данных, может потребоваться применение уникального значения (идентификатора), связанного с элементами или атрибутами XML-документа. Эта концепция не отличается от уникального идентификатора сущности из логической модели или первичного ключа таблицы базы данных. В случае с типом "ID" языка XML существуют некоторые ограничения.

Уникальный идентификатор можно определить как один или несколько элементов данных, уникально отличающих один набор данных от другого в пределах одного и того же контекста. Уникальные идентификаторы используются во многих системах предприятия для идентификации покупателей, планов, поставщиков, продукции, единиц, подразделений, сборок, физического имущества, людских ресурсов и многих других фундаментальных понятий из области данных. Элементы данных, идентифицирующие эту информацию, часто называют примерно так: "Customer Number" (код покупателя), "Vendor ID" (идентификатор поставщика) или "Part Number" (номер подразделения). Когда идентифицирующие данные хранятся в базе данных (т. е. они "persisted"-постоянные), уникальные идентификаторы часто играют роль первичных ключей или являются какой-либо формой уникального индекса.

Правила создания и присвоения идентифицирующих значений меняются от системы к системе. В некоторых системах при создании нового экземпляра данных (например, новой записи о покупателе), в качестве значения уникального идентификатора (первичного ключа) присваивают очередной порядковый номер. В других – создают значение уникального идентификатора как комбинацию значений других данных, как результат работы некоторого алгоритма или на основе текущего времени. Идентификаторы могут быть содержательными или несодержательными. Содержа-

тельный идентификатор включает в себя одну или несколько узнаваемых характеристик, несущих информацию для человека или прикладной программы. Несодержательные идентификаторы не заключают в себе никакой информации, зачастую это бессмысленный набор цифр или символов.

Рекомендация. *Идентифицирующие данные, являющиеся критически важными для обработки транзакции или сообщения XML, необходимо включать как элемент или атрибут и адекватно описывать с помощью имени или местоположения внутри транзакции (или сообщения).*

Так же, как для сущности модели данных или таблицы базы данных, уникальные идентификаторы могут иметь большое значение и для XML-транзакций. Это особенно справедливо, когда XML-транзакция включает в себя более одной совокупности однотипных данных (например, в одном XML-документе содержится информация о нескольких покупателях, заказах и продуктах). Возможность отличать одну совокупность данных от другой часто имеет критическое значение для работы приложения. Даже в тех случаях, когда XML-документ содержит лишь одну совокупность данных (т. е. информацию об одном покупателе, заказе или продукте), включение идентификаторов, извлеченных из источника этих данных, также оправдано. Идентифицирующая информация поможет избежать возможной неопределенности, дублирования или коллизий данных при обработке транзакции.

Существует несколько способов определения идентификатора для XML-транзакции. Идентификатор можно определить как XML-элемент. Зачастую этот элемент является дочерним элементом группы подобных или связанных элементов. Другим способом является определение идентификатора в качестве атрибута элемента (рис. 4.3).

Некоторые идентификаторы являются комбинацией нескольких значений данных. Когда идентификатор представляет собой совокупность раздельно определенных значений данных (его называют составным идентификатором или составным ключом), эти отдельные значения должны быть описаны в XML-документе отдельными элементами или атрибутами. Для упрощения навигации и обработки, элементы, составляющие идентификатор, необходимо размещать рядом друг с другом (возможно, как элементы, вложенные в общий родительский или групповой элемент). Также их следует задавать в правильной последовательности (рис. 4.4).

В некоторых случаях значения данных составного ключа обрабатываются как одно составное значение. Если же значения такого идентификатора обрабатывается и как отдельные элементы, и как одно составное значение, можно ввести дополнительный элемент или атрибут, содержащий сцепленное воедино значение. Это не очень желательный подход, поскольку значение такого компонента маловразумительное и неделимое. Составное или сцепленное значение данных идентифи-

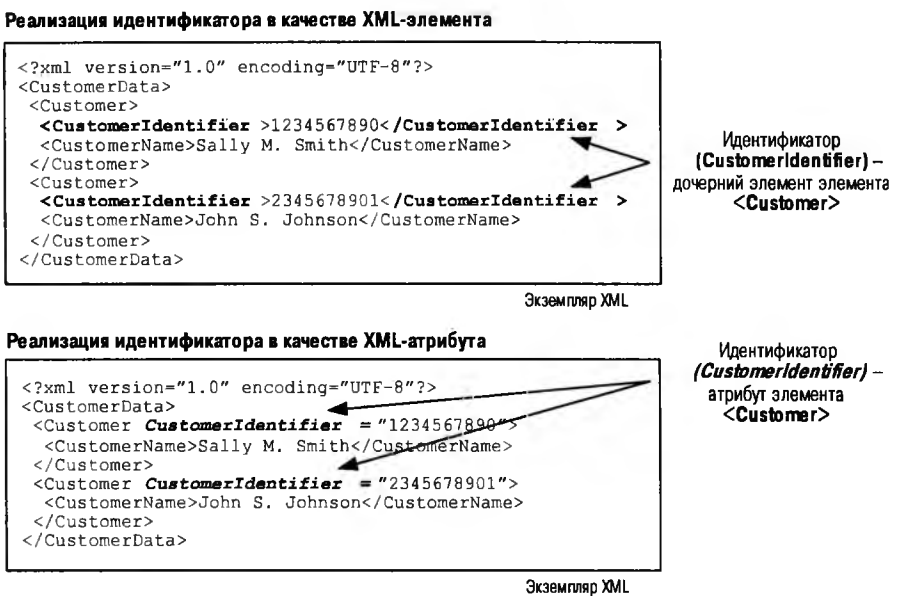


Рис. 4.3. Способы определения идентификаторов для XML-транзакции

катора должно быть вычисляемым (XML и W3C XML-Схемы не предоставляют встроенной возможности вычислять значения данных). Независимо от способа реализации уникальных идентификаторов в XML-транзакции, необходимо обеспечить адекватный уровень их детализации и состав.

Рекомендация. *В XML-транзакции должен использоваться тот же уровень детализации и состав уникальных идентификаторов, что определен источником данных или пунктом назначения. Если идентификатор является комбинацией отдельных элементов данных (составной ключ), для них должны быть определены аналогичные (источнику или пункту назначения) контейнеры.*

Хотя обеспечение передачи идентификатора как части XML-транзакции имеет большое значение, приложение при обработке идентификатора не обязано само проверять его уникальность. W3C XML-Схемы предоставляют тип данных – уникальный идентификатор ("ID"), который можно использовать при определении любого элемента или атрибута XML (рис. 4.5). При верификации XML-транзакции в соответствии с W3C XML-Схемой, синтаксический анализатор проверяет, нет ли одинаковых значений уникальных идентификаторов в пределах данного XML-документа, и, при обнаружении такой ситуации, информирует вызвавшее его приложение о нарушении уникальности или ошибке.

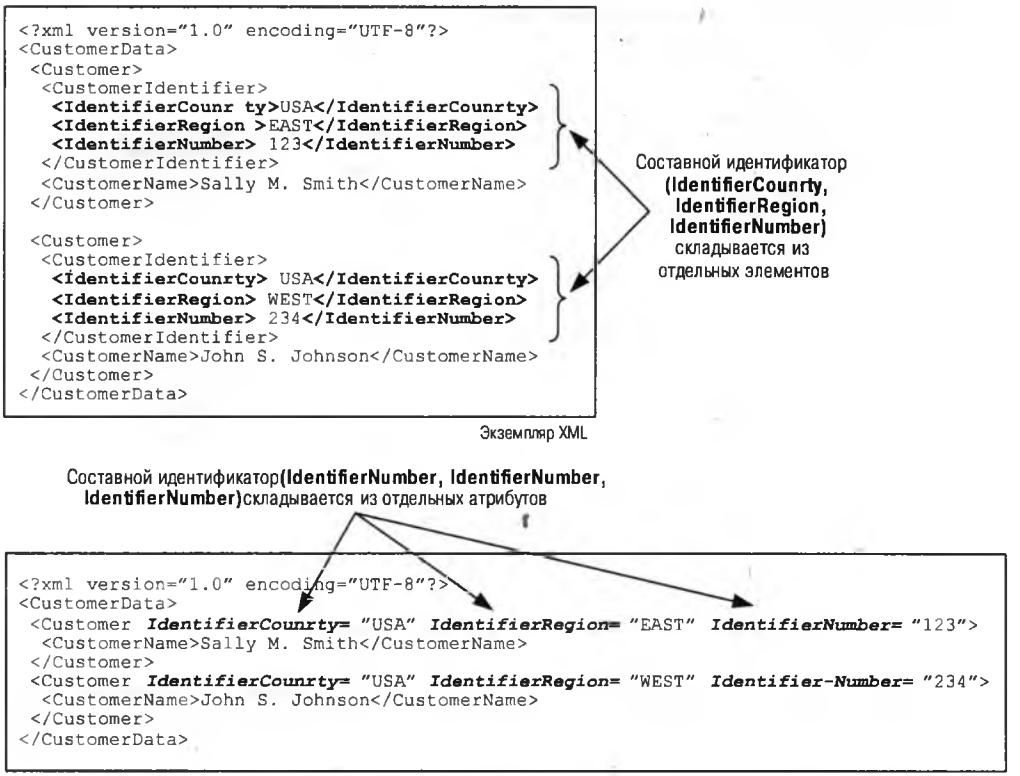


Рис. 4.4. Составные идентификаторы

Существует ряд ограничений, касающихся данных типа идентификатор ("ID"). Во-первых, тип "ID" в процессе верификации проверяется только на уникальность значений. Во-вторых, уникальность значений, содержащихся в контейнерах (элементах и атрибутах), объявленных как "ID", проверяется только в ределах отдельного XML-документа (транзакции). В XML или W3C XML-Схемах нет возможности проконтролировать уникальность идентификаторов по отношению ко внешним источникам данных, например базе данных. В-третьих, тип "ID" является производным от типа "string", значения этого типа должны отвечать синтаксису, используемому при определении имен элементов и атрибутов XML (т. е. не содержать пробелов, не начинаться на цифру и не включать двоеточие). Значения данных, определенные с типом "ID", должны быть строкой и не должны начинаться с цифры.

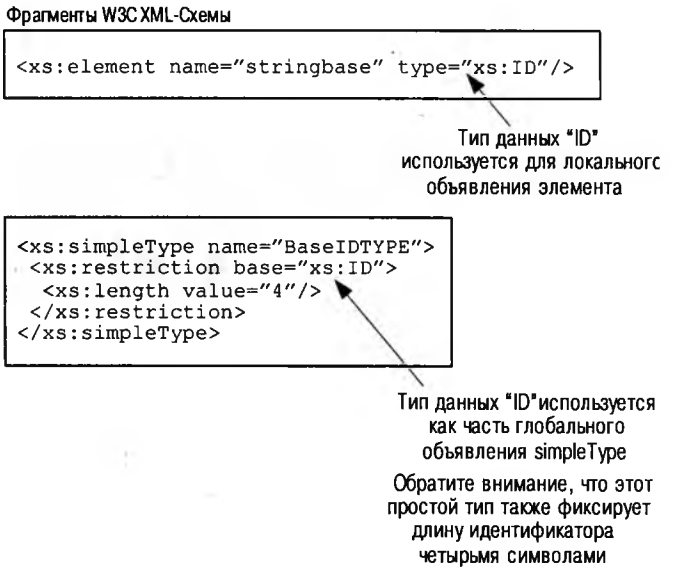


Рис. 4.5. Синтаксис использования типа данных "ID" в W3C XML-Схеме

Рекомендация. *Использование типа данных – уникальный идентификатор ("ID") W3C XML-Схемы должно ограничиваться только теми элементами и атрибутами, которым требуется уникальность содержащихся в них значений в пределах одного XML-документа. Из-за ограничений на допустимые значения тип "ID" невозможно использовать для описания первичных ключей базы данных или аналогичных уникальных идентификаторов. Проверка уникальности идентификаторов базы данных следует оставить самим системам баз данных.*

В дополнение к типу данных, можно установить ряд дополнительных ограничений на значения данных. К примеру, для элемента, определенного как "string", можно ограничить минимально и максимально допустимую длину и набор допустимых значений. Эти дополнительные ограничения определяются с помощью фасет типов данных.

Глава 5

Фасеты типов данных W3C XML-Схемы

Фасеты – это дополнительные ограничения, которые можно наложить на тип данных W3C XML-Схемы. Или, с позиции метаданных, фасеты – это метаданные, представляющие собой правила или ограничения для типов данных. Проектировщики данных быстро оценят расширенные возможности поддержки типов данных, предоставляемые фасетами. Наиболее знакомые из них – те, что касаются длины данных и количества знаков после запятой в десятичных числах. Подобные ограничения часто вводятся для атрибутов сущностей логической модели данных, для элементов объектов физической модели, а также для столбцов таблиц реляционной базы данных. Кроме того, многие процедурные и объектно-ориентированные языки программирования обладают возможностью задавать ограничения, аналогичные фасетам.

К наиболее употребимым фасетам относятся:

- фасет, ограничивающий длину (как количество символьных позиций);
- фасет значения (наибольшее и наименьшее возможное значения, как диапазон или порог);
- фасет, ограничивающий количество цифр десятичного числа (как общее количество цифр, так и количество цифр после запятой);
- фасет перечисления (список допустимых значений)
- фасет шаблона (шаблоны редактирования, включающие литералы и допустимые цифры);
- фасеты пробельных символов (применяются к строкам или символьным данным).

Ограничение на длину чаще всего используется в виде задания максимально возможной длины элемента данных. Символьные или строковые типы баз данных часто ограничены наибольшим возможным количеством символьных позиций. Точно так же для десятичного типа данных может быть установлено ограничение на общее количество цифр (часто это называют разрядностью) и количество цифр дробной части (часто это называют масштабом). W3C XML-Схемы поддерживают фасеты ограничения длины (несколько видов), в том числе, для десятичных чисел – общего количества цифр и количества цифр в дробной части.

Строковые значения данных можно ограничить с помощью нескольких разных фасетов длины. В W3C XML-Схемах поддерживается фасет фиксирования длины, фасет установки минимально возможной длины и фасет установки максимально возможной длины. Кроме фасетов длины, для строк поддерживается ряд других фасетов: фасет перечисления (список допустимых значений) и фасеты управления тем, сохранятся ли после верификации символы возврата каретки, перевода строки и табуляции.

Десятичные типы данных можно ограничить по длине, по количеству цифр после запятой, по допустимым значениям. Интересно, что ограничение по длине для числового типа данных, такого как "integer", синтаксически выглядит так же, как ограничение длины строки (те же фасеты). Ограничение длины десятичных типов производится с помощью фасета "totalDigits".

Еще более мощной разновидностью фасетов является шаблон. С помощью шаблона можно указать допустимые для строки символы, а также ожидаемые литералы. Рассмотрим номер социального страхования (Social Security Number – SSN), используемый в США. Его формат выглядит так: "999-99-9999". Согласно ему, за тремя цифрами следуют еще две, а затем еще четыре. Для отделения этих трех групп цифр друг от друга используются тире (литерал "-"). В W3C XML-Схемах для контроля за соблюдением подобных форматов используются шаблоны.

Факт. *Фасеты служат для задания пределов, ограничений и описания типов данных. В некоторых случаях, в зависимости от типов данных, фасеты можно комбинировать друг с другом.*

W3C XML-Схемы обеспечивают расширенную поддержку фасетов. В зависимости от типа данных, к которому они применяются и контекста, фасеты можно комбинировать друг с другом. К примеру, значения XML-элемента, имеющего десятичный тип данных, можно ограничить несколькими фасетами: общее количество цифр, количество цифр после запятой и диапазон допустимых значений.

Символьная длина

Большинство продуктов реляционных баз данных поддерживают несколько строковых типов данных. Один из таких типов известен, как "character" или "char", другой – "variable character" или "varchar". Как следует из названия, тип "character" предназначен для описания или хранения символьных (строковых) данных. Тип "variable character" – для строк переменной длины (количество символов в которых может меняться в зависимости от характеристик данных). Фасет длины, пожалуй, чаще всего используется для строковых данных.

Несмотря на слово "variable" (переменная), большинство типов данных для описания строк переменной длины имеют предел в виде максимально возможного числа символов, которое могут иметь строки этого типа. Содержимое столбца базы данных для строк переменной длины не может превышать по длине некоторого определенного числа символов. Теоретически, содержимое, имеющее меньшую длину, занимает меньше места в базе данных. Физическая реализация базы данных может быть разной и реальный размер пространства на диске, необходимый для хранения данных, включает также дополнительные байты, используемые базой данных для внутренних ссылок и т. п. Подобно строковым данным переменной длины, столбец базы данных с типом "character" также может содержать данные не длиннее заданного максимума. Типы "character" обычно предназначены для описания строк фиксированной длины (занимающих на диске одинаковое по размеру пространство, независимо от реальной длины хранящихся строк).

Техника. *Фасеты символьной длины можно использовать для разрешения отличий в характеристиках метаданных между источником и получателем данных, путем ограничения числа символьных позиций до наименьшего общего значения.*

W3C XML-Схемы поддерживают фасет длины – аналог ограничения длины в базах данных. Однако W3C XML-Схемы идут дальше, чем просто описание максимально возможной или фиксированной длины. Поддерживаемые фасеты могут ограничивать минимальную длину, максимальную длину и задавать фиксированную длину (рис. 5.1). Фасет *length* фиксирует количество символов. При его применении для элементов и атрибутов XML содержащиеся в этих контейнерах значения должны всегда иметь указанную длину (не больше и не меньше). При использовании фасета *minLength* содержащиеся значения должны иметь длину, равную указанной или больше ее. Наоборот, фасет *maxLength* задает длину данных не превышающую указанной.

Как уже отмечалось, некоторые фасеты можно использовать в комбинации друг с другом, ограничивая тип данных для элемента или атрибута. Так, при совместном использовании фасетов *minLength* и *maxLength* в отношении одного XML-элемента, получаем допустимый диапазон длин (например, не менее 8 позиций и не более

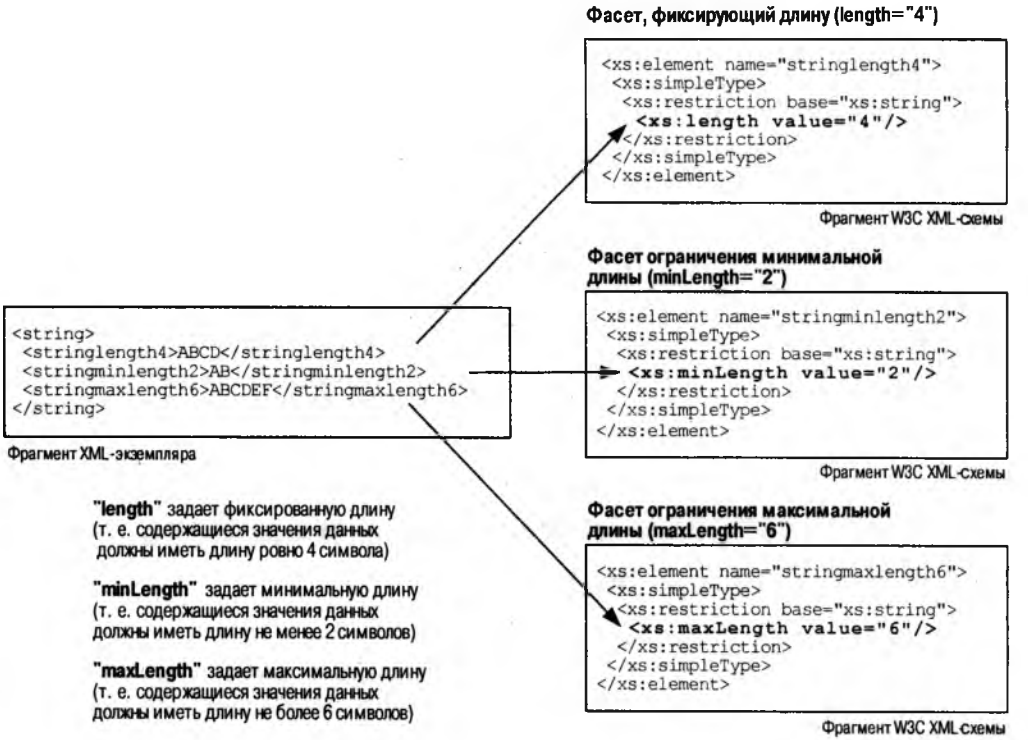


Рис. 5.1. Фасеты длины

20 позиций). Комбинирование фасетов длины может оказаться полезным для разрешения расхождений в описании данных в источнике данных и в пункте назначения, когда отличия заключаются в длине и не находятся в прямом противоречии. Однако, естественно, ограничения длины не разрешат всех форм расхождений в данных, и даже могут внести некоторые осложнения (например, усечение значений данных). Проектировщику данных необходимо уточнить разновидность расхождений и определить, как могут помочь фасеты длины.

Модальность (modality) – термин, описывающий концепцию необязательности (опциональности) или обязательности присутствия. XML-элемент определяется как обязательный присвоением *minOccurs* значения "1", атрибут определяется как обязательный присвоением *use* значения "required". Соответственно, элемент, определенный со значением *minOccurs,* равным "0", и атрибут, определенный со значением *use*, равным "optional", являются необязательными. Но такое объявление модальности указывает, обязательно ли наличие контейнера (элемента или атрибута) в ссы-

лающемся на схему XML-документе, но не объявляет, должен ли контейнер обязательно содержать значение или он может быть пустым. И хотя XML-элемент можно определить как обязательный (значение *minOccurs* равно "1"), он может находиться в XML-документе, не содержа данных.

Задание в W3C XML-схеме фасету minLength значения "1" может служить способом контроля обязательности присутствия значения данных. Однако помните, что это присутствующее значение может быть ошибочным. К примеру, одиночный пробел, помещенный в XML-элемент, успешно пройдет верификацию на соответствие фасету minLength со значением "1". Однако такое значение данных может быть неприемлемым для данного XML-элемента по смыслу.

Техника. *Фасет minLength может использоваться для контроля обязательности присутствия значения данных в элементе или атрибуте XML. Если его значение равно "1", верифицирующий синтаксический анализатор проконтролирует, что значение данных соответствующего контейнера, по крайней мере, равно одному пробелу (непустое). Если не будет совсем никакого значения (что эквивалентно "null") будет возбуждена ошибка верификации.*

Границы значений (наименьшее и наибольшее возможные значения)

Границы значений подобны диапазону, задаваемому минимальным и максимальным допустимым значениями. При применении фасетов минимального и максимального значений к некоторому контейнеру синтаксический анализатор во время верификации следит за тем, чтобы значение из этого контейнера находилось в указанном диапазоне. Если это значение выходит за границы диапазона, возбуждается ошибка.

Техника. *В некоторых случаях фасеты ограничения значений (minInclusive (>=), maxInclusive (<=), minExclusive (>), maxExclusive (<)) можно использовать для разрешения расхождений между источниками и получателями данных путем ограничения допустимых значений общим для них диапазоном .*

Поддержка границ значений в W3C XML-Схемах выходит за рамки обычного диапазона значений (рис. 5.2). Проектировщик данных или проектировщик схемы может задать включающую группу (т. е. группу значений, в которой значения данных должны находиться между заданными минимумом и максимумом, включая значения минимума и максимума), а может задать исключающую группу значений (группу значений, в которой значения данных должны находиться между заданными минимумом и максимумом, не включая эти граничные значения). Границы значений включающей группы задаются с помощью minInclusive и maxInclusive. Границы

значений исключающей группы задаются с помощью minExclusive и maxExclusive. Проектировщик данных может задать минимальное значение и не задать максимального или максимальное без минимального. В зависимости от типа данных, для которого задаются эти фасеты, это позволит иметь неограниченные значения.

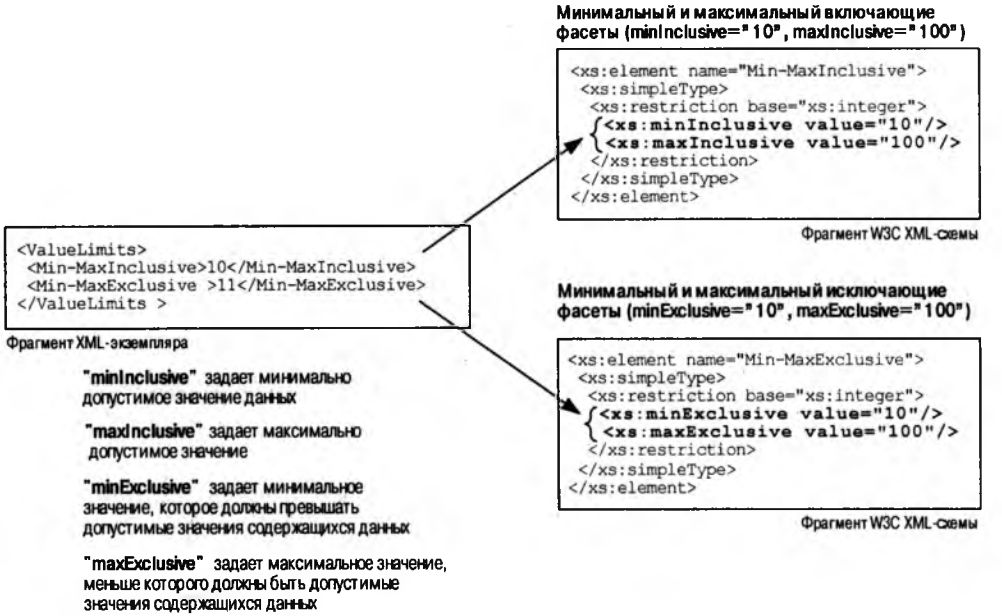


Рис. 5.2. Фасеты границ значений

Цифры (количество и тип)

Цифровые фасеты, определенные для числовых типов данных (decimal и производных от него: integer, long, int, short и byte), примерно аналогичны фасетам длины для строковых типов данных. Подобные ограничения, хотя и с иным синтаксисом, присутствуют в большинстве реляционных баз данных. W3C XML-Схемы поддерживают фасеты, ограничивающие общее количество цифр и количество цифр дробной части (рис. 5.3).

Для строковых или символьных данных фасеты длины ограничивают допустимое количество символьных позиций. Аналогично, цифровые фасеты ограничивают общее количество цифр (разрядность) и количество цифр после запятой (десятичный масштаб) для числовых типов данных. Фасет totalDigits описывает общее количество цифр для числового типа данных (включая цифры целой части

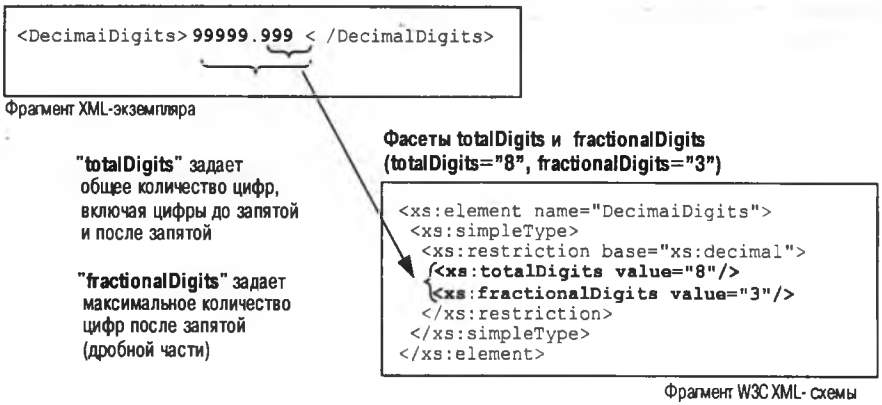


Рис. 5.3. Цифровые фасеты

и дробной части). Фасет fractionDigits задает количество цифр дробной части, которые следуют после запятой, не считая саму запятую (т. е. справа от запятой). Десятичная запятая и необязательный знак числа не включаются в количество, задаваемое обоими фасетами.

Техника. *Цифровые фасеты можно использовать для разрешения различий в характеристиках метаданных между источником и получателем данных, путем ограничения количества цифр и символьных позиций до наименьшего общего.*

Одной из часто встречающихся форм различий в данных между автономными системами предприятия является различие в определениях метаданных для денежных сумм. В системах финансовой отчетности, обработки заказов, управления складом, управления счетами и т. п. часто встречаются различия в наибольшем возможном количестве цифр и количестве цифр после запятой. К примеру, в системе оформления покупок цена единицы продукта может быть определена как десятичное число общей длиной в 7 цифр, из которых 2 – после запятой (в языке Cobol: “pic S99999v99”). Однако в системе управления складом эта цена может быть определена как десятичное число общей длиной 7 цифр, из которых 3 – после запятой (вязыке Cobol: “pic S9999v999”). Обмен данными между этими двумя системами может привести к отсечению, округлению и потенциальным ошибкам.

Большинство верифицирующих синтаксических анализаторов будет интерпретировать цифровые фасеты W3C XML-Схемы как наибольшее допустимое количество цифр. Обращаясь к предыдущему примеру различий в определениях цены единицы продукта, можно определить какой тип данных способен содержать значения данных из обеих систем, не вызывая ошибок. Новый нестандартный тип данных может быть определен как десятичный тип, с наибольшим общим количеством цифр

и количеством цифр после запятой. Сочетание общего количества цифр из обеих систем определит значение, которое необходимо установить в фасете totalDigits. Получившийся в результате тип данных можно затем использовать для описания цен продуктов в содержимом для обмена данных между разными приложениями предприятия. Однако такая техника не свободна от потенциальных ошибок. Значения данных, превышающие ограничения получающей системы, потребуют отбрасывания лишних цифр, округления или применения дополнительной бизнес-логики.

Еще более важной является возможность использования фасетов totalDigits и fractionalDigits для определения нескольких стандартных типов данных для денежных сумм. Таким образом, проектировщик данных может упорядочить множество финансовых и денежных элементов данных предприятия и определить, какие из характеристик метаданных являются для них общеприемлемыми. Для каждого из элементов можно определить нестандартный тип данных (используя simpleType) с помощью фасетов totalDigits и fractionalDigits. Полученные нестандартные типы данных будут представлять стандарты метаданных предприятия для описания денежных сумм. Эти типы можно применять для денежных сумм при проектировании новых документов, транзакций и сообщений XML.

Другой сферой применения фасетов totalDigits и fractionalDigits является глобальная электронная коммерция. В Северной Америке большинство денежных сумм определяется с двумя знаками после запятой. При рассмотрении разных валют мира этот подход часто недействителен. Некоторые валюты определяются вообще без дробной части, в других требуется 2 или 3 цифры после запятой. К примеру, динар Бахрейна, динар Ирака, динар Иордании и динар Кувейта могут потребовать трех цифр после запятой. А песета Андорры, рубль Беларуси, франк Коморов, франк Гвинеи, лира Италии и т. д. не требуют дробной части[1]. Для поддержки региональных единиц, можно создать соответствующие нестандартные типы данных с разными фасетами fractionDigits. Очевидное достоинство W3C XML-Схем заключается в их возможности определять или, скорее, расширять типы данных с помощью фасета дробной части, применительно к нуждам валют разных государств.

Перечисления (допустимые значения)

Некоторые элементы данных, согласно бизнес-правилам, могут принимать значения только из определенного набора значений. Для проектировщиков данных эта концепция также известна под названием "набор допустимых значений" или как элементы данных, представляющие стандартные "наборы кодов" ("code sets"). Вот некоторые из наиболее употребимых промышленных

[1] Currency Data Concept, Web Globalization Guide Framework. James Bean © 1996-2002. Доступно по адресу: http://www/globalwebarch.com/. – *Примеч. авт.*

и международных стандартных наборов кодов: аббревиатуры штатов США, коды языков ISO 639, коды стран ISO 3166, коды валют ISO 4217. Каждый из этих стандартов включает строго определенные "коды", представляющие допустимые значения данных для штата США, языка, страны и валюты. Традиционные приложения баз данных сравнивают значения кодов с соответствующими справочными таблицами стандартов, или, в некоторых случаях, могут включать списки значений стандартных кодов внутри логики программы (например, "88" уровни COBOL-программы).

Техника. *Фасеты перечисления (список допустимых значений) могут использоваться для поддержания внутренних стандартов и, в некоторых случаях, для разрешения различий в характеристиках метаданных между источником и получателем данных, путем ограничения допустимых значений набором, согласованным с обеими системами.*

Фасеты перечисления описывают список или набор допустимых значений данных для элемента или атрибута XML. Возможности перечислений W3C XML-Схем выходят за рамки просто справочных таблиц (рис. 5.4). К слову сказать, DTD-типы схем позволяют задавать перечисления только для атрибутов, тогда как W3C XML-Схемы позволяют задавать перечисления и для атрибутов, и для элементов. В сценарии, где транзакции интеграции предприятия проходят через множество автономных систем, прикладная программа одной из систем может не иметь доступа к справочной таблице или списку допустимых значений поддерживаемых другой системой. Для транзакции обмена это не позволит передающей системе быть уверенной в том, что принимающая система сможет распознать корректность передаваемых в транзакции значений. Данный сценарий можно расширить, рассмотрев B2B-приложения электронной коммерции, где транзакции перемещаются между внешними по отношению к предприятию приложениями. В этом случае данные могут посылаться и приниматься многими сотрудничающими предприятиями и торговыми партнерами. Возможность доступа к единому общему репозитарию допустимых значений – неэффективное и непрактичное решение. Точно так же попытки синхронизации справочных таблиц для большого числа источников и получателей данных не реалистичны.

Как и транзакция, система – источник данных также может включать W3C XML-Схему (или подсхему-компонент). Система – источник определяет с помощью фасетов перечисления стандартные значения кода и включает их в W3C XML-Схему. При таком подходе получающая система может удостовериться, что все полученные ею значения (или данные, подготовленные к отправке в ответной транзакции) согласуются со списком допустимых значений из схемы.

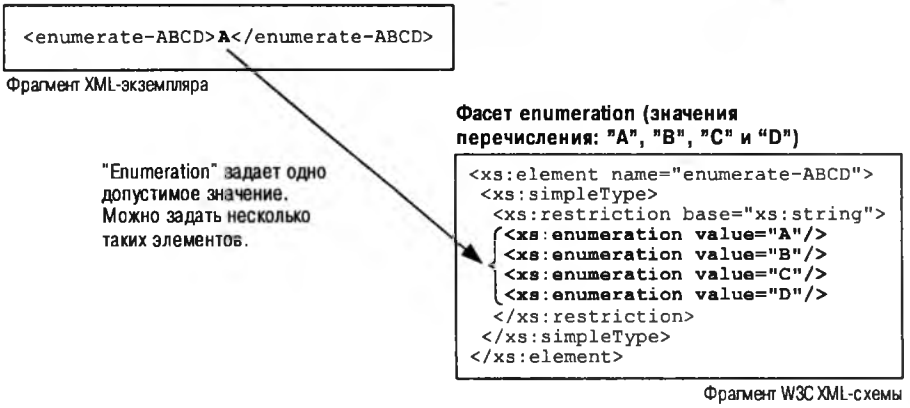


Рис. 5.4. Фасеты перечислений (допустимых значений)

Техника. *Фасеты перечисления (список допустимых значений) можно использовать для описания стандартов данных, таких, как наборы кодов (например, коды стран, валют, штатов США). Однако следует соблюдать осторожность в случаях, когда списки перечислений определяются внутри нескольких схем, а не во внешнем подмножестве схем, на которые можно ссылаться.*

К спорной области в использовании списков перечислений относится частота их изменения. Значения стандартных кодов стран и валют время от времени меняются. И хотя эти изменения происходят не ежедневно и даже не ежемесячно, это реальность. Когда допустимые значения определяются в статических списках, периодически возникает необходимость их обновления. Если эти списки напрямую встроены в несколько W3C XML-Схем (как реплицируемые копии), за каждой из этих схем необходимо следить и каждую необходимо поддерживать в актуальном состоянии. Подобные подходы характеризуются не только увеличенной стоимостью обслуживания, но и могут привести к ошибкам. Использование перечислений, которые определены внутри нескольких схем (т. е. несколько раз "жестко прошиты"), не рекомендуется. Более эффективный подход – использовать возможности W3C XML-Схем по созданию подсхем-компонентов, при этом списки допустимых значений определяются в единственной внешней подсхеме-компоненте и затем, по необходимости, повторно используется другими схемами с помощью ссылок (мы обсудим эту тему в главе 8).

Несмотря на всю полезность перечислений, с ними следует соблюдать осторожность. При проектировании и разработке каждой из W3C XML-Схем следует учитывать степень их обоснованности. Предыдущая техника подразумевает, что во многих случаях W3C XML-Схемы включаются в XML-транзакцию для проверки того, что источник и получатель данных "согласовали" верификацион-

ные правила и, в частности, списки допустимых значений. Обычно эффективный, в некоторых сценариях такой подход может оказаться непрактичным. Если список перечисления имеет большой размер или часто обновляется, может снизиться производительность и возрасти стоимость обслуживания. В таких случаях рекомендуется использовать иные архитектурные решения.

Техника. *Списки перечислений, обычно полезные, не всегда являются лучшим архитектурным решением для проверки допустимых значений, особенно если размеры списков велики (более 200 значений), или эти списки часто обновляются (чаще 1-2 раз в месяц).*

Шаблоны

Фасеты шаблонов ограничивают внешний вид или форму значений данных (не путайте с шаблонами проектирования или шаблонами объектов). Можно определить шаблоны для описания: допустимых символов и позиций, которые эти символы могут занимать, количества однотипных символов в логической группе, включаемых в данные литералов. Примером может служить шаблон уже упоминавшегося номера социального страхования США (рис. 5.5). Фасеты шаблонов W3C XML-Схем можно также использовать при описании форматов почтового индекса, телефонных номеров или других аналогичных данных, чей формат должен быть ограничен.

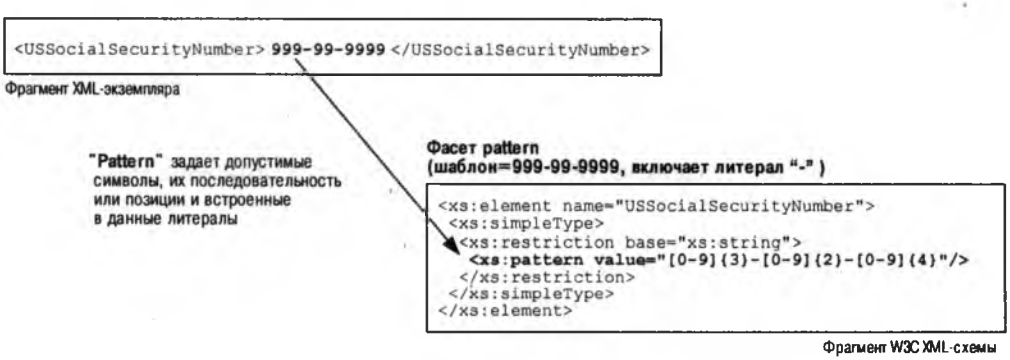


Рис. 5.5. Фасеты шаблонов

Техника. *Фасеты шаблонов можно использовать для ограничения допустимых символов и возможных позиций этих символов в данных (наподобие формата отображения). Так же, как и списки перечислений, шаблоны необходимо выделять во внешние схемы и затем ссылаться на них, чтобы избежать лишних затрат на поддержание схемы.*

Традиционно проектировщики данных считают, что шаблоны представляют собой данные со встроенным интеллектом[2] или данные, допускающие денормализацию. Если в базу данных данные вводятся по шаблону, это гарантирует ее адекватность. Обычно не рекомендуется хранить данные в их "парадном виде" (вместе с полагающимися по шаблону литералами). Однако в контексте транзакций глобальной электронной коммерции или интеграции предприятия, возможность предварительно описать шаблоны данных, и затем каждый раз проверять данные транзакции на соответствие этим шаблонам, очень важна. Предположим, в некотором сценарии глобальной электронной коммерции нам необходимо работать с разными форматами почтового индекса. Почтовые индексы разных стран могут включать разные формы сведений, разные последовательности их указания и шаблоны. В W3C XML-Схему можно включить отдельный шаблон для каждой из стран, в которых имеются отличия в почтовом индексе.

Пробельные символы (white space)

Термин *пробельные символы* может сбить с толку. По определению схемы, пробельные символы – это просто пробел, символ возврата каретки, символ перевода строки и символ табуляции, находящиеся в строке. При использовании строкового или символьного содержимого, присущего документноориентированной транзакции, может потребоваться введение запрета или ограничения на использование пробельных символов. К примеру, содержимым элемента может быть текстовый параграф, включающий символы табуляции для позиционирования или выравнивания данных. Однако система, принимающая данные, может не распознавать или некорректно обрабатывать символы табуляции. W3C XML-Схемы поддерживают фасеты пробельных символов, описывающие три разные возможности (рис. 5.6).

Рекомендация. *В случаях, когда содержимое XML-документов не является документноориентированным, следует избегать использования фасетов пробельных символов. В зависимости от синтаксического анализатора и процесса верификации, использование фасетов пробельных символов может привести к тому, что извлеченные получателем данные будут отличаться от первоначального содержимого XML-документа.*

При определении фасета пробельных символов значение "preserve" означает, что все пробельные символы (возврат каретки, перевод строки, табуляция) должны быть сохранены или оставлены в их исходном виде. Значение "replace" указывает синтаксическому анализатору на необходимость замены этих символов пробелами. И, наконец, значение "collapse" указывает на то, что все пробельные

[2] Шаблон определяет структуру данных и гарантирует, что данные будут ей соответствовать. С точки зрения данных это выглядит так, как если бы они обладали интеллектом, благодаря которому они поддерживают себя в соответствии со своим жестким форматом, как правило, связанным с их семантикой. – *Примеч. ред.*

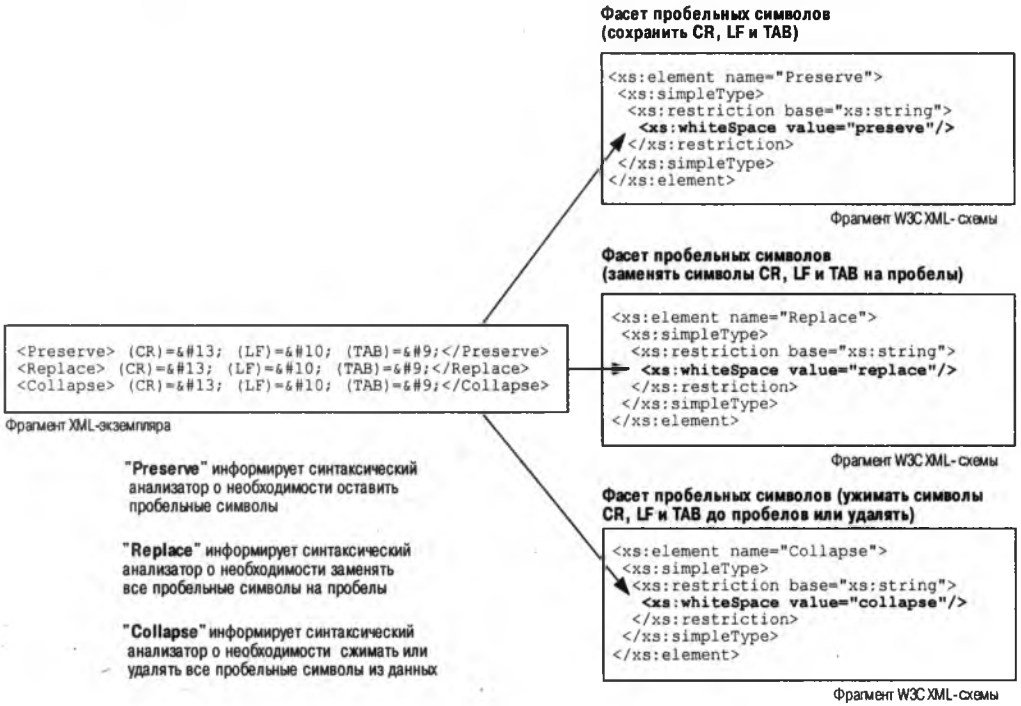


Рис. 5.6. Фасеты пробельных символов

символы (возврат каретки, перевод строки, табуляции и начальные и конечные пробелы) необходимо свернуть или удалить. Важно отметить, что во всех случаях могут существовать разные варианты поддержки этих возможностей со стороны разных синтаксических анализаторов и приложений.

В зоне ответственности за типы данных и относящиеся к ним ограничивающие фасеты базовую роль играет проектировщик данных. Он также отвечает за определение элементов данных, объединение их в группы и задание отношений между этими группами. Во многих отношениях эта деятельность является частью процесса создания моделей данных. Хотя XML не является моделью данных, как мы увидим в следующей главе, ряд аналогичных действий можно применить и к XML.

Глава 6

Структурные модели

Модель данных – это набор определенных и взаимосвязанных информационных фактов и метаданных. Модель данных описывает структуры метаданных, контейнеры и их характеристики, но не касается текущих значений данных. Большинство моделей данных относятся к базам данных либо с реляционной, либо с объектно-реляционной архитектурами, в которых сущность или таблица может иметь отношения с другими сущностями. В отличие от наиболее распространенных моделей данных, структура XML-документа является иерархической, и отношения здесь часто определяются местоположением, или позицией элемента. Если один XML-элемент вложен в другой элемент, подразумевается отношение "родитель-потомок" (т. е. сын, отец, дед, прадед и т. д.). Различия между упомянутыми структурами представления существенны, но преодолимы (возможен переход).

Модели данных часто представляют графически, в виде диаграммы отношений сущностей (entity relationship diagram – ERD), в которой сущности изображают прямоугольниками, внутри каждой сущности перечисляют ее атрибуты, а отношения изображают линиями между сущностями. XML-документ является экземпляром данных (т. е. набором из контейнеров и содержащихся в них значений), представимым иерархически. Хотя большинство синтаксических или литеральных форм лучше всего представлять графически, модель итоговой реализации может представлять и XML-документ в форме прототипа.

Иерархическая структура документа или транзакции XML просматривается сверху вниз, слева направо. Обычно иерархия строится из родительских элементов, которые могут иметь дочерние элементы. Несколько потомков одного родителя, находящихся на одном уровне, называют братьями. Потомки, в свою очередь, также могут иметь потомков, являясь по отношению к ним родителями (рис. 6.1).

Факт. *Все XML-структуры являются иерархическими (и читаются сверху вниз, слева направо).*

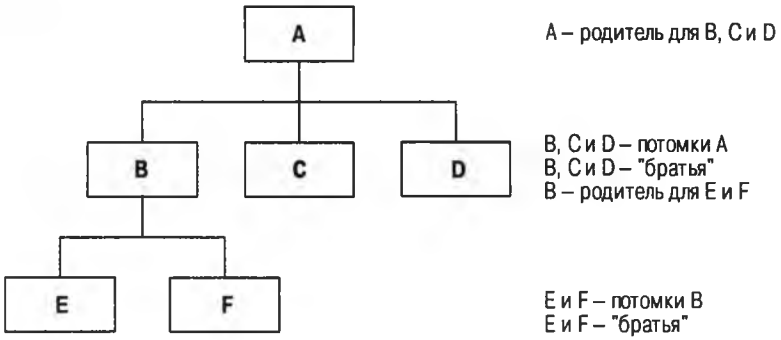


Рис. 6.1. Типичная иерархическая структура

Иерархическая структура XML-документа очевидна. Однако следует понять ряд базовых концепций и правил. Первое: XML-документ является экземпляром. То есть XML-документ обычно создается, заполняется и обрабатывается прикладной программой. XML-документ содержит значения данных. Тогда как W3C XML-Схема является набором правил и ограничений, описывающих XML-документ. Эти концепции аналогичны концепциям реляционных баз данных. Таблица и столбцы базы данных содержат значения данных. Чаще всего эти значения данных формируются и обрабатываются прикладной программой. "Системный каталог" реляционной базы данных описывает структуру (таблицы, столбцы и отношения) и правила базы данных.

Кроме базовых отличий между XML-документом или "экземпляром" данных и W3C XML-Схемой или набором правил и ограничений, следует помнить также несколько структурных и синтаксических правил. Как мы уже знаем, элемент (контейнер) самого верхнего уровня в XML-документе называется корневым (в XML-документе или транзакции может быть только один корневой элемент). Внутри корневого элемента может находиться произвольное количество других элементов и групп элементов. Вложение одних элементов в другие представляет концепцию вложенности. Каждый новый уровень вложенности создает еще один уровень иерархии (как в иерархии родитель-потомки). Элементы могут содержать атрибуты. Некоторые из элементов могут быть определены как пустые и не могут содержать данных. Одни и те же элементы могут повторяться несколько раз, что является рудиментарной формой кардинальности или мультипликативности[1] (рис. 6.2).

[1] Под этим фактически подразумевается возможность неоднократного повторения однаковых ("одноименных") элементов внутри одного элемента-родителя (аналог связи один-ко-многим в, например, ER-модели). – *Примеч. ред.*

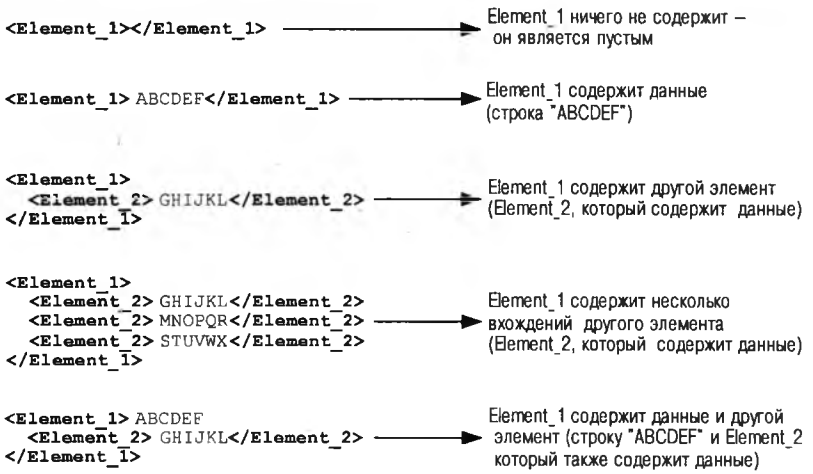


Рис. 6.2. Контейнеры-элементы

XML-атрибуты – что-то вроде другого типа контейнера. Атрибуты должны быть определены для некоторого элемента (т. е. они не могут существовать сами по себе). И элементы, и атрибуты могут содержать значения данных, а могут быть определены как всегда пустые. XML-атрибуты имеют такой же синтаксис, как HTML-атрибуты. XML-атрибут определяется парой "атрибут-значение" (рис. 6.3). Атрибут не может содержать других атрибутов, кроме того, атрибут не обладает кардинальностью или мультипликативностью. То есть если для некоторого элемента определено несколько атрибутов, эти атрибуты не могут повторяться, используя одно и то же имя.

Документы и транзакции XML, подобно традиционным представлениям метаданных (моделям данных и диаграммам классов объектов), также могут представлять собой модели. А именно: прототипы XML-документов, заполненные образцами данных, могут обеспечивать простой и наглядный способ моделирования структуры XML-документов. Когда документ или транзакция XML представляет собой прототип структуры, такая структурная модель наглядно показывает количество и типы контейнеров (элементов и атрибутов). Прототип структуры может использоваться проектировщиком данных в качестве шаблона при разработке соответствующей схемы, а программист может использовать его, чтобы начать работу по созданию кода приложения.

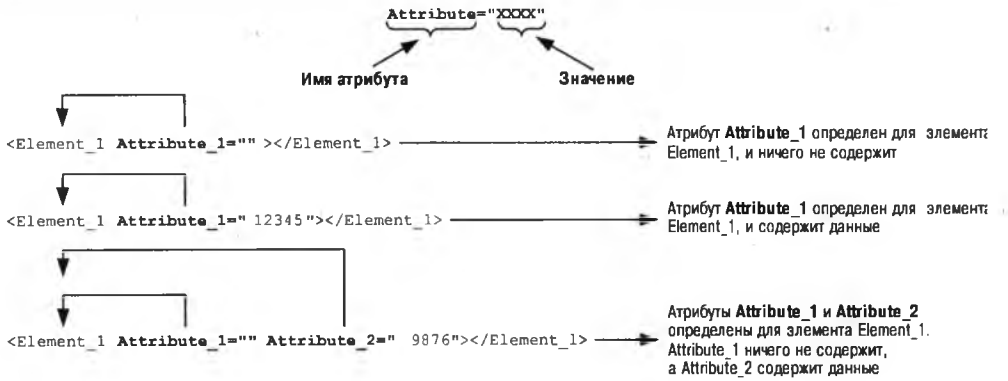


Рис. 6.3. Контейнеры-атрибуты

В некоторых XML-документах используются только контейнеры-элементы. В других – основной упор делается на атрибуты. Во всех остальных используются и элементы и атрибуты. Могут быть некоторые вариации, но к основным структурным моделям XML относятся следующие модели:

- вертикальная;
- горизонтальная;
- компонентная;
- гибридная.

Вертикальная модель

Вертикальная модель – это наиболее часто используемая для построения транзакций и сообщений структура, она отличается преимущественным использованием контейнеров-элементов. Взаимосвязанные или подобные контейнеры-элементы могут быть логически сгруппированы. Термин "вертикальная" описывает иерархическую структуру, присущую почти всем документам и транзакциям XML (сверху вниз, слева направо). В сочетании со вложенными группами контейнеров-элементов, вертикальные модели обычно наиболее интуитивно понятны (рис. 6.4).

Вертикальные структурные модели характеризуются гибкостью (можно спроектировать динамически расширяемую, или приспосабливаемую к контексту содержащихся данных структуру). Такая гибкость является очень важным качеством, если содержащиеся данные могут меняться, она достигается, когда отношения родитель-потомок между контейнерами сочетается с кардинальностью. Примером переменных данных могут служить данные, используемые в глобаль-

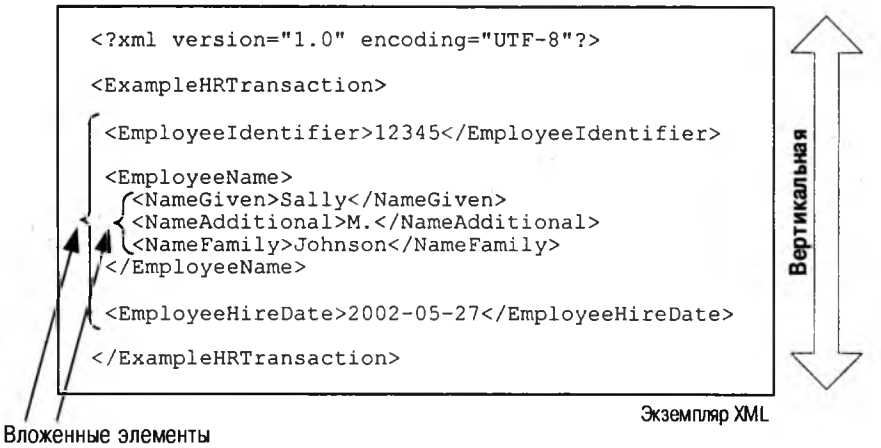


Рис. 6.4. Вертикальная модель структуры XML-документа

ной электронной коммерции. Строки с адресами улиц и другие компоненты международного почтового адреса могут значительно меняться от страны к стране. В Северной Америке почтовый адрес включает 1-2 строки для улицы в сочетании с городом, штатом или округом и почтовым индексом. А вообще, международный почтовый адрес может включать от 1 до 4 строк для улицы, города, различных региональных подуровней (область, район, страна или регион), страны и почтовый индекса. Возможности вертикальной структурной модели по описанию и хранению международных почтовых адресов могут быть с успехом использованы в транзакциях глобальной электронной коммерции (рис. 6.5).

> **Факт.** *Вертикальные структурные модели компонуются преимущественно из контейнеров-элементов. В сочетании с повторением элементов (кардинальность или мультипликативность), вертикальные модели структур XML являются наиболее гибкими. Они могут быть динамически расширены или сжаты в соответствии с характеристиками содержащихся данных.*

При необходимости в дополнительных экземплярах контейнеров-элементов, их можно вставить в соответствующую родительскую группу элементов. Применительно к примеру с международными адресами, если структура адреса диктует необходимость четвертой строки адреса для улицы, в групповой элемент StreetAddressLines можно добавить дополнительный элемент "<StreetAddressLine>". Такой тип структурных изменений обычно можно произвести с минимальным влиянием на ограничивающие схемы и на логику приложения. С точки зрения традиционного моделирования данных, отношению один-ко-многим между сущностями, при котором экземпляры подчиненной сущности могут повторяться, часто присуща гибкость (рис. 6.6).

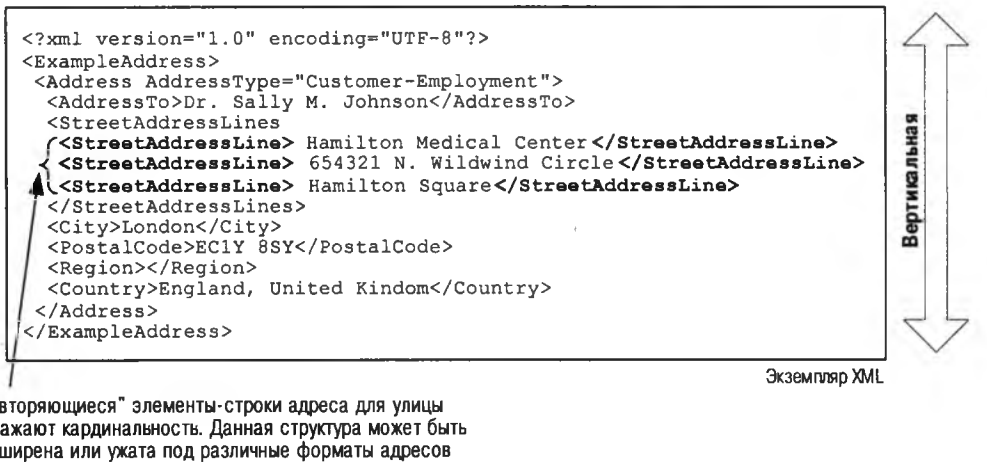


Рис. 6.5. Использование вертикальной структурной модели при описании структуры международного почтового адреса

> **Рекомендация.** *В зависимости от количества контейнеров-элементов, объема содержащихся данных и стандарта именования, используемого для присвоения имен элементам, вертикальные структурные модели могут привести к чрезмерной многословности и существенному размеру документа. Если очень важна производительность приложения, или ограничена пропускная способность канала передачи XML-документа, проектировщику данных придется рассмотреть возможность сжатия, обратиться к другим структурным моделям, или применить другой подход проектирования.*

С преимущественным использованием в вертикальных структурных моделях контейнеров-элементов косвенно связан один их потенциальный недостаток. Поскольку каждый из контейнеров-элементов должен иметь открывающий и закрывающий теги (кроме случаев использования сокращенного синтаксиса тега, когда элемент не содержит данных), вертикальная структурная модель характеризуется большей многословностью, чем другие модели. Суммарный размер XML-транзакции возрастает за счет добавления дополнительных символов от тегов каждого из элементов. Если общий размер транзакции слишком велик или ограничена пропускная способность канала обмена, проектировщику данных можно порекомендовать использовать какую-либо форму сжатия транзакции, или рассмотреть другие структурные модели, или применить иной подход проектирования.

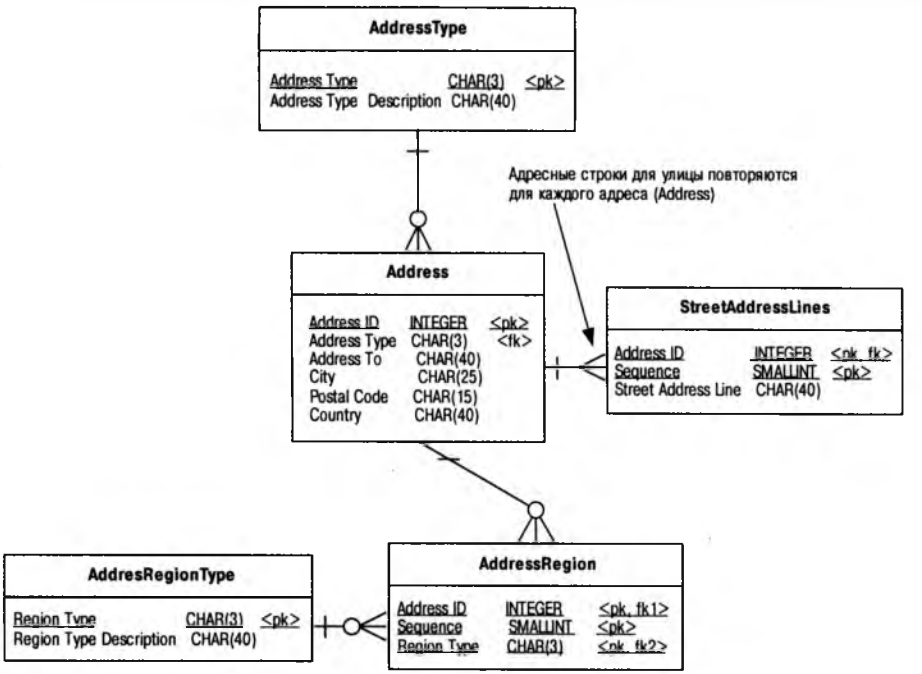


Рис. 6.6. Традиционная модель данных с отношениями один-ко-многим

Горизонтальные модели

Горизонтальные структурные модели менее распространены, чем вертикальные. В горизонтальных моделях для данных используются преимущественно контейнеры-атрибуты, а не контейнеры-элементы. Благодаря этому они менее многословны, чем вертикальные модели, поскольку не так загромождены концевыми тегами. Структура документа, получающаяся в результате их использования, является горизонтальной или "плоской" (рис. 6.7). Большое преимущество горизонтальных структурных моделей – в заметно меньших размерах соответствующих им XML-документов по сравнению с вертикальными моделями, кроме того, они больше соответствуют простым выборкам из реляционной базы данных.

Факт. *В горизонтальных структурных моделях используется значительное количество атрибутов. Поэтому они имеют скорее "горизонтальную" протяженность, нежели вертикальную.*

Чаще всего горизонтальные структурные модели применяются в случаях, когда существенным фактором является общий размер XML-документа или транзакции, или когда пропускная способность канала обмена транзакциями ограничена. Потенциальным местом использования XML-транзакций, чья структура отвечает горизонтальной модели, является перенос данных от традиционных систем к хранилищам данных, или аналогичные условия. В таких случаях очень важна минимизация объемов передаваемых данных.

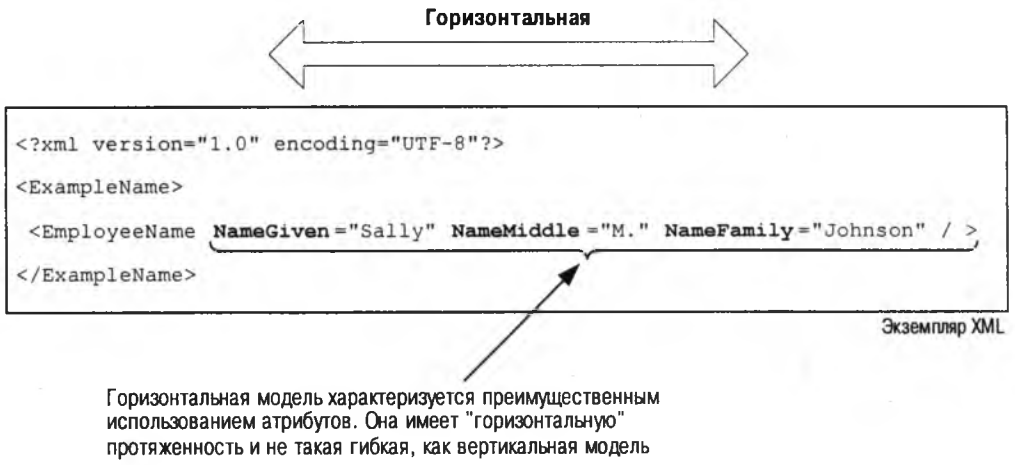


Рис. 6.7. Горизонтальная структурная модель XML

Рекомендация. *Когда суммарный объем XML-документа или транзакции слишком велик или когда пропускная способность канала его передачи ограничена, возможно имеет смысл использовать горизонтальную структурную модель.*

При сравнении вертикальной или горизонтальной структуры XML с традиционными моделями данных можно отметить несколько отличий. Наиболее заметное – это то, что горизонтальные модели не поддерживают неоднократное повторение элементов с одними и теми же именами, известное в моделях данных как кардинальность или мультипликативность. Предположим, некоторая модель данных должна включать сущности и атрибуты сущностей для представления частей полного имени человека. Один из подходов к ее построению – моделировать части полного имени как сущность – полное имя, с отношением один-ко-многим к другой, подчиненной сущности – часть имени.

В сценарии, когда проектировщик данных должен разработать горизонтальную XML-структуру для данных, извлекаемых из реляционной модели, с повторяющимися строками или экземплярами одинаково поименованных данных, каждый такой экземпляр необходимо размещать в контейнере-атрибуте с уникальным именем. Необходимо использовать трансляцию повторяющихся элементов, отдельно по каждому из возможных видов или контекстов, в контейнеры-атрибуты с тем же содержимым (рис. 6.8).

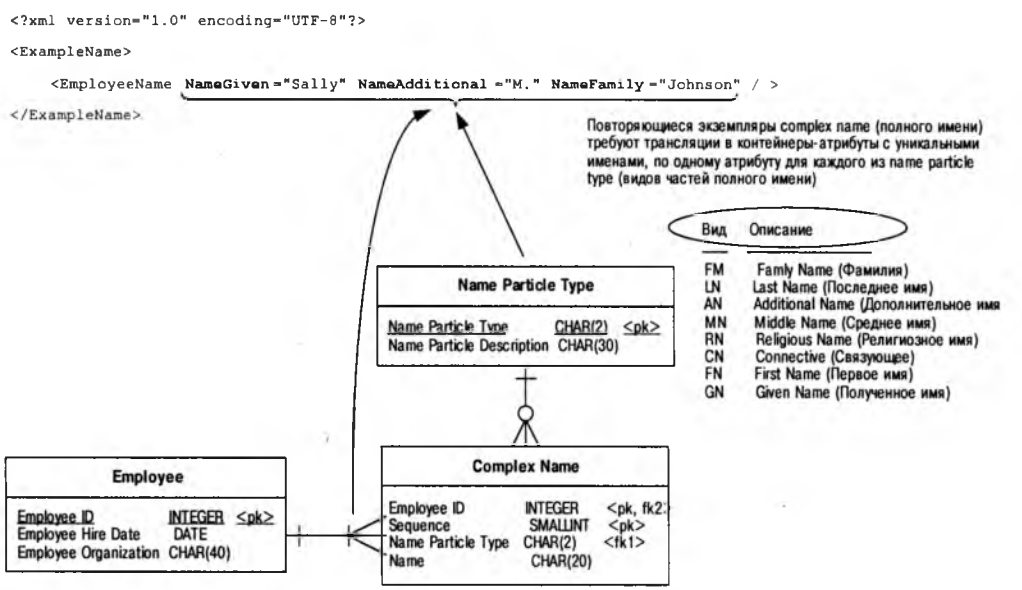


Рис. 6.8. Трансляция сущности с отношением один-ко-многим в горизонтальную модель XML-структуры

Недостатком горизонтальных моделей XML-структуры является отсутствие гибкости. Из-за того что невозможно повторение атрибута с одним и тем же именем в пределах контейнера-элемента, для которого атрибут определен, атрибуты должны определяться как жестко заданный набор уникально описанных контейнеров. В XML-документ невозможно динамически вставить дополнительные экземпляры атрибутов с теми же именами. И если возникнет необходимость в дополнительных данных, то в XML-документе и в схемах, на которые он ссылается, потребуется доопределить один или несколько новых атрибутов с уникальными именами. А это потенциально грозит внесением изменений и в приложение.

Компонентные модели

Одной из серьезнейших задач, стоящих перед проектировщиком данных, является обеспечение возможности повторного использования модели данных или структуры метаданных. Во многих случаях модель данных состоит из большого числа сущностей, и повторное использование означает добавление отношения к (или от) еще одной новой сущности. В случае расширенного повторного использования этот подход требует введения отношений к сущностям из других моделей (рис. 6.9). В результате, структуру, представляющую такую сущность, можно использовать повторно, но, главным образом, при реализации и нисходящей разработке (например, для генерации синтаксиса, наподобие DDL[2], при реализации для базы данных).

Возможность повторного использования концепции данных или сущности в других моделях данных и в других структурах данных часто ограничивается применяемым инструментом проектирования или моделирования данных. Некоторые из инструментов моделирования данных поддерживают ссылки или импорт сущностей из других моделей. Однако многие инструменты моделирования не предоставляют таких возможностей, и процесс повторного использования может оказаться столь же неудобным, как копирование и вставка сущностей. Процесс копирования-вставки в лучшем случае малоэффективен и ведет к размножению дублирующих сущностей, вместо определения корпоративных стандартов метаданных и структур, которые определяются всего один раз, находятся под присмотром в одном месте и используются повторно с помощью ссылки.

В качестве альтернативы процессу копирования-вставки, W3C XML-Схемы предоставляют возможность широкого повторного использования в форме компонентных структурных моделей. Компонентные структурные модели основаны на концепции повторного использования. Суть компонентной структурной модели в том, что XML-документ складывается из модульных групп или наборов взаимосвязанных контейнеров с данными (рис. 6.10). На абстрактном уровне, каждая из модульных групп контейнеров определяется в отдельной схеме. В зависимости от уровня абстракции имен элементов и атрибутов XML, существует возможность повторного использования схем с определениями модульных групп контейнеров во многих других контекстах. К примеру, схему, описывающую модульную группу элементов для "Person Name" (полного имени человека), можно использо-

[2] Data Definition Language – подмножество SQL, в которое входят команды, связанные с созданием и модификацией объектов баз данных (CREATE, ALTER, DROP). – *Примеч. ред.*

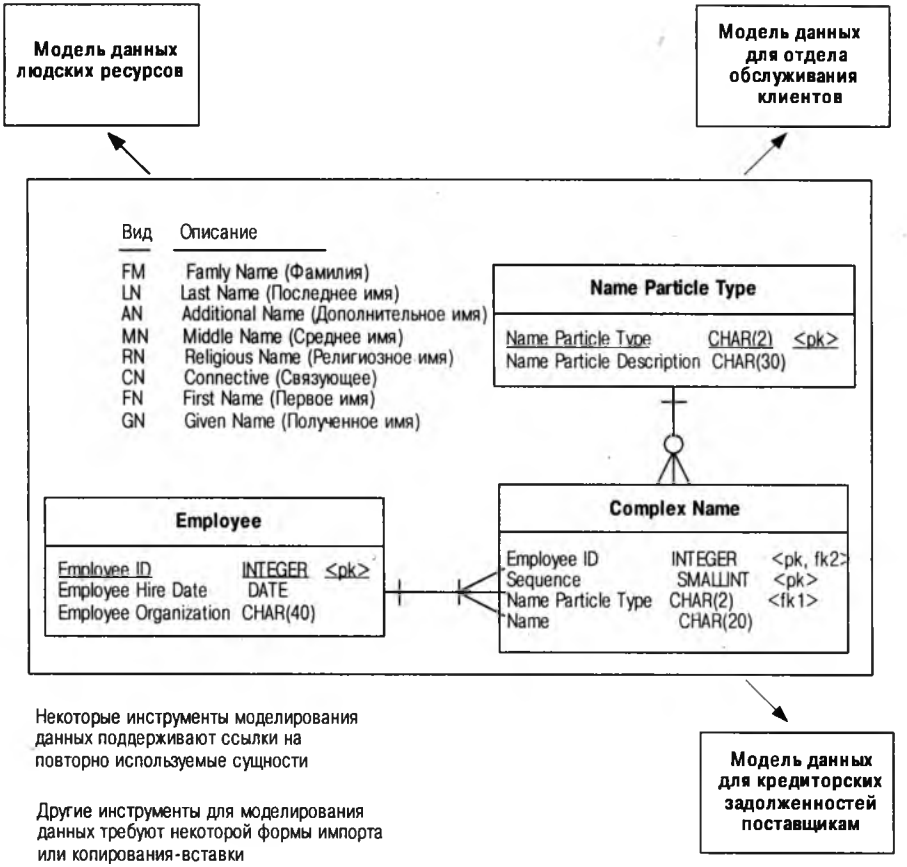


Рис. 6.9. Повторное использование сущности базы данных в других моделях и контекстах

вать в контекстах имени служащего, имени покупателя или имени представителя поставщика. В результате достигается снижение затрат на проектирование и разработку (на величину стоимости разработки новой схемы) при каждом повторном использовании такой стандартной структуры.

Факт. *Использование компонентных структурных моделей обеспечивает хорошие предпосылки для повторного использования. Применяя внешние подсхемы, которые можно, с помощью ссылок, повторно использовать в разных схемах, получаем высоко стандартизированные и модульные по конструкции группы контейнеров.*

Рис. 6.10. Модульные группы взаимосвязанных контейнеров с данными

При применении повторно используемых схем-компонентов в W3C XML-Схемы, описывающие весь XML-документ в целом, помещают ссылки на каждую из этих модульных подсхем. В процессе верификации синтаксический анализатор разрешает все ссылки на модульные подсхемы и включает в схему адресуемые ссылками контейнеры, типы и т. п. Таким образом, общая W3C XML-Схема, или *мастер-схема*, собирается из модульных подсхем. Преимущество такой структурной модели XML заключается в возможности определить высокоструктурированные и модульные XML-структуры, которые можно использовать повторно в разных контекстах, разными XML-схемами.

Хотя подобные возможности ссылаться на определенные вовне структуры поддерживаются большинством типов схем, W3C XML-Схемы предоставляют несколько дополнительных опций. Синтаксис XML-схемы для подключения внешних подсхем возможно наиболее прост, поскольку происходит просто подстановка контейнеров и структур, на которые имеются ссылки (рис. 6.11). Использование синтаксиса подключения XML-схемы является основой повторного использования, он будет рассмотрен в главе 8.

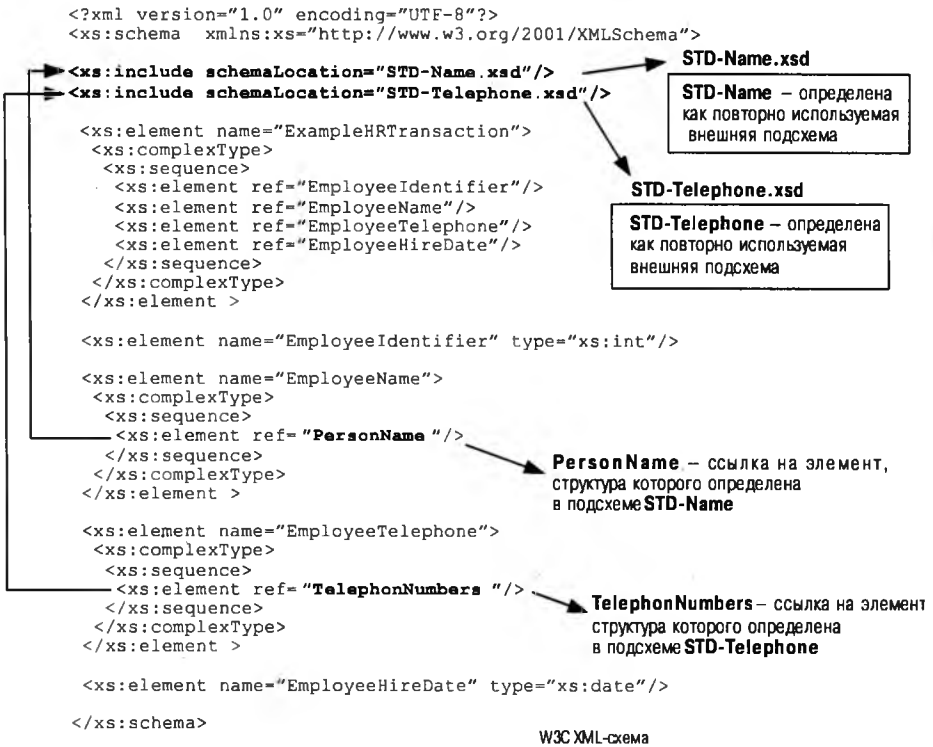


Рис. 6.11. Компонентная структурная модель XML, применяемая к W3C XML-Схеме

Рекомендация. *Наиболее эффективно использовать XML-атрибуты в качестве контейнеров для порядкового номера элемента в последовательности повторяющихся элементов, для описательной классификации элемента, для значений стандартных кодов, описания функции или вида деятельности и для хранения отдельных частей элемента, содержащего данные.*

Гибридные модели

Гибридные модели являются наиболее предпочтительным способом организации структуры для большинства данных, ориентированных на транзакции. Как следует из названия, гибридные модели сочетают в себе наиболее удачные характеристики трех других структурных моделей (вертикальной, горизонтальной и компонентной). Вертикальные модели характеризуются преимущественным

использованием контейнеров-элементов и гибкостью архитектуры. Горизонтальные модели используют атрибуты для наиболее сжатого описания характеристик содержащихся данных, а компонентные модели фокусируются на модульности подсхем для обеспечения возможности повторного использования (рис. 6.12).

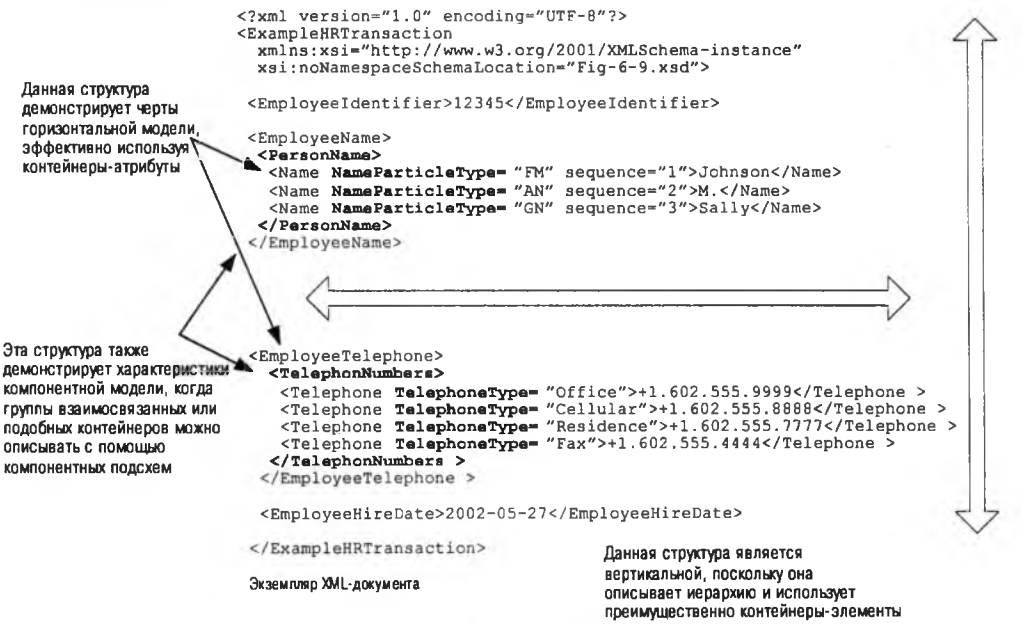


Рис. 6.12. Применение гибридной модели для XML-транзакции

Подобно вертикальной структурной модели, гибридная модель использует в качестве контейнеров для данных элементы. Гибридные модели также эффективно используют и контейнеры-атрибуты. Применение атрибутов ограничивается описанием порядка следования повторяющихся элементов (т. е. атрибутами с именами типа "sequence" (положение), "line" (строка), "number" (номер)), кроме этого, атрибуты используются как метод классификации элементов (атрибуты "type" (тип), "class" (класс)), а также для представления функции или вида деятельности (для функций транзакции, таких, как "add" (добавить), "delete" (удалить)). В некоторых случаях атрибуты используются для представления отдельных частей значения данных (например, фамилии, имени, отчества, как частей полного имени). Атрибуты также можно использовать для размещения значений стандартных кодов (рис. 6.13).

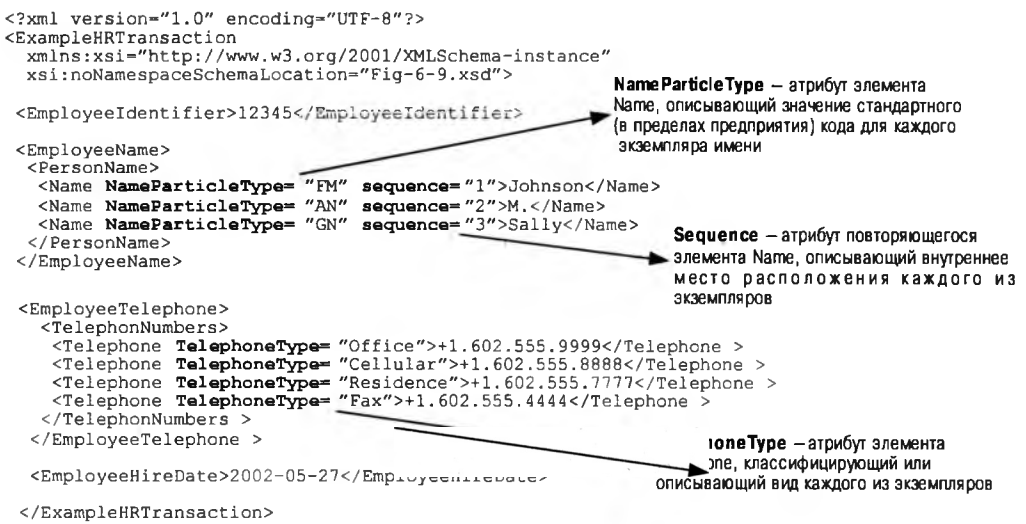


Рис. 6.13. Примеры использования контейнеров-атрибутов

Факт. *Гибридные структурные модели сочетают характеристики трех других структурных моделей (вертикальной, горизонтальной и композитной). При правильном использовании гибридные модели приводят к созданию гибких, хорошо определенных и повторно используемых XML-структур.*

Гибридные модели рекомендуются в качестве структурных моделей для большинства XML-документов и транзакций. В них усиливаются сильные стороны других моделей и снимаются присущие этим моделям ограничения. Хотя структурные модели обычно применяются к прототипам XML-документов, эффективная разработка соответствующей схемы требует использования шаблона метаданных. В случае XML в качестве шаблонов выступают архитектурные формы контейнеров.

Глава 7

Архитектурные формы контейнеров

Архитектурная форма контейнеров – это обобщение заметного, повторяющегося шаблона, применяемого в структурной модели. Шаблоны – в чем-то спорная тема. Шаблоны – это заслуживающие внимания образцы или структуры, которые можно неоднократно применять в их обычной форме, или немного изменив. Концептуально шаблоны, применяемые в XML и в других информационных технологиях, имеют сходство с шаблонами, используемыми при проектировании архитектуры зданий. В качестве примера из области проектирования зданий и структурной разработки можно рассмотреть в качестве шаблона дверь. Двери являются неотъемлемой частью почти любого здания. В наиболее общей форме дверь – это просто проход, позволяющий людям, животным, предметам, воздуху и свету проникать в здание и покидать его. Она может находиться в одном из двух взаимоисключающих состояний: открыта или закрыта. Данное определение двери, как части структуры здания, вполне работоспособно.

Кроме этого, существуют варианты использования или назначения двери. В доме дверь может открываться внутрь или наружу и находиться на уровне земли. А на складе дверь может открываться снизу вверх, иметь гораздо большие размеры и находиться на высоте погрузки. Можете себе представить, сколько разных вариантов двери может существовать. Шаблон, представляющий дверь, вполне понятен и может использоваться в разных контекстах. Каждый новый контекст подразумевает новое применение данного шаблона, как архитектурной формы.

Применительно к XML, архитектурные формы контейнеров представляют собой нечто аналогичное. Структурные модели XML, описанные в предшествующей главе (вертикальная, горизонтальная, компонентная и гибридная), являются моделями структуры транзакций. Разработка прототипа XML-документа сначала ведется в соответствии с одной или несколькими из этих структурных моделей.

Архитектурные формы контейнеров, в сочетании с характеристиками содержащихся данных, и назначением или характером использования XML-документа, дополняют структурную модель документа или транзакции, подгоняя ее к одному или нескольким контекстам.

К примеру, документ-прототип, описывающий адресные данные для доставки заказов, также может быть использован для почтовой рассылки рекламных материалов. Элементы, в которых содержатся адресные данные, а также связанная с ними обработка, могут отличаться от приложения к приложению. Во многих отношениях адаптация архитектурных форм контейнеров напоминает традиционные процессы проектирования и анализа, расширяющие простую модель данных до высоко стандартизированной и потенциально повторно используемой архитектуры данных.

В XML-документе можно использовать три основных архитектурных формы контейнера:

1. жесткую;
2. абстрактную;
3. гибридную.

Каждая из форм обладает одной или более положительными характеристиками, а в гибридной форме эти характеристики объединяются. Задача проектировщика данных – определить, какую из архитектурных форм контейнера использовать в конкретной структурной модели XML. Адаптация архитектурных форм контейнеров к структурной модели XML – процесс, требующий архитектурного подхода, а не заученный набор шагов. Возвращаясь к аналогии с дверью, проектировщик здания должен определить характеристики здания, способ использования двери (в настоящее время и в будущем), применяемые стандартные конструкции и строительные коды, также необходимо учитывать возможные структурные изменения. Подобно этому и проектировщик данных должен думать шире, чем просто первоначальное использование XML-транзакции и традиционная конструкция таблиц базы данных. Для выбора требуемой формы контейнера проектировщик данных должен учесть ряд важных критериев.

- Степень, в которой итоговый XML-документ будет отвечать текущим информационным потребностям.
- Направление, в котором XML-документ будет эволюционировать (с учетом текущей поддержки).
- Изменчивость содержащихся в документе данных.
- Сложность XML-документа (т. е. равномерное вложение, количество вложенных элементов, глубину вложенности и контейнеры с производными или агрегатными данными).

- Согласование документа с международными, отраслевыми и корпоративными стандартами.
- Согласование и интеграция документа с архитектурой данных источника и получателя.
- Возможные изменения в использовании, обработке, объемах и производительности.
- Простота навигации по XML-документу.
- Возможность повторного использования в таком же контексте или в других контекстах.

Каждый из перечисленных критериев может значительно повлиять на то, насколько XML-документ или транзакция будет отвечать настоящим или будущим требованиям. Архитектурные формы контейнера расширяют архитектурную модель XML (первоначально представленную прототипом XML-документа), являясь шаблоном для разработки высоко стандартизированной, гибкой, расширяемой и повторно используемой W3C XML-схемы.

Жесткие формы контейнеров

Как видно из имени, жесткая форма контейнера – это адаптация статичного или фиксированного шаблона. При адаптации жесткой формы контейнера, контейнеры-элементы получают конкретизированные имена на основе строгого стандарта именования. В этом случае контейнерам-элементам нельзя давать абстрактные имена и эти контейнеры, как правило, не должны повторяться (т. е. применение кардинальности ограничено). Пользуясь примером с полным именем человека и адаптируя жесткую форму контейнера к вертикальной структурной модели, каждый контейнер-элемент получит конкретизированное и уникальное имя, сужающее контекст использования содержащихся в нем значений (листинг 7.1).

Листинг 7.1 Применение жесткой формы контейнера

```
<?xml version="1.0" encoding="UTF-8"?>
<PersonName>
  <NameGiven>Sally</NameGiven>
  <NameMiddle>M.</NameMiddle>
  <NameFamily>Smith</NameFamily>
</PersonName>
```

На листинге 7.1 показан прототип XML-документа, адаптирующего жесткую форму контейнера к вертикальной структурной модели. XML-структура первоначально складывается из контейнеров-элементов, расположенных сверху вниз (вертикальная модель). Каждый контейнер-элемент имеет конкретизированное имя,

однозначно описывающее содержащиеся в нем данные (что объясняет название "жесткая"). В этом примере родительский элемент "<PersonName>" обеспечивает общую классификацию или контекст для вложенных в него элементов. Как следует из имени тега родительского элемента, дочерние элементы представляют части полного имени человека. Дочерний элемент "<NameGiven>" (данное имя) предназначен для данного человеку имени, его первого имени или прозвища. Аналогично, дочерний элемент "<NameFamily>" (фамилия) предназначен для фамилии.

Рекомендация. *Жесткие архитектурные формы контейнера рекомендуется применять для XML-документов, используемых для обмена информацией с внешними для предприятия сотрудничающими группами, а также для определения интерфейса Web-сервиса. Во избежание ошибочной интерпретации, внешние сущности обычно требуют использования стандарта именования создающего конкретизированные описательные имена.*

Документ с полным именем человека отличается применением строгого стандарта именования к именам элементов. Одним из преимуществ жесткой формы является то, что контейнеры получают описательные имена, очень хорошо описывающие содержащиеся в них данные. Подобно атрибутам отдельной сущности модели данных, или столбцам реляционной таблицы базы данных, каждый элемент данных имеет уникальное имя. Таким образом, установить содержимое каждого элемента нетрудно. Также для жесткой формы характерно то, что ни один из элементов не появляется дважды внутри одного и того же родительского элемента. При использовании жесткой формы контейнера в вертикальной модели получающаяся структура проста и понятна (рис. 7.1).

Жесткие формы контейнера легко использовать в любой структурной модели XML (вертикальной, горизонтальной, гибридной или компонентной), они хорошо подходят для транзакций обмена с внешними сотрудничающими группами. Использование XML сотрудничающими группами, например сообществом торговых партнеров, диктует необходимость создания согласованного общего словаря. XML-транзакции, курсирующие между участниками таких групп, будут содержать значения данных, отвечающие ожиданиям и отправителя, и получателя. Жесткие формы контейнера с конкретизированными описательными именами контейнеров элементов часто используются для того, чтобы избежать неясности и, следовательно, предпосылок для ошибок.

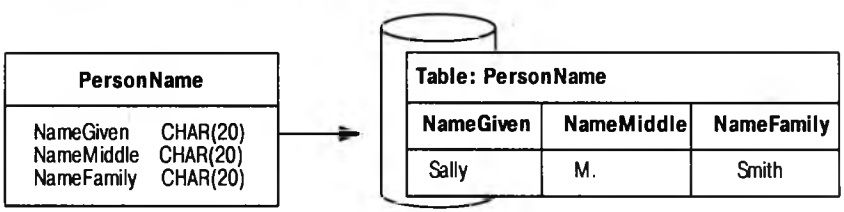


Рис. 7.1. Пример модели данных, соответствующей жесткой форме контейнера

Жесткие формы контейнера также полезны для определения интерфейса Web-сервиса. В хорошо определенном Web-сервисе интерфейс явно описан в минисловаре. Этот минисловарь может быть записан с использованием языка описания Web-сервисов (Web Services Description Language – WSDL), и, как и в сценарии с сотрудничающими группами, жесткая форма контейнера помогает избежать ошибочной интерпретации данных. Обращающийся к Web-сервису, основываясь на контексте и схеме интерфейса сервиса, определяет, какие данные необходимо включить в запрос к сервису, а также структуру ответа. Подробнее о Web-сервисах мы поговорим в главе 10.

Факт. *Жесткие архитектурные формы контейнера могут упростить навигацию и обработку структуры XML-документа.*

Другим преимуществом жестких форм контейнеров является их согласованность с навигационными методами большинства синтаксических анализаторов. Как вы помните, XML-документ имеет иерархическую структуру. Каждый из контейнеров-элементов синтаксический анализатор представляет как узел в иерархии (существует несколько других видов узлов, но нас сейчас интересует именно элементы). Синтаксические анализаторы также предоставляют в распоряжение прикладных программ *методы*, являющиеся программными функциями или API, которые позволяют прикладной программе ориентироваться и перемещаться по структуре XML-документа.

При использовании в XML-документе жестких форм контейнера, предельно точные имена контейнеров упрощают навигацию по XML-документу. Большинство DOM-синтаксических анализаторов поддерживает метод “getElementsByTagName” (“tagNameArgument”). Будучи вызван, этот метод возвращает список всех элементов XML-документа, или список элементов, чье имя соответствует имени тега, заданному в качестве аргумента. Прикладная программа имеет возможность “искать” определенные элементы по именам, вместо того, чтобы путешествовать по всему документу, сверяя имя каждого из элементов с именем элемента, который требуется обработать. Навигация по XML-документу упрощается, поскольку прикладная программа может напрямую получить необходимый для обработки элемент. Благодаря тому, что каждый элемент имеет уникальное имя и отсутствует повторение элементов, не требуется дополнительная логика для перемещения по повторяющимся элементам-братьям (элементам с одним и тем же именем, расположенным внутри одного родительского элемента) и проведения необходимой обработки. Что касается SAX-синтаксических анализаторов, то хотя навигация по документу с их помощью осуществляется с помощью событий, связанных с узлами, существует интерфейс “DocumentHandler”, обеспечивающий похожий способ достижения узлов по имени.

W3C XML-Схема для жесткой формы контейнера довольно прямолинейна. Элементы объявляются с однозначными именами, и элементы не повторяются в пределах содержащих их родительских элементов. При адаптации жесткой формы контейнера в горизонтальной или гибридной структурной модели, помимо элементов определяются и атрибуты. Типы данных для элементов и атрибутов назначаются с помощью атрибута "type" W3C XML-схемы, или используя "simpleType" (рис. 7.2).

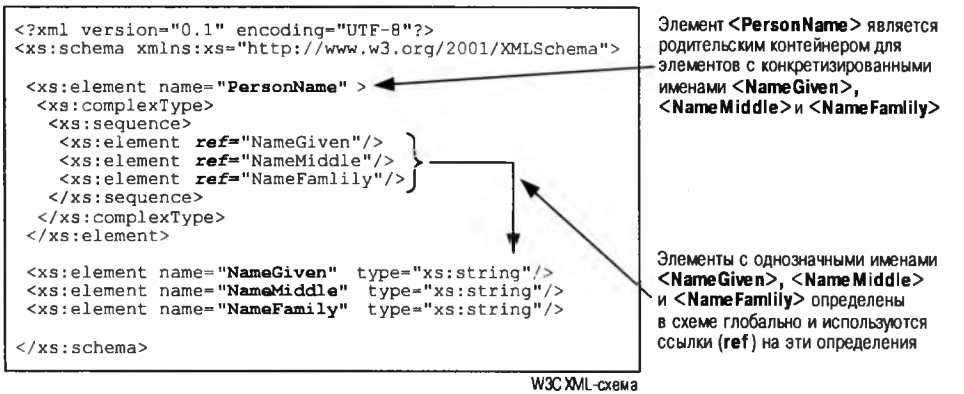


Рис. 7.2. Простая W3C XML-Схема для жесткой формы контейнера

Факт. *Жесткие архитектурные формы контейнера, как правило, не отличаются гибкостью. Добавление новых контейнеров с уникальными именами обычно требует изменения структуры XML-документа, соответствующих схем и, возможно, прикладных программ, обрабатывающих документ.*

Жесткие формы контейнера имеют и недостатки. В них редко применяется повторение элементов с одинаковыми именами и заданной кардинальностью. В результате, XML-документы, использующие жесткие формы контейнеров, не обладают гибкостью. Их нельзя динамически расширить или сжать, обеспечив поддержку кардинальности для элементов. Если потребуются дополнительные элементы, будет добавлен новый контейнер-элемент со своим собственным именем. А это повлечет за собой изменение соответствующих XML-схем и, возможно, прикладной программы, обрабатывающей документ.

Абстрактные формы контейнера

Абстрактные формы контейнера – адаптация шаблона, включающего изменяемые и повторяющиеся данные. Такой шаблон наблюдается, когда набор элементов данных применяется несколько раз в одном, или в похожем, контексте (кардинальность). Примером такого шаблона является почтовый адрес с несколькими адресны-

ми строками для улицы. Существует много форматов почтовых адресов, каждый из которых может требовать своего количества адресных строк для улицы, а также прочих компонентов адреса. С этим часто приходится сталкиваться в глобальной электронной коммерции, и это может вызвать определенные трудности.

Рекомендация. *Абстрактные формы контейнера рекомендуется применять для структур, в которых присутствует шаблон данных, несколько раз повторяющийся в одном или в похожем контексте (например, строки для улицы в почтовом адресе, части полного имени человека и телефонные номера разных видов).*

Отдельные проектировщики данных могут возразить, что в примере с международным почтовым адресом можно использовать жесткую форму контейнера. Однако рассмотрим тот факт, что международный адрес требует от 1 до 4 строк для улицы. Адаптация жесткой формы контейнера потребует определения четырех контейнеров для этих строк, каждый с уникальным именем (например, <StreetAddressLine1>, <StreetAddressLine2>, <StreetAddressLine3> и т. д.). Также из-за того, что количество адресных строк, необходимых для разных международных адресов, может различаться, каждый из элементов для адресной строки требуется определить как опциональный, а не как обязательный. Это поможет избежать внесения в XML-документ лишних, пустых, контейнеров-элементов (листинг 7.2).

Листинг 7.2 Применение жесткой формы контейнера для почтового адреса

```
<?xml version="1.0" encoding="UTF-8"?>
<PostalAddress>
  <AddressTo>Sally M. Smith</AddressTo>
  <StreetAddressLine1>Harcourt House</StreetAddressLine1>
  <StreetAddressLine2>50 Sheffield Place</StreetAddressLine2>
  <AddressCity>London</AddressCity>
  <AddressPostalCode>EC3Y-9SY</AddressPostalCode>
  <AddressCountry>England, United Kingdom</AddressCountry>
</PostalAddress >
```

Подобно единственной таблице базы данных, содержащей столбцы для всего адреса, XML-документ с жесткой формой контейнеров, пытающийся одной структурой охватить все возможные форматы международных почтовых адресов, имеет очень негибкую архитектурную форму. Аналогичная таблица базы данных должна включать 4 поля для всех четырех возможных адресных строк, несмотря на то что реально могут быть заполнены данными лишь некоторые из этих полей, а остальные останутся пустыми (или “null”) (рис. 7.3). Приложение должно идентифицировать, отображать и вставлять данные в отдельные экземпляры, пользуясь именем элемента или другим подобным способом. Если проектировщик не знал о необходимости иметь именно 4 адресных строки и предусмотрел всего 2 или 3 элемента, последующее добавление четвертого элемента потребует внесения изменений логику прикладной программы, используемую для заполнения этих элементов, а также изменения схем.

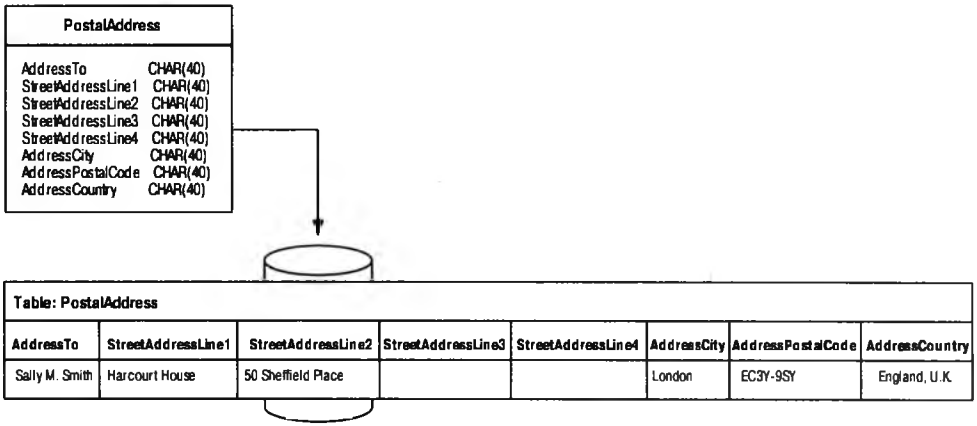


Рис. 7.3. Пример модели данных для единой таблицы с адресом, соответствующей жесткой форме контейнера

Факт. *Абстрактные формы контейнера основываются на концепциях повторения и кардинальности.*

Абстрактные формы контейнера представляют значительно более гибкий подход, адаптирующий шаблоны повторяющихся данных. Абстрактная форма контейнера использует возможности XML по поддержке повторяющихся контейнеров-элементов и кардинальности. Вместо того чтобы представлять международный почтовый адрес как набор элементов с уникальными именами, предлагается структура, включающая группу повторяющихся элементов под адресные строки, где каждый из элементов группы имеет одно и то же имя (листинг 7.3). Если потребуется включить в эту структуру дополнительную строку, для поддержки еще одного адреса, XML-документ может быть расширен динамически, с минимальным (или вообще без) воздействием на определения существующей схемы или прикладную логику.

Техника. *Абстрактные формы контейнера используют, до некоторой степени, абстрактные имена элементов. Один из основополагающих принципов абстрактной формы контейнера - имена элементов должны, с одной стороны, достаточно хорошо описывать содержимое элементов, а, с другой стороны, должны быть достаточно абстрактными, для того, чтобы было возможно повторение. Если же все еще остается неясность относительно содержимого повторяющихся элементов, родительский элемент повторяющейся группы должен обеспечивать дополнительный контекст.*

При использовании абстрактной формы контейнера некоторая степень определенности удаляется из имени повторяющихся элементов (в соответствии с термином "абстрактный"). Некоторые из проектировщиков данных могут возразить, что абстрактное именование нарушает концепцию строгого стандарта именования

и ведет к разрастанию XML-документа, что способствует возникновению неоднозначности и ошибок. Если процесс именования элементов проходит без плана или ограничений, этот аргумент можно признать справедливым, однако одним из неотъемлемых принципов абстрактной формы контейнера является то, что несмотря на абстрактность, имя контейнера должно иметь достаточную степень определенности, чтобы человек все еще мог определить, что содержится в элементе. Слишком абстрактные имена нарушают принцип самоописываемости. В то же время слишком конкретные имена снижают возможности повторного использования.

Листинг 7.3 Применение абстрактной формы контейнера для почтового адреса

```
<?xml version="1.0" encoding="UTF-8"?>
<PostalAddress>
  <AddressTo>Sally M. Smith</AddressTo>
  <AddressLines>
    <StreetAddressLine>Harcourt House</StreetAddressLine>
    <StreetAddressLine>50 Sheffield Place</StreetAddressLine>
  </AddressLines>
  <AddressCity>London</AddressCity>
  <AddressPostalCode>EC3Y-9SY</AddressPostalCode>
  <AddressCountry>England, United Kingdom</AddressCountry>
</PostalAddress >
```

Если требуется добавить контекст (например, роль или классификацию), его можно обеспечить с помощью родительского элемента группы. К примеру, в случае международного почтового адреса, повторяющийся элемент с именем “<Line>”(строка), вместо “<StreetAddressLine>”(строки адреса для улицы), имеет слишком абстрактное имя. Элемент <Line>, если судить по имени, может иметь почти любое содержимое. Разработчик приложения, сбитый с толку этим именем, может заложить в программу некорректную логику обработки этого элемента.

Факт. *Абстрактные формы контейнера часто включают атрибут для описания порядкового номера повторяющегося элемента в группе таких элементов.*

Другим важным аспектом абстрактной формы контейнера являются атрибуты. Абстрактные имена повторяющихся элементов могут породить неясность. И потребуется добавить в элемент некоторые дополнительные характеристики, например, порядковый номер элемента в последовательности повторяющихся элементов. В вертикальной структурной модели с абстрактной формой контейнера, для описания порядкового номера в повторяющихся элементах часто используются атрибуты. Как мы увидим позднее, атрибуты также успешно используются в гибридной форме контейнера. Однако использование атрибутов должно ограничиваться описанием следующих характеристик:

- порядковый номер;
- тип или классификация;

- стандартные коды;
- закрепленная функция или вид деятельности;
- отдельные части значения данных.

Рекомендация. *За исключением горизонтальных структурных моделей, рекомендуется использовать атрибуты только для указания порядкового номера, типа или классификации, стандартных кодов, закрепленной функции или вида деятельности, отдельных частей значения данных.*

Повторяющиеся элементы располагаются в логическом порядке, в соответствии с иерархией (сверху вниз, слева направо), а их определения, используемые при верификации, задаются в схеме, в "group" или "complexType". Однако могут быть случаи, когда фактический порядок следования элементов в иерархии не отвечает их назначению или когда необходимо выборочно обработать только некоторые экземпляры из повторяющихся элементов, а не все подряд. В таких случаях в определение повторяющихся элементов добавляют "порядковый" атрибут (номер элемента в последовательности), описывающий требуемое внутреннее положение элемента среди повторяющихся элементов, и соответствующим образом заполняют его (рис. 7.4) Обрабатывающее приложение извлекает порядковый атрибут и по нему определяет, какая обработка нужна для данного элемента, обычная или специальная.

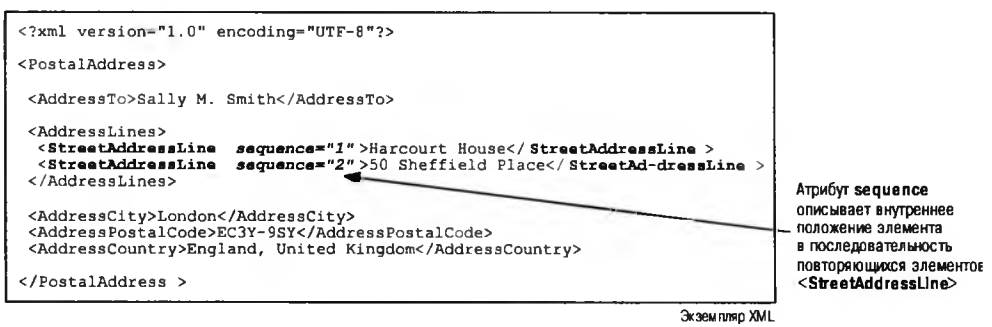


Рис. 7.4. Абстрактная форма контейнера с атрибутом для порядкового номера

Хотя логическая модель данных не может полностью представлять XML-структуру, с ее помощью все же можно создать примерное представление абстрактной формы контейнера (рис. 7.5). Сущности логической модели представляют родительские или групповые контейнеры-элементы. Повторяющиеся элементы определяются с кардинальностью, представляемой как "один-ко-многим". Единственное отличие заключается в том, что логические модели данных поддерживают для сущности только атрибуты сущности с уникальными именами.

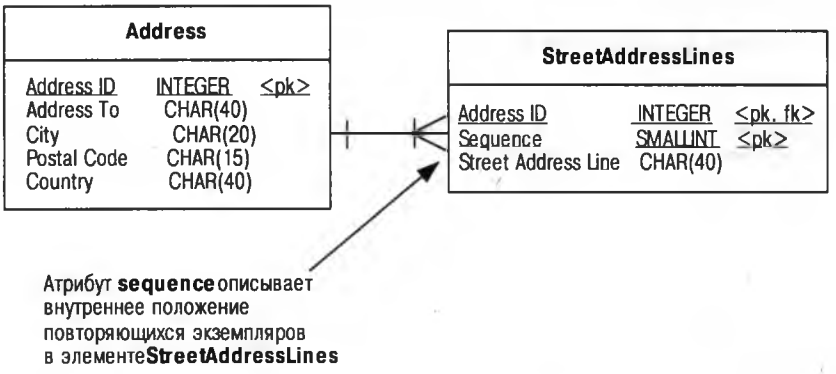
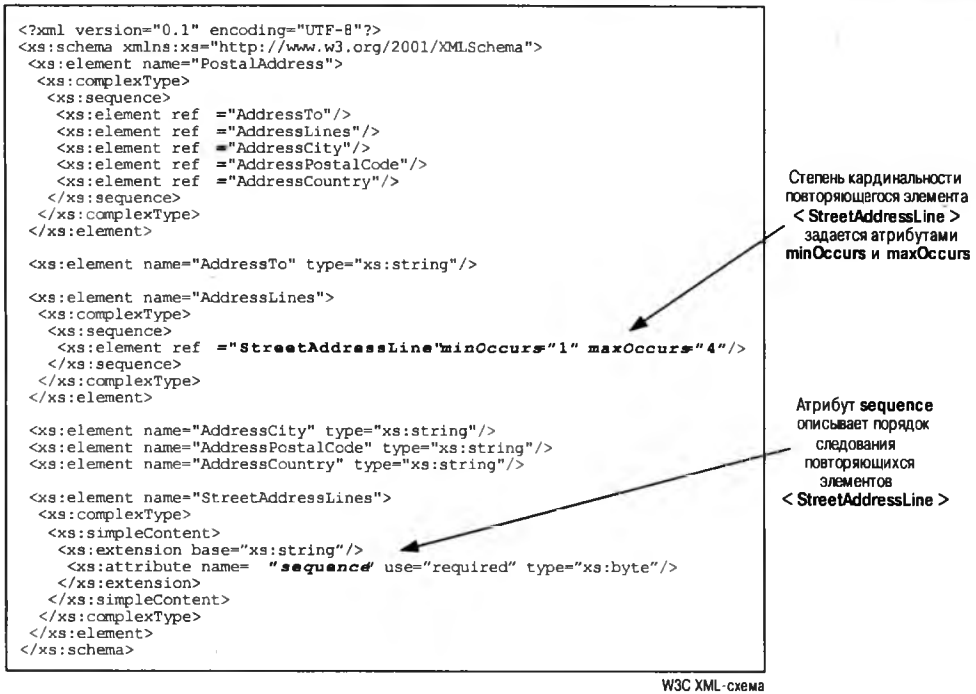


Рис. 7.5. Модель данных, представляющая архитектурную форму контейнера с атрибутом для порядкового номера

Схема для абстрактной формы контейнера должна отвечать либо вертикальной, либо гибридной структурной модели, которые складываются преимущественно из элементов. Повторяющиеся элементы определяются как дочерние элементы из группы родительского элемента. В синтаксисе W3C XML-Схемы для определения кардинальности повторяющихся элементов используются атрибуты minOccurs и maxOccurs (рис. 7.6). Концептуально, *степень* кардинальности описывает минимальное и максимальное допустимое количество экземпляров повторяющихся данных. Если степень кардинальности неизвестна, или не может быть определена, минимальная и максимальная границы бесконечны. Применительно к W3C XML-Схеме, атрибут minOccurs будет установлен в ноль, а атрибут maxOccurs – в "unbounded"(т. е. minOccurs=0, а maxOccurs="unbounded").

Если требуется установить определенные значения для минимальной и максимальной степени кардинальности, эти значения синтаксически определяются с помощью атрибутов minOccers и maxOccurs W3C XML-Схемы. Требуется осторожность при задании для повторяющихся элементов значений атрибутам minOccers и maxOccurs. Эти значения являются ограничителями, и если характеристики повторяющихся элементов изменятся, потребуется вносить изменения в схему. При необходимости, для повторяющихся элементов можно также определить атрибут для хранения порядковых номеров. Такой атрибут имеет целочисленный тип (в зависимости от возможного диапазона номеров: "byte", "short", "int", "long" или "integer"). Его значение, вычисляемое обрабатывающим приложением, определяет внутренний порядок повторяющихся элементов (относительное положение элемента в группе повторяющихся элементов).




```
<?xml version="0.1" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="PostalAddress">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref  ="AddressTo"/>
    <xs:element ref  ="AddressLines"/>
    <xs:element ref  ="AddressCity"/>
    <xs:element ref  ="AddressPostalCode"/>
    <xs:element ref  ="AddressCountry"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name="AddressTo" type="xs:string"/>

 <xs:element name="AddressLines">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref  ="StreetAddressLine" minOccurs="1" maxOccurs="4"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

 <xs:element name="AddressCity" type="xs:string"/>
 <xs:element name="AddressPostalCode" type="xs:string"/>
 <xs:element name="AddressCountry" type="xs:string"/>

 <xs:element name="StreetAddressLines">
  <xs:complexType>
   <xs:simpleContent>
    <xs:extension base="xs:string"/>
     <xs:attribute name=  "sequence" use="required" type="xs:byte"/>
    </xs:extension>
   </xs:simpleContent>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Рис. 7.6. W3C XML-Схема для простой абстрактной формы контейнера

Техника. *При правильном применении, задание значений атрибутам minOccers и maxOccurs для повторяющихся элементов является мощной возможностью W3C XML-Схем по указанию степени кардинальности. Однако следует соблюдать осторожность, если кардинальность или характеристики повторяющихся элементов подвержены изменениям. Такие изменения потребуют модификации соответствующих W3C XML-Схем.*

Факт. *Абстрактные архитектурные формы контейнеров могут внести сложность в навигацию и обработку структуры XML-документа. Для опроса характеристик атрибутов может потребоваться дополнительная логика.*

Для абстрактной формы контейнера характерно использование повторяющихся элементов с абстрактными именами и атрибутами, описывающими порядок следования этих элементов. В зависимости от сложности структуры и уровня абстракции, абстрактные формы контейнера усложняют навигацию по документу. Если в документе присутствуют группы повторяющихся элементов, такие методы синтаксического анализатора, как “getElementsByTagName”, не обеспечивают

необходимой уникальности каждого возвращаемого элемента. Для обеспечения уникальности или выборочной обработки определенных экземпляров из числа повторяющихся элементов, приложению требуется дополнительная логика по работе с "порядковыми" атрибутами.

Гибридные формы контейнера

Гибридные формы контейнера усиливают наиболее удачные характеристики жесткой и абстрактной форм. Гибридные формы контейнера тоже включают контейнеры-элементы с конкретизированными уникальными именами. А кроме этого, в их структуру можно включать и группы повторяющихся элементов с абстрактными именами для поддержки изменяемых данных. Для описания дополнительных характеристик, таких как порядковый номер, используются атрибуты (рис. 7.7).

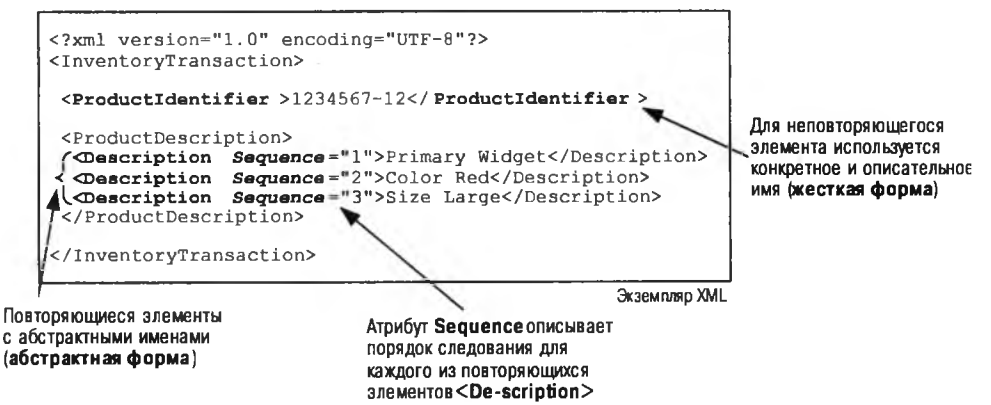


Рис. 7.7. Гибридная форма контейнера

В гибридной форме контейнера атрибуты также используются и для описания добавочных характеристик повторяющегося элемента, таких, как тип (классификация). Если прикладной программе требуется дополнительная информация для устранения неоднозначности, можно ввести атрибут, содержащий определяемый пользователем тип (классификацию), это поможет уточнить контекст или способ использования данного элемента (рис. 7.8). Обрабатывающее приложение должно проанализировать содержимое этого атрибута "type"(тип) и применить соответствующую логику обработки.

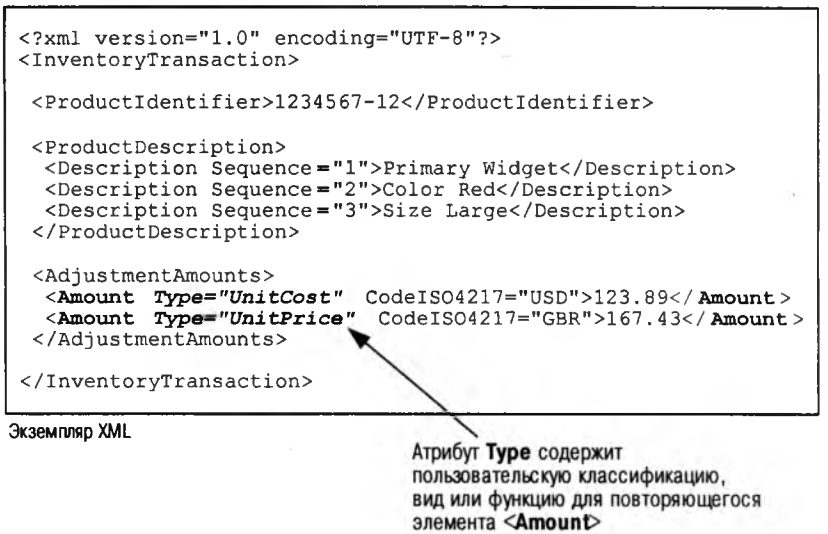


Рис. 7.8. Гибридная форма контейнера с атрибутом типа

В гибридной форме контейнера атрибуты также можно использовать для представления значений стандартных кодов. Этот способ наиболее эффективен, когда основное содержание контейнера-элемента может быть дополнительно описано одним или более кодами. Рассмотрим элемент, предназначенный для денежных сумм. Значение суммы является числом и может иметь дробные десятичные позиции. Кроме того, для этого элемента определен атрибут, описывающий для каких целей предназначена сумма (рис. 7.9). Такой подход придает элементу черты интеллекта.

Гибридные формы контейнеров также использую атрибуты для описания связанной с элементом функции или деятельности. Это особенно важно, когда XML-документ предназначен для содержимого, ориентированного на транзакцию. Для транзакций, например, по настройке инвентаризации, помимо описания необходимых характеристик, требуется указать и то, какое действие следует выполнить. Традиционные приложения предприятия часто обрабатывают файлы транзакций, включающие код или вид транзакции. Подобные характеристики можно определить и для XML-транзакции, предназначенной для тех же целей. И хотя связанную с транзакцией функцию или деятельность можно описать как с помощью элемента, так и с помощью атрибута, применительно к вертикальной модели и гибридной форме контейнера, лучше использовать для этого атрибуты (рис. 7.10).

```
<?xml version="1.0" encoding="UTF-8"?>
<InventoryTransaction>

 <ProductIdentifier>1234567-12</ProductIdentifier>

 <ProductDescription>
  <Description Sequence="1">Primary Widget</Description>
  <Description Sequence="2">Color Red</Description>
  <Description Sequence="3">Size Large</Description>
 </ProductDescription>

 <AdjustmentAmounts>
  <Amount Type="UnitCost" CodeISO4217="USD" >123.89</Amount>
  <Amount Type="UnitPrice" CodeISO4217="GBR" >167.43</Amount>
 </AdjustmentAmounts>

</InventoryTransaction>
```

Экземпляр XML

Атрибут **CodeISO4217** описывает значение стандартного кода (кодировку) для элемента <**Amount**>

Рис. 7.9. Гибридная форма контейнера с атрибутом для значений стандартных кодов

```
<?xml version="1.0" encoding="UTF-8"?>
<InventoryTransaction>

 <ProductIdentifier>1234567-12</ProductIdentifier>

 <ProductDescription>
  <Description Sequence="1">Primary Widget</Description>
  <Description Sequence="2">Color Red</Description>
  <Description Sequence="3">Size Large</Description>
 </ProductDescription>

 <AdjustmentQuantities>
  <Quantity Type="Add" Location="AC1" UnitOfMeasure="EA">100</ Quantity>
  <Quantity Type="Sub" Location="DE6" UnitOfMeasure="EA">167.43</ Quantity>
 </AdjustmentQuantities>

 <AdjustmentAmounts>
  <Amount Type="UnitCost" CodeISO4217="USD">123.89</Amount>
  <Amount Type="UnitPrice" CodeISO4217="GBR">167.43</Amount>
 </AdjustmentAmounts>

</InventoryTransaction>
```

Экземпляр XML

Для элемента <**Quantity**> атрибут **Type** описывает связанную с элементом функцию

Рис. 7.10. Гибридная форма контейнера с атрибутом Type для описания функции

Другая область эффективного использования контейнеров-атрибутов – описание составных частей значения данных. В некотором отношении это подразумевает перенос в XML-транзакции избыточных данных (т. е. как значения целиком, так и его разрозненных составных частей). Принимая во внимание тот факт, что XML-транзакция может создаваться для нескольких целей, такая техника является вполне уместной. Рассмотрим пример с полным именем человека. XML-документ или транзакция, содержащая полное имя, может обрабатываться приложениями, формирующими списки рассылки, письма и тому подобную корреспонденцию. Для приложения подобного типа достаточно иметь полное имя в целом. Другим приложениям может потребоваться сортировка, группирование и отбор данных на основании отдельных частей полного имени (например, сортировать список покупателей по фамилиям, или вначале по last name (последнему имени), а затем по first name (первому имени). Техника, позволяющая обращаться к различным кусочкам полного значения и обрабатывать их, заключается в том, чтобы хранить полное имя в качестве данных элемента и также хранить отдельные части полного имени, как значения атрибутов этого элемента (рис. 7.11). И хотя такая техника позволяет получать высоко повторно используемую XML-структуру, она увеличивает размер транзакции, а кроме того, требуется дополнительная программная логика, позволяющая определить, когда следует обработать содержимое элемента, а когда – содержимое атрибутов.

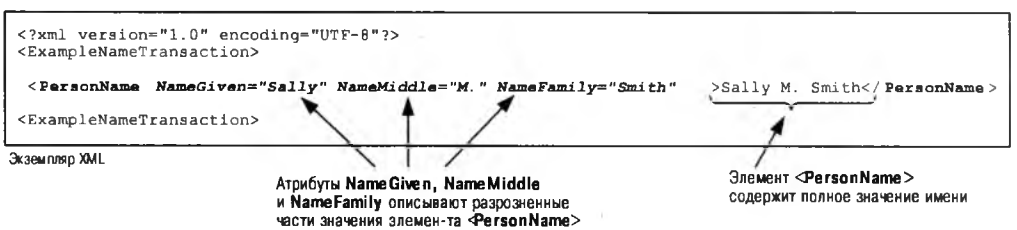


Рис. 7.11. Использование атрибутов для разбиения данных элемента на части

Суммируя, можно сказать, что гибридная форма контейнера сочетает использование элементов с конкретизированными именами и использование повторяющихся элементов с определенной кардинальностью, имеющих абстрактные имена. Во избежании неопределенности и для обеспечения дополнительного контекста определяются атрибуты: порядка следования, типа или классификации, стандартных кодов, а также указания функции или деятельности. Имена для атрибутов и элементов назначаются проектировщиком данных в соответствии со стандартами именования предприятия и стандартами именования XML. Для назначения типов

для данных используется атрибут “type” W3C XML-Схемы или применяется нестандартный тип данных, определяемый с помощью “simpleType”. При необходимости используются фасеты, налагающие дополнительные ограничения на тип данных (листинг 7.4).

Листинг 7.4 Пример XML-схемы для адаптированной гибридной формы контейнера

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="InventoryTransaction">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="ProductIdentifier"/>
        <xs:element ref="ProductDescription"/>
        <xs:element ref="AdjustmentQuantities"/>
        <xs:element ref="AdjustmentAmounts"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="ProductIdentiner" type="xs:string"/>

  <xs:element name="ProductDescription">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Description" min0ccurs="l"
        max0ccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="AdjustmentQuantities">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Quantity" min0ccurs="l"
        max0ccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="AdjustmentAmounts">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Amount" min0ccurs="l"
        max0ccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

```
<xs:element name="Description">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="Sequence" use="required"
        type="xs:byte"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="Quantity">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:integer">
        <xs:attribute name="Type" use="required">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="Add"/>
              <xs:enumeration value="Sub"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="Location" use="required"
        type="xs:string"/>
        <xs:attribute name="UnitOfMeasure" use="required">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="EA"/>
              <xs:enumeration value="DZ"/>
              <xs:enumeration value="GR"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:element name="Amount">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="AmountSimpleType">
        <xs:attribute name="Type" use="required">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="UnitCost"/>
              <xs:enumeration value="UnitPrice"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="CodeIS04217" use="required">
          <xs:simpleType>
```

```
            <xs:restriction base="xs:string">
              <xs:enumeration value="GBR"/>
              <xs:enumeration value="USD"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:simpleType name="AmountSimpleType">
  <xs:restriction base="xs:decimal">
    <xs:totalDigits value="7"/>
    <xs:fractionDigits value="2"/>
  </xs:restriction>
</xs:simpleType>

</xs:schema>
```

Рекомендация. *С некоторыми исключениями, гибридная форма контейнера – рекомендуемая архитектурная адаптация для структуры XML-транзакции.*

К преимуществам гибридной формы контейнера относятся: применение строгих практик именования, что ведет к выработке интуитивно понятных имен, в сочетании с группами повторяющихся элементов, динамически наращиваемыми или сокращаемыми, в соответствии с характеристиками текущих данных. Гибридная форма контейнера имеет сходство с устойчивой логической моделью данных, созданной для поддержки повторяющихся экземпляров данных (т. е. отношения и кардинальность) и описательных характеристик (рис. 7.12). Применительно к XML-транзакции, гибридная форма контейнера имеет несколько недостатков. Подобно абстрактной форме, ей присуща некоторая сложность навигации по повторяющимся элементам и необходимость в логике, связанной с дополнительными атрибутами. Гибридные формы контейнера хорошо совместимы с вертикальной, компонентной и гибридной структурными моделями. Возможна и некоторая совместимость с горизонтальными структурными моделями (теми, что опираются в основном на атрибуты), но здесь серьезным ограничением является невозможность повторения атрибутов. В качестве общей рекомендации, использование тщательно сконструированных гибридных форм контейнера, отвечающих требованиям проекта, использующих эффективный стандарт именования и допускающих данные переменного состава, является наиболее оптимальным подходом.

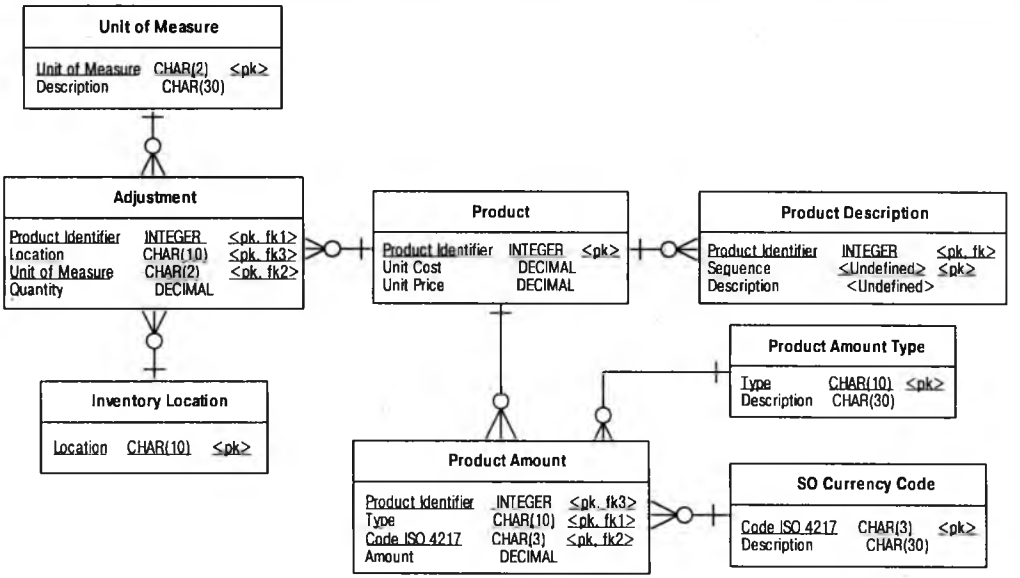


Рис. 7.12. Простая модель данных, представляющая гибридную форму контейнера

Как мы узнали, традиционное моделирование данных и моделирование XML-структуры имеют множество общих черт и областей деятельности. Кроме того, мы узнали также, что архитектурные формы контейнера отражают наиболее заслуживающие внимания и часто используемые шаблоны данных. Их применение относится к компетенции и обязанностям проектировщика данных. Для проектировщика данных существуют и другие поводы обратиться к мощи W3C XML-Схем. Один из них – повторное использование.

Глава 8

W3C XML-Схемы и повторное использование

Повторное использование – одна из самых запутанных концепций информационных технологий. Многие из исполнителей, работающих в этой области, предполагают, что что-либо из их наработок (например, модели, проекты, прикладные программы, базы данных, интерфейсы и транзакции) может быть с пользой применено повторно. Они надеются получить экономические выгоды от повторного использования, но часто находятся в неведении о тактической цене, которую необходимо заплатить за достижение этих выгод. Менеджеры, работающие в области технологии, сталкиваясь с жесткими графиками и ограниченными бюджетами, часто избегают изменения методологии разработки, которое требуется для достижения эффективного повторного использования. Разработчики стремятся к быстрой отдаче, к скорейшему увеличению количества отлаженных строк кода. Иными словами, цели, которые управляют разработкой новых приложений, не совпадают с поддержкой повторного использования. Однако не все потеряно. Синтаксические и функциональные возможности, предоставляемые W3C XML-Схемами, обеспечивают мощную поддержку возможности повторного использования метаданных. Первой задачей на этом пути является достижение понимания основ повторного использования.

С точки зрения информационной технологии, *повторное использование* – процесс, состоящий из двух частей: первая – разработка информационных активов, специально с расчетом на возможность использовать их более одного раза; вторая – собственно повторное использование наработанных с помощью этой технологии активов:

1. Разработка, ориентированная на неоднократное использование результатов (reuse engineering);
2. Повторное использование наработанного (reuse harvesting).

Разработка с прицелом на повторное использование – это набор практических приемов и технических подходов, необходимых для проектирования, разработки и описания технологического актива, который можно использовать неоднократно. Чтобы разработать что-либо, пригодное для повторного использования, проектировщик данных должен учитывать то, как этот информационный актив будет использоваться вначале, а также то, как он будет использоваться в дальнейшем. Разработка, ориентированная на повторное использование, включает в себя архитектурный подход, при котором разработка не встречает ограничений, диктуемых текущими обстоятельствами и требованиями проекта. Здесь также важно определить, можно ли повторяемый шаблон, представляемый технологическим активом, использовать повторно в его текущем контексте (внутридоменное или доменно-зависимое повторное использование) и в других контекстах (междоменное повторное использование[1]).

Факт. *Разработка повторноиспользуемой XML-схемы – это набор практических приемов, технических подходов и мероприятий, необходимых для разработки W3C XML-Схемы или схемы-компонента, с учетом необходимости ее повторного использования.*

И внутридоменное, и междоменное повторное использование приносят пользу. Выгоды от внутридоменного повторного использования обычно проявляются с ростом количества применений повторноиспользуемых элементов в границах одного контекста. Таким контекстом может быть W3C XML-Схема, используемая в качестве репрезентативного словаря (например, схемы представляющие транзакцию по заказу покупателя, транзакцию по закупке, или транзакцию по платежной ведомости). Внутридоменное повторное использование, как правило, ограничивает область применимости технологического актива и, следовательно, требует менее значительных усилий по разработке. В качестве примера можно рассмотреть первоначальное использование и дальнейшее повторное использование определения типа данных для денежных сумм в W3C XML-Схеме – словаре (рис. 8.1).

Факт. *Собственно повторное использование XML-схемы – процесс, включающий действия по определению возможности повторного использования, проверке и реализации повторного использования W3C XML-Схем, подсхем или схем-компонентов.*

Междоменное повторное использование предполагает большую степень абстракции и обобщения, допускающую широкомасштабное повторное использование, и может потребовать более значительных усилий по разработке. Благодаря тому, что междоменное повторное использование предоставляет возможность повторного использования в других контекстах, потенциальные выгоды здесь могут быть больше, чем от внутридоменного повторного использования. Междоменное повторное использование W3C XML-Схемы подразумевает, что эта схема определяется как внешняя подсхема, и на нее можно ссылаться из других W3C XML-Схем. При этом каждая из ссылающихся схем предоставляет свой специфический контекст применения (рис. 8.2).

[1] Примечание: Karlson, E-V. Software Reuse – A Holistic Approach. John Wiley & Sons, New York, 1995.

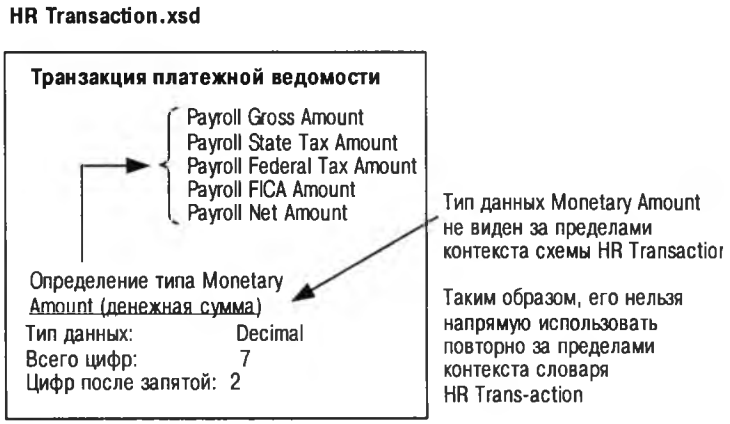


Рис. 8.1. Внутридоменное повторное использование (внутри схемы)

Собственно повторное использование – это набор процессов, связанных с установлением возможности повторного использования, поиском информационных активов – кандидатов на повторное использование, проверкой соответствия этих активов возможности повторного использования (наподобие проверки соответствия шаблону), и подключения или организации ссылки на этот информационный актив (т. е. сборка при помощи ссылок, вместо новой разработки). Главной составляющей издержек на первоначальное повторное использование являются затраты на разработку, ориентированную на повторное использование (reuse engineering). Однако существуют некоторые дополнительные издержки, связанные с собственно повторным использованием (reuse harvesting). Они складываются из усилий по определению активов, которые можно использовать повторно, проверки того, что эти активы отвечают требованиям текущего проекта, и реализации повторного использования активов.

С каждым новым повторным использованием подготовленного для этого информационного актива, выгода становится все более очевидной. Стоимость, связанную с разработкой и автономным тестированием повторноиспользуемого информационного актива до некоторой степени можно перенести на каждую из последующих реализаций этого актива (благодаря снижению стоимости всех последующих разработок, в результате повторного использования информационного актива и отсутствия необходимости разрабатывать и автономно тестировать "новый" актив). Разработка, ведущаяся с прицелом на повторное использование – начальная стадия процесса повторного использования, и именно здесь происходят наибольшие затраты. Собственно повторное использование – тот этап, где выгоды от повторного использования можно ощутить и измерить.

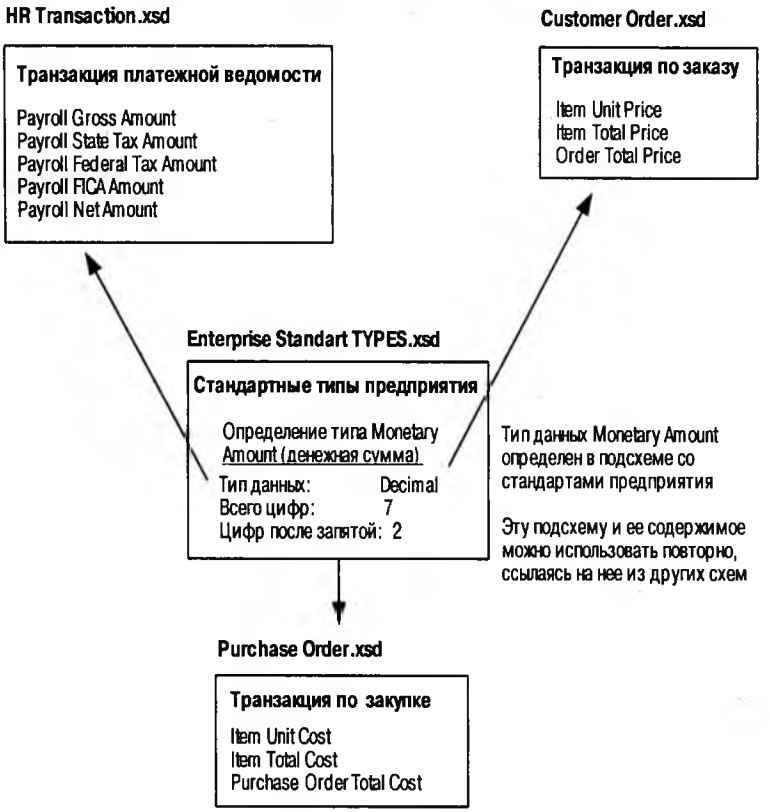


Рис. 8.2. Междоменное повторное использование (за пределами схемы)

W3C XML-Схемы предоставляют несколько типов возможностей повторного использования. Самые основные возможности согласуются с концепцией внутридоменного повторного использования. *Доменом* является отдельная прикладная система, набор взаимосвязанных бизнес-функций, или определенный контекст. В частности, словарь, базирующийся на W3C XML-Схеме – это набор контейнеров, представляющий домен или некоторую часть домена. Контейнеры (атрибуты и элементы XML), группы контейнеров и типы данных (характеристики метаданных и нестандартные типы данных) можно определить так, чтобы их можно было использовать повторно в пределах отдельного словаря – W3C XML-Схемы. Кроме того, стандартные для предприятия контейнеры, структуры и определения метаданных могут быть разработаны в виде подсхем. При этом последующие W3C XML-Схемы можно

разрабатывать как сборки из ранее созданных, определенных вовне W3C XML-Схем и схем-компонентов. Проектировщик данных, на базе W3C XML-Схемы, может разработать словарь, ссылающийся на стандартные подсхемы, вместо того чтобы кодировать в словаре каждую из них.

Повторное использование в пределах одной W3C XML-Схемы

Повторное использование в пределах W3C XML-Схемы может иметь несколько форм. Вообще, элементы в W3C XML-Схеме могут определяться либо локально, либо глобально. Локально определяемые контейнеры-элементы определяются по имени в той точке схемы, где они объявлены (рис. 8.3). Такие локально определяемые элементы и атрибуты обычно не используются повторно в других местах схемы, вне точки их объявления. Глобально определяемые элементы представляют собой основную форму повторного использования. Они определяются для всей схемы, а не только для того места, где они находятся. Глобально определенные элементы – это контейнеры, которые определяются с целью неоднократного их использования в разных местах схемы с помощью ссылок (рис. 8.4).

Может оказаться полезным и даже необходимым определить и использовать по ссылке не отдельные элементы, а целый набор похожих или взаимосвязанных элементов. В W3C XML-Схемах это можно сделать, используя синтаксис "complexType" или "group". С помощью *complexType* (сложный тип) определяется набор контейнеров-элементов; если этот набор имеет имя, его можно использовать повторно с помощью ссылки (используя в ссылающемся элементе синтаксис "extension"). Аналогично, с помощью *group* определяется набор контейнеров (также называемый группой модели элемента), предназначенный специально для повторного использования. Два глобально определенных элемента из предыдущего примера можно было бы определить и повторно использовать как единый набор элементов, не применяя две отдельные ссылки на элементы. Такой набор, с помощью complexType, можно определить глобально, для всей схемы, и ссылаться на него по имени из разных элементов (используя extension) (рис. 8.5).

Рекомендация. *Все контейнеры-элементы XML следует определять глобально, чтобы их можно было использовать повторно. Это не касается только тех элементов, которые намеренно защищаются от повторного использования.*

Рекомендация. *Наборы взаимосвязанных контейнеров-элементов, специально предназначенные для повторного использования, следует определять как группу ("group"). За исключением случаев, когда очевидны преимущества использования для этих целей сложных типов ("complexType").*

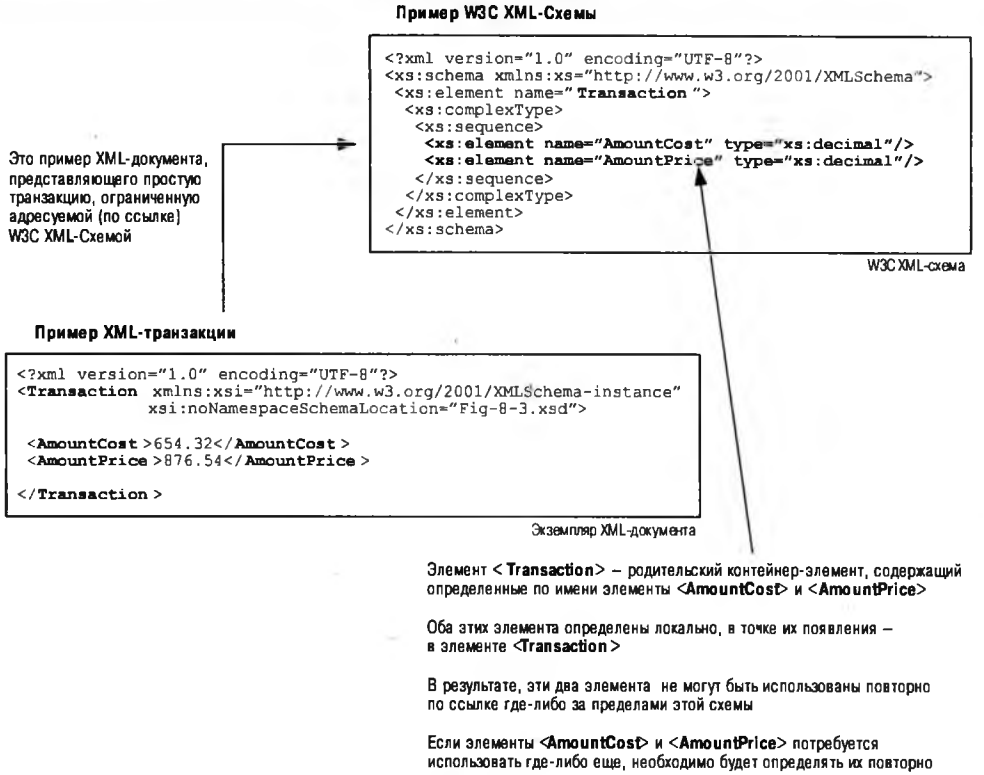


Рис. 8.3. Локальные определения элементов в W3C XML-Схеме

Как и сложный тип ("complexType"), группа ("group") также является набором контейнеров-элементов. Понятие группы похоже на понятие сложный тип, но для группы используется немного другой синтаксис. Кроме того, группа определяется специально для повторного использования с помощью ссылок, тогда как сложный тип может определяться и для повторного использования, и локально, без возможности повторного использования. Содержимым группы может быть набор элементов или сложный тип (рис. 8.6).

И сложный тип, и группа допускают использование компоновщика. *Компоновщик* (compositor) задает порядок следования и выборочное появление контейнеров, определенных в группе или сложном типе. Существуют следующие компоновщики: *sequence* (последовательность), *all* (все) и *choice* (выбор) (табл. 8.1). Компоновщик *sequence* объявляет, что элементы, определенные в сложном типе или группе, должны появляться в соответствующем XML-документе в том же

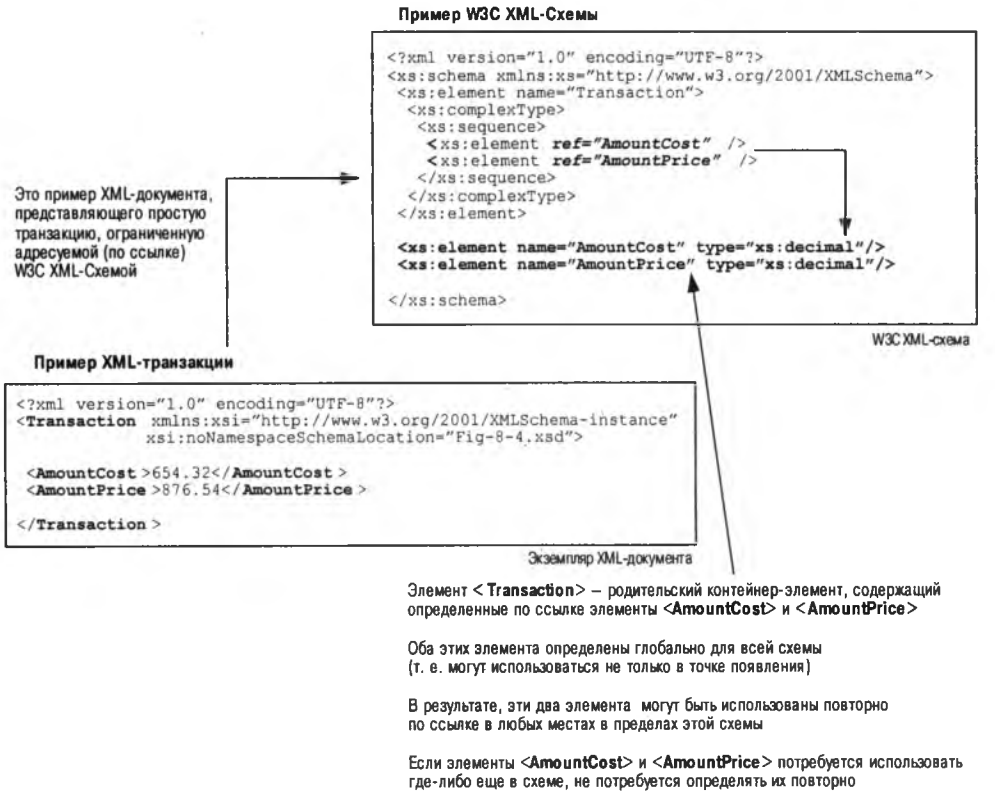


Рис. 8.4. Глобальные определения элементов в пределах W3C XML-Схемы

порядке, в каком они были определены. Компоновщик *all* объявляет, что в XML-документе должны присутствовать все элементы, определенные в сложном типе или группе, причем в любом порядке. Компоновщик *choice* указывает, что из всех элементов, определенных в группе или сложном типе, в соответствующем XML-документе может появиться только один.

Кроме того, большинство элементов, определенных в группе или сложном типе (в том числе элементов, на которые имеются ссылки), могут повторяться. Для указания степени кардинальности повторяющихся элементов используются атрибуты *minOccurs* и *maxOccurs*. Атрибут *minOccurs* определяет минимальную степень кардинальности или наименьшее возможное количество повторений элемента. Значение атрибута *minOccurs* равное нулю (“0”) указывает на необязательность (опциональность) элемента. Атрибут *maxOccurs* определяет максимальную

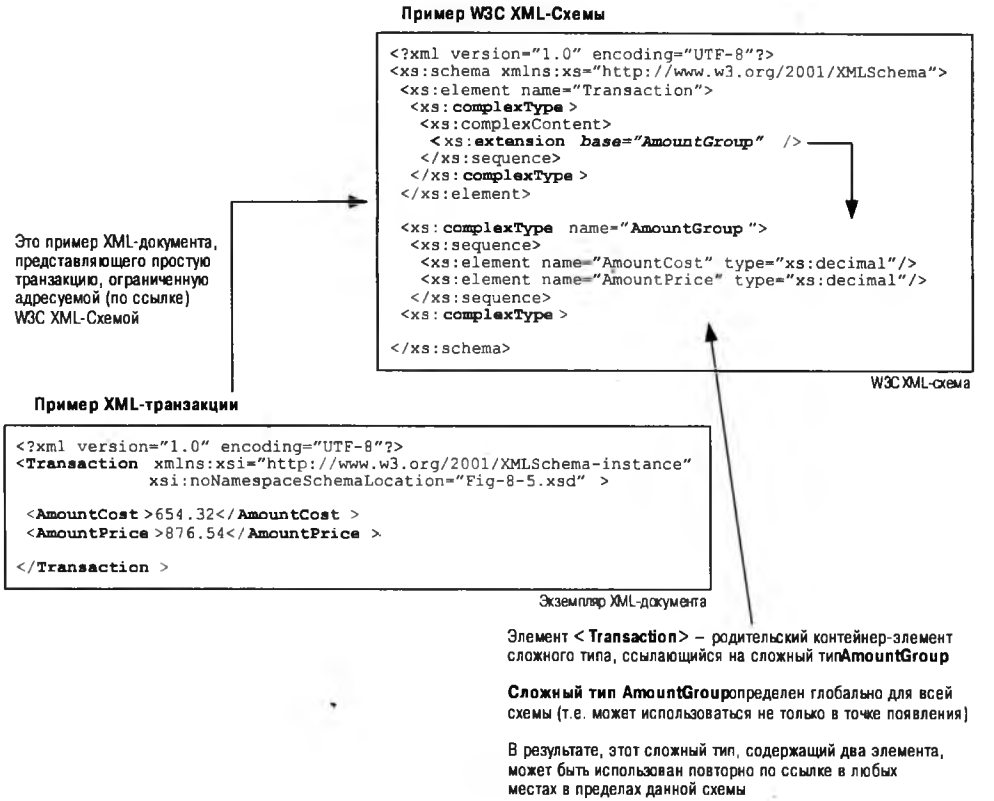


Рис. 8.5. Глобальное для W3C XML-Схемы определение сложного типа (complexType)

степень кардинальности, или наибольшее допустимое количество повторений элемента. И минимальная, и максимальная степени кардинальности элемента могут задаваться определенным значением (например, "3" или "200"). Максимальная степень кардинальности, кроме того, может быть указана как бесконечная (значение "unbounded"). Важно отметить, что в некоторых случаях правила, диктуемые компоновщиком, могут ограничивать кардинальность элементов (например, компоновщик "all" не допускает повторения элементов).

Как описано в главе 4, W3C XML-Схемы обеспечивают расширенную поддержку типов данных, включая многочисленные предопределенные и производные типы данных, которые можно использовать для ограничения допустимых значений любого элемента или атрибута. Также существует возможность определить собственный, нестандартный тип данных, создав простой тип (*simpleType*):

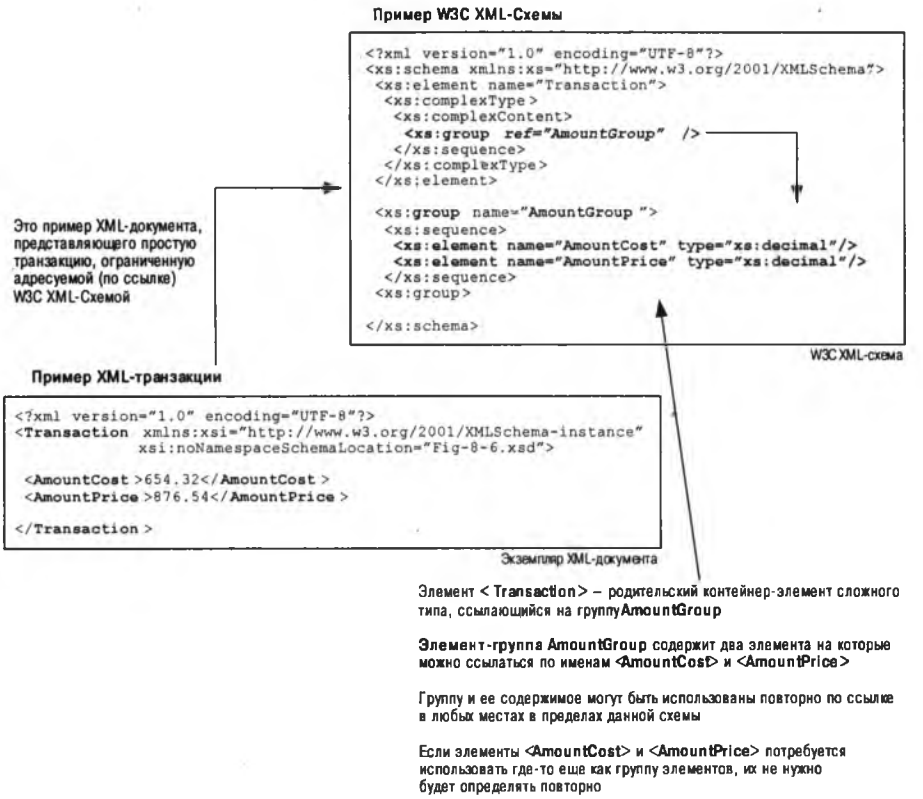


Рис. 8.6. Глобальное определение группы в W3C XML-Схеме

Таблица 8.1. Компоновщики

Вид компоновщика	Описание
sequence	Дочерние элементы должны появляться в соответствующем XML-документе в том же порядке, в каком они были определены. Степень кардинальности, указанная для дочерних элементов, определяет, может должен ли (или может ли) дочерний элемент появляться и сколько раз он может повторяться
all	Все дочерние элементы должны присутствовать в соответствующем XML-документе. Они могут появляться в любом порядке, но не должны повторяться (т. е. нельзя задать степень кардинальности *maxOccurs* больше чем "1")
choice	Из всех элементов должен присутствовать только один

используя в качестве базового один из поддерживаемых типов и добавляя к нему ограничивающие фасеты. Возможность создания нестандартного типа данных – мощная основа повторного использования. Во многих организациях созданы наборы стандартных (в рамках предприятия) определений элементов данных. Когда определяются новые элементы данных такого же типа, проектировщику данных требуется лишь применить к ним стандартные характеристики метаданных (т. е. тип данных, длину, количество цифр и допустимые значения).

Техника. *Простой тип (simpleType) W3C XML-Схемы является мощным методом определения стандартных типов данных предприятия и ограничений на допустимые значения контейнеров (элементов и атрибутов).*

В качестве примеров стандартных элементов данных предприятия можно привести элементы для денежных сумм, идентификаторов (например, как в первичном ключе или уникальных идентификаторов), текстовых описаний и значений стандартных кодов. Подобные стандарты предприятия можно определять как нестандартные типы данных, реализуя их в W3C XML-Схеме как простые типы. Подобно элементу, простой тип (*simpleType*) можно определить локально, и тогда он будет относиться только к одному элементу или атрибуту, а можно определить глобально, для всей схемы, и использовать неоднократно по ссылке. Определение нестандартного типа с помощью *simpleType* включает объявленный тип данных W3C XML-Схемы и любые допустимые фасеты (рис. 8.7). В этом примере нестандартные типы данных для денежных сумм определены как десятичные типы данных (база – тип "decimal") с фасетами *totalDigits* и *fractionDigits*. Использование простого типа не ограничено денежными суммами или десятичными типами данных. В них можно применять любой поддерживаемый в W3C XML-Схеме тип данных и любые фасеты.

Повторное использование элементов, групп элементов и типов данных в пределах одной W3C XML-Схемы – очень мощная возможность. Однако повторное использование таких внутренних конструкций вне контекста W3C XML-Схемы, где они определены, может оказаться сложным и в некоторых случаях невозможным. Для этого, возможно, потребуется повторить определения этих конструкций в каждой из других схем (в форме копирования-вставки), где предполагается их использовать. В таком подходе есть свои преимущества, но с ним связаны риск и возрастание затрат. К очевидным преимуществам относится хотя и своеобразное, но распространение и повторное использование стандартных элементов и типов данных предприятия. Однако стоимость обслуживания каждой из получаемых таким образом схем возрастает. К тому же с каждым разом возрастает угроза возникновения ошибок в результате несинхронных изменений копий. Более эффективным подходом является размещение определений конструкций, предназначенных для повторного использования, во внешних W3C XML-Схемах.

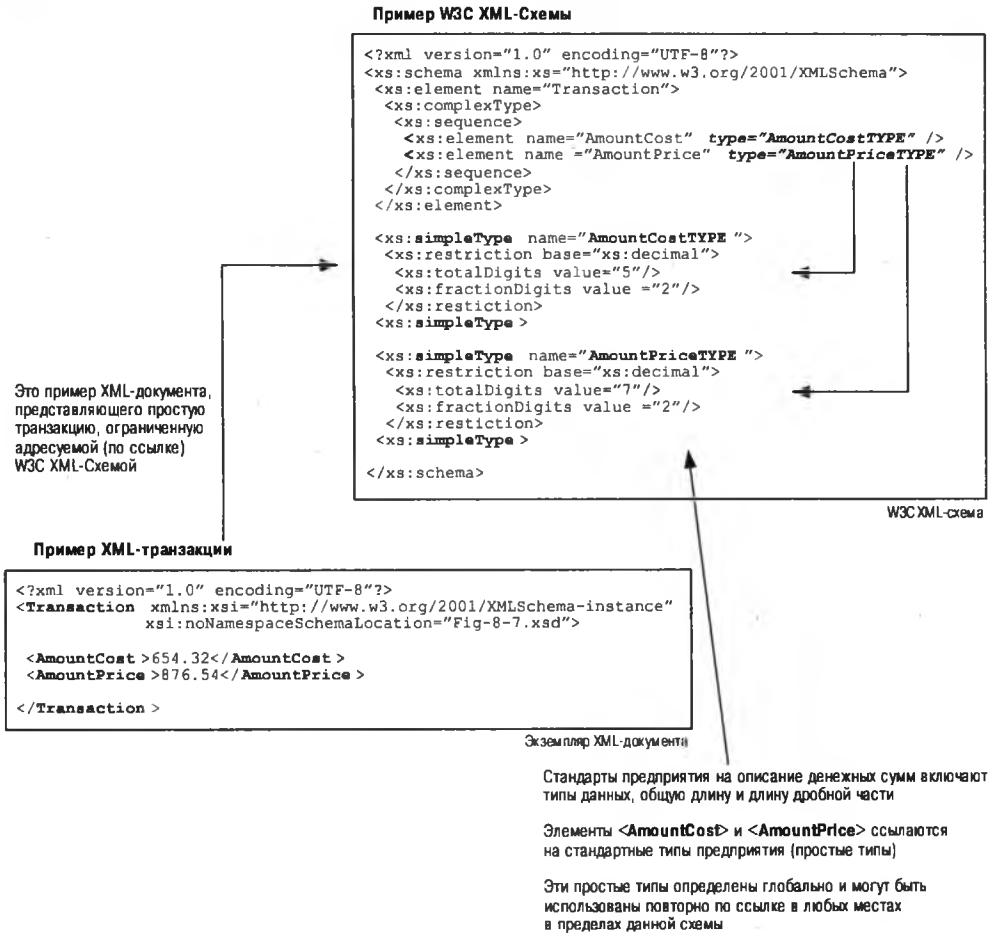


Рис. 8.7. Глобальные для W3C XML-Схемы определения простых типов

Повторное использование внешней W3C XML-Схемы(подсхемы-компонента))

С точки зрения повторного использования, W3C XML-Схема обеспечивает широкие возможности и в области междоменного повторного использования. При междоменном повторном использовании схема (назовем ее в этом случае подсхемой) представляет собой повторяемый шаблон, который можно использовать в разных кон-

текстах, в различных приложениях. Способ реализации – определение модульных W3C XML-Схем в качестве внешних подсхем. Другие W3C XML-Схемы (возможно принадлежащие к разным контекстам) могут ссылаться на эти подсхемы и использовать повторно элементы, группы и типы данных, определенные во внешних подсхемах. В адресуемой подсхеме-словаре необходимо определять контейнеры, структуры и типы, относящиеся исключительно к одному определенному контексту. Повторное использование определенных в подсхемах элементов, групп и типов данных с помощью ссылки является формой разработки методом сборки (рис. 8.8).

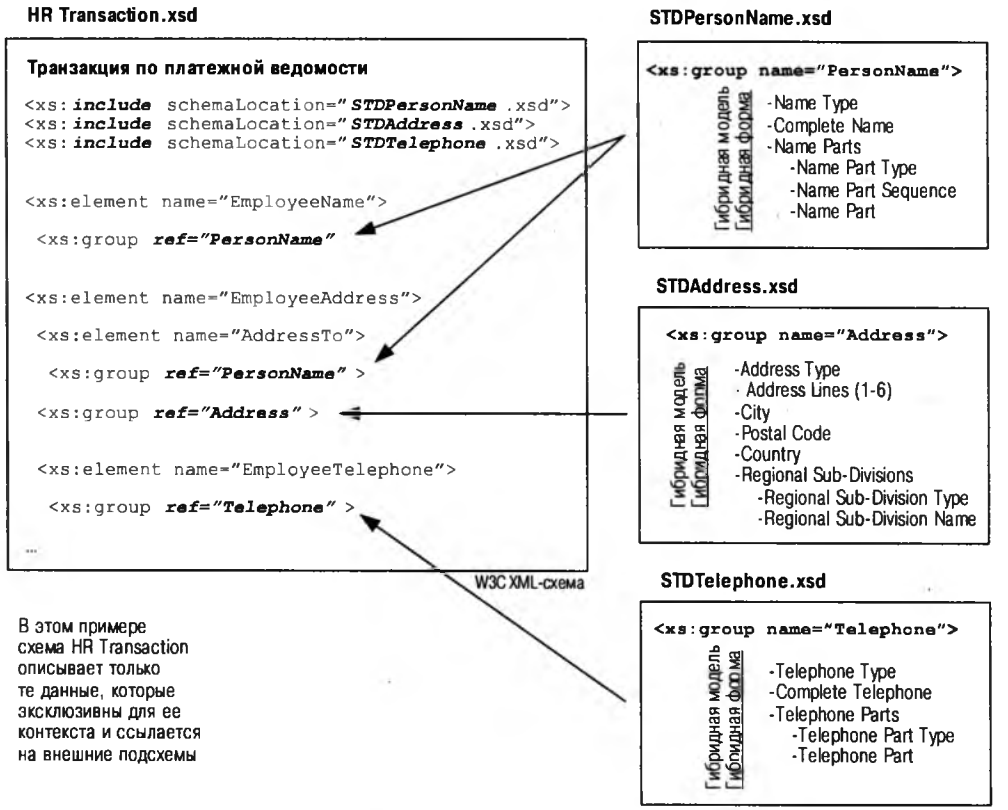


Рис. 8.8. Повторное использование W3C XML-Схемы с помощью ссылки ("сборка")

Факт. *Повторное использование W3C XML-Схемы или подсхемы – концептуальная форма разработки методом сборки. Основная W3C XML-Схема собирается с помощью ссылок на содержимое других, определенных вовне W3C XML-Схем.*

Разработка повторноиспользуемой схемы – это процесс создания W3C XML-Схем (или подсхем), ведущийся с прицелом на их повторное использование. Каждая из внешних подсхем должна определяться таким образом, чтобы способствовать ее широкомасштабному повторному использованию и с обязательной ориентацией на стандартные структуры метаданных предприятия. И здесь существенную роль играет проектировщик данных. Выяснение того, какие из элементов, структур и типов данных являются кандидатами на повторное использование, находится в русле обычной практики проектировщика данных. Наилучшими кандидатами на разработку в виде повторноиспользуемой схемы являются:

- наиболее стандартизированные структуры данных (например, полное имя или почтовый адрес, телефонный номер, структуры семейств продуктов, географические структуры);
- стандартные коды и допустимые значения;
  - списки стандартных кодов предприятия;
  - списки международных, отраслевых стандартных кодов (например, сокращения названий штатов США, коды стран, коды валют);
- стандартные для предприятия типы данных (например, стандартные типы данных предприятия для денежных сумм, текстовых описаний и идентификаторов).

Факт. *Наиболее общие виды повторноиспользуемых подсхем включают стандартизированные структуры, стандартные коды и допустимые значения, а также нестандартные (собственные) типы данных.*

Синтаксис W3C XML-Схемы позволяет определять в подсхеме элементы, группы, сложные типы и простые типы, которые можно подключить, и на которые можно сослаться из других W3C XML-Схем. Эта концепция совершенно аналогична Copybook "include" из языка COBOL, когда определение структур данных и файлов, выполненные в соответствии со стандартами предприятия, затем подключаются при помощи ссылки к исходным текстам других COBOL-программ. Это не только позволяет обеспечить высокую степень повторного использования, но и способствует поддержанию стандартов предприятия на структуры и метаданные.

Одна из серьезных проблем, стоящих сегодня перед предприятиями бизнеса, заключается в разнообразии данных вследствие появления глобальной электронной коммерции. При учете национальных и территориальных различий, даже имя покупателя представляет собой интересную проблему. XML, как технология объединяющих транзакций, может использоваться для обмена информацией об именах между тактическими системами предприятия, такими, как система учета людских ресурсов, система обработки заказа и система обслуживания покупателей. Эта же информация об именах может импортироваться в (или обмениваться с) стратегические системы, такие как маркетинг, CRM-система[2] и хранилище данных. Функ-

циональное использование данных об имени предполагает наличие повторяющихся шаблонов. Однако степень детализации и форматы данных для полного имени могут отличаться от системы к системе. Еще большую сложность вносят в этот вопрос культурные отличия.

В США общий формат для полного имени – это просто комбинация первого, среднего и последнего имен. Однако эти части имени в минимальной степени (или совсем никак) соответствуют национальным и культурным различиям. Во многих странах элементы данных, описывающие имя как набор из первого, среднего и последнего имени не применимы. Варианты имени могут включать данное (given) имя, прозвище (surname), фамилию и дополнительное (additional) имя[3]. Также может отличаться порядок следования частей имени. Кроме того, многие имена не ограничиваются тремя частями и также могут включать связующие (например, "von" или "de").

Другим важным аспектом является то, как имя будет обрабатываться или использоваться. Прикладным программам, генерирующим корреспонденцию, письма и наклейки для доставки, может быть достаточно полного имени покупателя, как единого целого, сцепленного из всех составных частей в требуемой последовательности. Другим приложениям, таким, как маркетинг, управление по связям с пользователями, может потребоваться доступ к отдельным частям имени, например для сортировки, или группирования информации. Высоко повторноиспользуемая структура имени должна поддерживать разные способы применения данных об имени[4].

W3C XML-Схема, описывающая структуру и формат полного имени, может быть разработана с расчетом на широкомасштабное повторное использование. Прочие W3C XML-Схемы могут затем повторно использовать это описание, ссылаясь на нее, как на подсхему, вместо того, чтобы включать собственные узко направленные описания структуры имени, разработанные под конкретное использование. Стратегические приложения, такие как глобальное управление связями с покупателями (global customer relationship management – G-CRM), часто действуют как точка объединения информации о покупателе. В этом случае возможность получать, проверять, импортировать и обрабатывать имена и составные части имен покупателей из разных стран, становится важной составляющей общего успеха. Разработка модульной подсхемы для полного имени требует, чтобы полученная структура позволяла производить разную обработку, поддерживала несколько форматов данных, и при этом все еще использовала стандарты и ограничения (по необходимости) (рис. 8.9).

---

[2] CRM (Customer Relationship Management) – класс прикладных back-end систем для управления взаимодействием с заказчиками. – *Примеч. ред.*

[3] Примечание: Bean J. XML Globalization and Best Practices. Active Education, Colorado, U.S., 2001.

[4] Примечание: Bean J. Engineering Global E-Commerce Sites. Morgan Kaufmann. San Francisco, 2003.

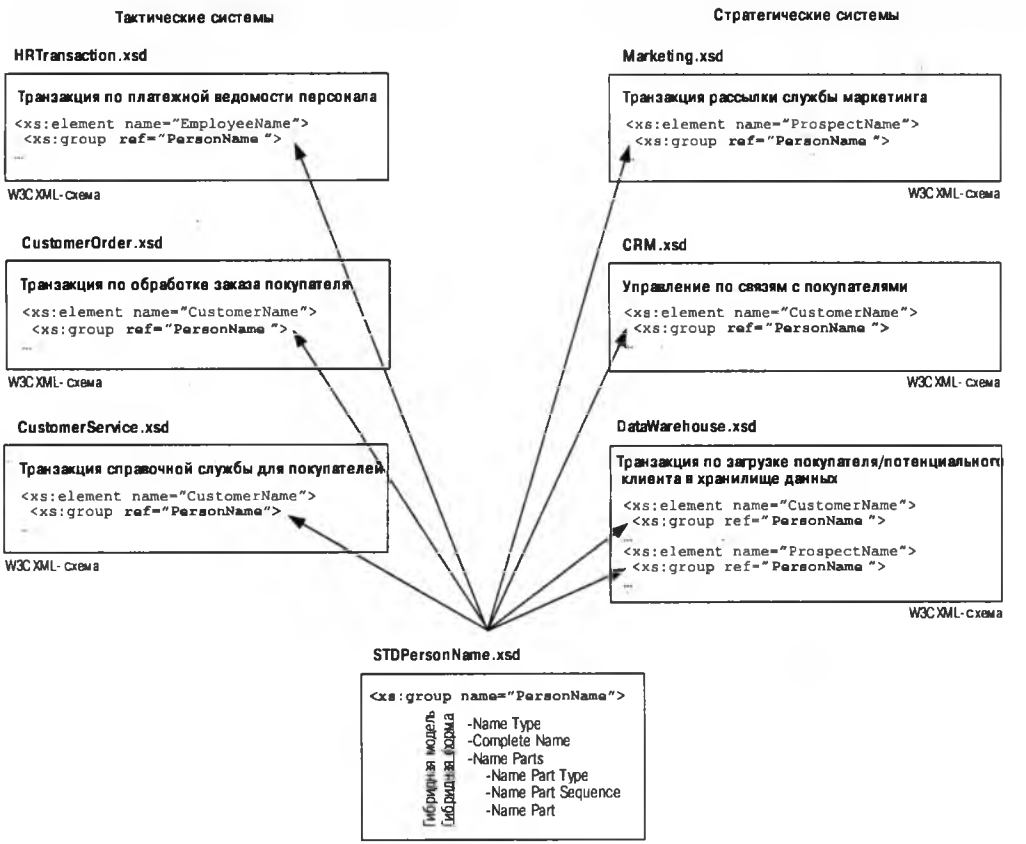


Рис. 8.9. Варианты использования стандартной подсхемы предприятия для полного имени

Схема для полного имени, разрабатываемая с прицелом на широкомасштабное, междоменное повторное использование, должна поддерживать различные формы обработки и разные структурные форматы (рис. 8.10). В тех случаях когда XML используется для описания данных, которые будут импортироваться в структуру базы данных, очень важное значение имеют формат, степень детализации и стандарты именования, используемые в такой структуре. Если данные предназначены для обмена между системами, имеющими разные функции и цели, или если структура, форма и содержимое данных подвержены изменениям, то особое значение приобретает гибкость архитектуры. Установление характера текущего использования, перспективы дальнейшего использования, изменчивость и степень детализации данных, – все это требует внимания проектировщика данных. Чтобы разработать

повторноиспользуемую схему для полного имени, проектировщик данных должен установить, каким образом эти данные будут использоваться, структурироваться, обозначаться и описываться.

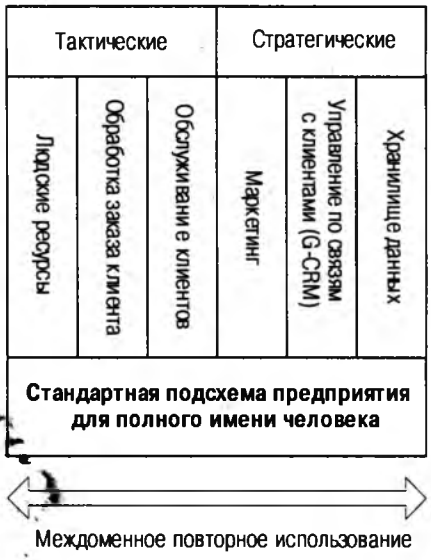


Рис. 8.10. Широкомасштабное междоменное использование подсхемы для полного имени

Архитектурный подход к разработке, ориентированной на повторное использование

Разработка W3C XML-Схем, ориентированных на неоднократное использование, включает действия по проектированию и созданию повторноиспользуемых схем. В этом процессе заметную роль играет проектировщик данных. Чтобы обеспечить широкомасштабное, междоменное повторное использование W3C XML-Схемы, проектировщик данных должен применить архитектурный подход. Этот подход должен учитывать несколько важных аспектов проектирования:

- обозначение схемы (имя файла, допускающее повторное использование);
- управление версиями;
- разные варианты использования и обработки (т. е. как XML-документ будет "потребляться");
- применение структурных моделей;
- адаптация архитектурных форм контейнеров.

Рекомендация. *Внешнее имя файла повторно используемой W3C XML-Схемы (подсхемы) должно быть интуитивно понятным, в оправданной степени конкретным. В нем должны использоваться символы обоих регистров (стиль "camel case") и не должно быть пробелов. Длина имени не должна превышать 32 символов.*

Обозначение и именование схемы критически важны для повторного использования. Внешняя W3C XML-Схема должна именоваться интуитивно понятным образом, упрощающим последующую идентификацию и повторное использование. Имя должно отвечать используемым на предприятии стандартам именования. Подобно важности подходов к именованию, применяемых для отдельных XML-контейнеров (элементов и атрибутов), внешнее имя файла, в котором будет храниться вся схема, также важно. Имя схемы должно быть описательным (информативным), но если оно будет слишком конкретным или ограниченным рамками отдельной системы, приложения или процесса, повторное использование схемы потенциально будет ограничено. Если же наоборот, имя окажется слишком абстрактным, оно может ввести в заблуждение и стать причиной неправильного решения при отборе кандидатов на повторное использование. К примеру, имя "Name" является слишком абстрактным. Оно может означать и имя человека, и название фирмы, и название места, и торговое имя, и многое другое. А имя схемы "PersonName" является описательным, не связано с контекстом и интуитивно понятно.

На физическом уровне, W3C XML-Схема является текстовым файлом в кодировке Unicode (чаще UTF-8 или ASCII). Внешнее имя этого файла должно отвечать ограничениям системы управления файлами и операционной системы (длина имени, возможность использования специальных символов и пробелов, регистр символов). В общем случае не рекомендуется присваивать имена длиннее 32 символов. Имена файлов длиной до 32 символов поддерживаются большинством серверных операционных систем (например, Windows). Если же платформа, где будет храниться подсхема, диктует более жесткие ограничения на максимальную длину, следует придерживаться именно их.

Что касается пробельных символов, некоторые операционные системы допускают использование пробелов в имени файла. Однако рекомендуется удалять из имени все пробельные символы. Также, где это возможно, рекомендуется использовать в имени символы обоих регистров (для визуального деления имени на части, как это было описано относительно именования элементов и атрибутов). Сочетание упомянутых подходов именования позволит создавать описательные и интуитивно понятные имена файлов W3X XML-Схем.

Рекомендация. *Внешнее имя файла повторноиспользуемой W3C XML-Схемы (подсхемы) должно включать номер версии. Номер версии может быть префиксом или суффиксом имени, в зависимости от стандартов предприятия.*

Рекомендация. *Повторноиспользуемые W3C XML-подсхемы должны содержать в имени ту или иную форму классификации или вида (например, "STD" [стандартная структура], "CODES" [список значений кода или перечисления], или "TYPES" [нестандартные типы данных]).*

Управление версиями редко относится к компетенции проектировщика данных. Однако оно играет важную роль в разработке повторноиспользуемых схем, как часть имени схемы (имени внешнего файла). Любым технологическим активам присуща эволюция и изменения. И хотя целью разработки с прицелом на повторное использование является создание повторноиспользуемой структуры схемы, которая будет также стандартом предприятия, со временем требуется вносить изменения и в стандарты. В таких случаях возможность различать разные версии по имени файла становится критически важной. Во время выбора для повторного использования ранее созданных схем, разработчик или проектировщик данных должен иметь возможность легко определить версию схемы-кандидата. Таким образом, включение обозначения версии во внешнее имя схемы имеет большое значение. Способ обозначения версий (номера, символы и комбинации версии и релиза) должен опираться на стандарты предприятия. Обозначение версии схемы должно включаться в имя внешнего файла (обычно в качестве префикса или суффикса).

Может оказаться полезным описывать схему не только по имени и версии, но и по виду (классифицировать ее). К наиболее общим видам W3C XML-Схем относятся: стандартные структуры, наборы значений кода и типы данных. Возможность идентифицировать вид схемы может упростить процесс ее повторного использования. К примеру, виды схем можно классифицировать следующим образом.

- STD – схемы, представляющие стандартные структуры предприятия.
- CODES – схемы, представляющие списки значений стандартных кодов.
- TYPES – схемы, представляющие стандартные типы данных предприятия.

Во внешнее имя файла можно включать вид схемы в соответствии с принятой классификацией. Когда разработчики или проектировщики данных будут подбирать схемы для использования, они смогут ограничить круг поисков определенным видом схем. Имя файла W3C XML-Схемы завершается расширением (когда это поддерживается системой управления файлами и операционной системой). Рекомендуемое расширение имени файла для W3C XML-Схемы – "xsd" (рис. 8.11).

В отличие от традиционного моделирования данных, XML-транзакции для обмена данными между системами могут включать ненормализованные данные, абстрактные имена элементов данных, и даже дублирующиеся данные. В тех случаях, когда транзакция предназначена для обмена данными между различающимися системами и для интеграции предприятия, необходима возможность описывать содержимое в разных формах, под разными именами, и со структурой, легко трансформируемой в разные форматы. Рассмотрим пример, когда

Рис. 8.11. Пример формата имени файла с W3C XML-Схемой

данные одного общего содержания обмениваются между разными системами, в которых по-разному обрабатываются и используются. Чтобы обеспечить возможность разных вариантов использования и обработки данных разными приложениями, может потребоваться одновременно описать несколько структур или форматов одних и тех же данных. В этом случае проектировщик данных сталкивается с необходимостью расширения традиционных навыков и подходов к проектированию данных.

Рекомендация. *Повторноиспользуемая W3C XML-Схема, предназначенная для обмена данными (интеграции предприятия), должна включать в себя несколько разных структур и форматов, учитывающих потенциальные варианты использования и обработки данных.*

Рассмотрим транзакцию для переноса имени покупателя. Данные о полном имени, используемые системой обработки заказов и системой управления связями с клиентами, возможно, должны иметь разную структуру и требуют разной обработки. Системе обработки заказов может требоваться полное имя покупателя (т. е. сцепленный набор всех частей имени). Такая структура данных обычно используется в корреспонденции и документации, связанных с заказом, в строке "Address To" ("Кому") информации по доставке заказа, и для контактов с покупателями. Что касается системы G-CRM или маркетинговых приложений, то для них может потребоваться очень гибкая структура, позволяющая использовать как имя целиком, так и отдельные части полного имени. Здесь обработка может включать сортировку и группировку данных о покупателе по составным частям полного имени (рис. 8.12).

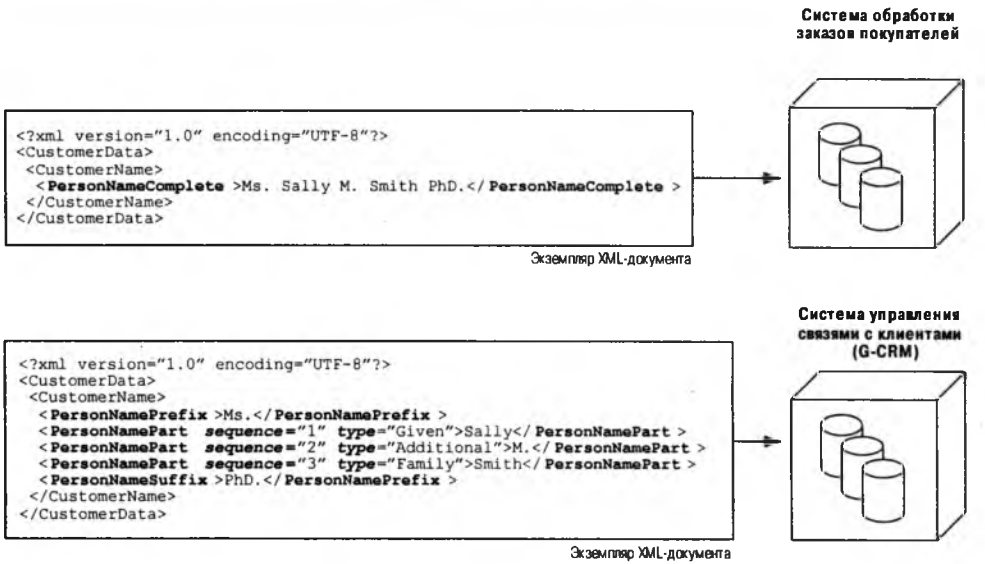


Рис. 8.12. Различные XML-транзакции, содержащие данные о полном имени человека

Чтобы удовлетворить обоим типам обработки имени, повторноиспользуемая W3C XML-Схема должна включать в себя две разные структуры имени. Одна из структур, определенных в схеме, – отдельный элемент данных, предназначенный для всего имени целиком. Другая – гибкая структура для отдельных частей имени. Если обе эти структуры определены как опциональные, а не как обязательные, получающуюся схему можно использовать для описания и ограничения разнообразных XML-документов и транзакций (листинг 8.1).

Листинг 8.1 W3C XML-Схема для разных форм данных о полном имени человека

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="CustomerData">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="CustomerName"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="CustomerName">
    <xs:complexType>
      <xs:sequence>
```

```
      <xs:element ref="PersonNameComplete" min0ccurs="0"
       max0ccurs="1"/>
      <xs:group ref="PersonNameParts" min0ccurs="0"
       max0ccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="PersonNameComplete"
 type="PersonNameCompleteTYPE"/>

<xs:group name="PersonNameParts">
  <xs:sequence>
    <xs:element name="PersonNamePrefix" type="PersonNamePartTYPE"
     min0ccurs="0"/>
    <xs:element ref="PersonNamePart" min0ccurs="1"
     max0ccurs="unbounded"/>
    <xs:element name="PersonNameSuffix" type="PersonNamePartTYPE"
     min0ccurs="0"/>
  </xs:sequence>
</xs:group>

<xs:element name="PersonNamePart">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="PersonNamePartTYPE">
        <xs:attribute name="sequence" use="required"
         type="xs:byte"/>
       <xs:attribute name="type" type="PersonNamePartTypeCODE"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:simpleType name="PersonNameCompleteTYPE">
  <xs:restriction base="xs:string">
    <xs:maxLength value="80"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="PersonNamePartTYPE">
  <xs:restriction base="xs:string">
    <xs:maxLength value="20"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="PersonNamePartTypeCODE">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Family"/>
    <xs:enumeration value="Last"/>
    <xs:enumeration value="Given"/>
    <xs:enumeration value="First"/>
    <xs:enumeration value="Surname"/>
    <xs:enumeration value="Additional"/>
```

```
        <xs:enumeration values"Middle"/>
        <xs:enumeration value="Religious"/>
        <xs:enumeration value="Other"/>
      </xs:restriction>
    </xs:simpleType>

</xs:schema>
```

Вместе с прототипом XML-документа, потенциально пригодным для разных вариантов использования и разных вариантов обработки, можно определить подходящие структурные модели XML и архитектурные формы контейнера. Элемент, предназначенный для всего имени целиком, в сочетании с набором элементов для отдельных частей имени, хорошо вписывается в вертикальную структурную модель. Элементы для частей имени должны также включать атрибуты для указания порядка следования (предопределенного порядка, в котором должны следовать части имени), а также атрибут указания вида данной части имени, что характерно для горизонтальной структурной модели. Возможность повторного использования структуры полного имени в разных контекстах (например, для обработки заказов покупателей, контактов с покупателями, служащими) подразумевает, что эта W3C XML-Схема хорошо отвечает компонентной структурной модели. Кроме того, типы данных и допустимые значения элементов имени могут также стать стандартами метаданных предприятия, реализуемыми как структурные модели в других модульных компонентах. В целом, схема для полного имени человека – прекрасный кандидат для гибридной структурной модели, сочетающей в себе лучшие характеристики всех прочих типов моделей.

Приложениям по управлению связями с клиентами часто требуется выполнить сортировку, группирование и отбор данных о клиентах, на основе имен клиентов. Изза того что полное имя можно указать в разных формах, для обработки этого типа не подходит использование элемента, содержащего полное имя, как единое целое. Префиксы и суффиксы имени редко используются для сортировки и могут осложнить группирование сходных имен. Поэтому для префиксов и суффиксов подойдут жесткие формы контейнеров, с конкретными именами. Разделение имени на составные части позволяет эффективно использовать данные о полном имени для сортировки, группирования и отбора. Обработка имен покупателей из других стран, у которых количество, порядок и вид частей полного имени может быть самым разным, требует применения абстрактной формы контейнера, способной динамически расширятся и сжиматься в соответствии с культурными различиями в форматах имени (рис. 8.13).

Нестандартные типы данных и списки стандартных значений кода предоставляют возможность для дальнейшей декомпозиции схемы полного имени. Подсхемы, представляющие стандартную для предприятия структуру полного имени, типы данных, для описания полного имени, и применяемые здесь значения стандартного кода – все это прекрасные кандидаты на повторное использование (рис. 8.14).

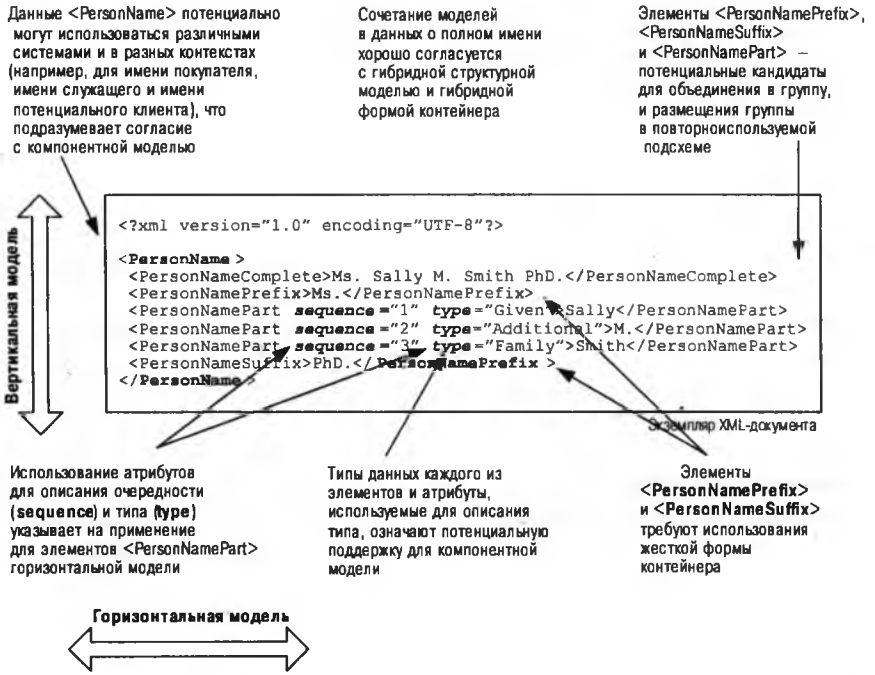


Рис. 8.13. Структурные модели и формы контейнеров для данных о полном имени человека

Элемент <PersonNameComplete> (полное имя целиком) определяется как имеющий строковый тип данных с ограничением на максимально допустимую длину. Элементы <PersonNamePrefix> (префикс имени), <PersonNamePart> (часть полного имени), и <PersonNameSuffix> (суффикс имени), также определяются как строки, но с меньшей максимально допустимой длиной. Все типы данных для этих элементов определены как простые типы (simpleType) и эти определения включены в подсхему, содержащую все типы данных для данных о полном имени. Атрибут "type" элемента <PersonNamePart> описан как принимающий значения из перечисления содержащего стандартные коды. Это перечисление определено в подсхеме и в него входят все допустимые значения видов частей полного имени. В результате имеем набор из трех схем-компонентов. Стандартная структура для полного имени использует по ссылке схему типов для полного имени и схему с кодами видов частей полного имени. Каждую из этих схем можно использовать как в целом, так и частями.

- Структурная схема полного имени человека (листинг 8.2).
- Схема с типами данных для полного имени (листинг 8.3).
- Схема с кодами видов частей полного имени (листинг 8.4).

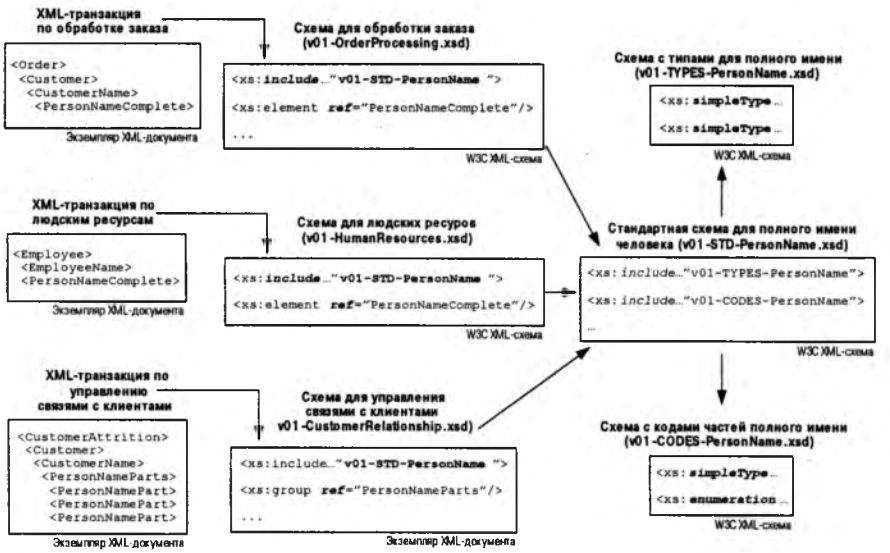


Рис. 8.14. Ссылки на подсхемы-компоненты

Листинг 8.2 W3C XML-Схема для структуры полного имени человека

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:include schemaLocation="v01-TYPES-PersonName.xsd"/>
  <xs:include schemaLocation="v01-CODES-PersonName.xsd"/>

  <xs:element name="PersonNameComplete"
   type="PersonNameCompleteTYPE"/>
  <xs:element name="PersonNameParts">
    <xs:complexType>
      <xs:seguence>
        <xs:group ref="PersonNamePartsGroup"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:group name="PersonNamePartsGroup">
    <xs:sequence>
      <xs:element name="PersonNamePrefix"
       type="PersonNamePartTYPE" min0ccurs="0"/>
      <xs:element ref="PersonNamePart" min0ccurs="1"
       max0ccurs="unbounded"/>
      <xs:element name="PersonNameSuffix"
       type="PersonNamePartTYPE" min0ccurs="0"/>
    </xs:sequence>
  </xs:group>
```

```
<xs:element name="PersonNamePart">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="PersonNamePartTYPE">
          <xs:attribute name="sequence"
           use="required" type="xs:byte"/>
          <xs:attribute name="type" type="PersonNamePartTypeCODE"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

Листинг 8.3 W3C XML-Схема с типами данных для полного имени

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:simpleType name="PersonNameCompleteTYPE">
    <xs:restriction base="xs:string">
      <xs:maxLengt value="80">
    </xs:restriction>
  <xs:simpleType>

  <xs:simpleType name="PersonNamePartTYPE">
    <xs:restriction base="xs:string">
      <xs:maxLengt value="20">
    </xs:restriction>
  <xs:simpleType>

</xs:schema>
```

Листинг 8.4 W3C XML-Схема с кодами видов частей полного имени

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs=
"http://www.w3.org/2001/XMLSchema">

  <xs:simpleType name="PersonNamePartTypeCODE">
    <xs:restriction base="xs:string">
      <xs:enumeration value="Family"/>
      <xs:enumeration value="Last"/>
      <xs:enumeration value="Given"/>
      <xs:enumeration value="First"/>
      <xs:enumeration value="Surname"/>
      <xs:enumeration value="Additional"/>
      <xs:enumeration value="Middle"/>
      <xs:enumeration value="Religious"/>
      <xs:enumeration value="Other"/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

Синтаксис ссылки на схему-компонент

Для повторного использования некоторой W3C XML-Схемы на нее необходимо сослаться из другой, использующей её схемы. Как уже отмечалось, работа ссылки на схему концептуально похожа на работу синтаксиса включения Copybook языка COBOL. Когда основная W3C XML-Схема используется для проверки документа (верификации), синтаксический анализатор определяет местонахождение ресурса–подсхемы, на которую указывает ссылка, включает в основную схему объявления из подсхемы, а затем уже разрешает ссылки на отдельные конструкции. Уровень вложенности ссылок на подсхемы не ограничен. Подсхема может ссылаться на другую подсхему, а та – на следующую и т. д. Не существует задокументированного ограничения на уровень вложенности, однако не стоит в этом особенно усердствовать. Большая вложенность может замедлить процесс синтаксического анализа. Синтаксис W3C XML-Схемы для ссылки на схему (подсхему) может иметь одну из трех синтаксических форм:

- Включение.
- Переопределение.
- Импорт.

Рекомендация. *Когда основная (ссылающаяся W3C XML-Схема) имеет единый контекст (т. е. пространство имен), для ссылки на определенную вовне подсхему лучше использовать включение ("include").*

Когда основная W3C XML-Схема имеет единый контекст, для ссылки на подсхему лучше использовать включение ("include"). В этом случае включаемая схема становится частью общего пространства имен основной (ссылающейся) схемы (мастер-схемы). Когда основная W3C XML-Схема складывается из нескольких контекстов и ссылка на требуемую подсхему должна относиться только к одному из этих контекстов, более подходит импорт ("import"). Синтаксис импорта нацелен на определенный контекст и использует для обеспечения уникальности контекста пространство имен. Синтаксис переопределения ("redefine") похож на синтаксис включения. Однако переопределение не только включает в мастер-схему подсхему, на которую указывает ссылка, но и позволяет изменить отдельные адресуемые ("ref=...") конструкции. Но независимо от выбранного синтаксиса, ссылки на требуемые конструкции подсхемы должны быть выполнены и разрешены.

Синтаксис включения сочетает ссылку на имя и местонахождение требуемой подсхемы с одной или более внутренних ссылок на компоненты, объявленные в подсхеме (рис. 8.15). Синтаксис W3C XML-Схемы для включения схемы прост и интуитивно понятен. В основной (ссылающейся) W3C XML-Схеме используется элемент "include" (<xs:include>). Атрибут "schemaLocation" этого элемента содержит имя и местонахождение адресуемой подсхемы. Значением атрибута "schemaLocation" может быть или относительное местоположение ресурса, или абсолютное местоположение. *Относительное местоположение ресурса* требует, чтобы адресуемая подсхе-

ма размещалась в том же месте или каталоге, что и основная (ссылающаяся) схема. *Абсолютное местоположение ресурса* содержит не только имя ресурса (адресуемой подсхемы), но также идентифицирует место размещения подсхемы (например, имя сервера, каталог, путь к файлу, или аналогичную информацию о местоположении подсхемы). В процессе верификации, синтаксический анализатор распознает элемент "include", выделяет атрибут "schemaLocation", ищет указанный ресурс и включает содержимое адресуемой подсхемы в основную схему.

Рис. 8.15. Синтаксис включения W3C XML-Схемы

Хотя синтаксис включения W3C XML-Схемы является эффективным способом повторного использования схем по ссылке, здесь следует соблюдать осторожность. Если содержимое и контекст адресуемой подсхемы неизвестен, существует вероятность включить из адресуемой подсхемы неприменимые конструкции. Кроме того, адресуемая W3C XML-подсхема может, в свою очередь ссылаться на другие подсхемы и т. д. Все это может привести к неэффективной для синтаксического анализатора цепочке сборки и обработки.

Синтаксисы W3C XML-Схемы для импорта и переопределения похожи на синтаксис включения, хотя итоговые методы разрешения ссылок отличаются. Синтаксис импорта требует наличия в основной схеме элемента "import" ("<xs:import>"). В этот элемент входят атрибуты "schemaLocation" и "namespace". Атрибут "schemaLocation" указывает местоположение адресуемой подсхемы. Атрибут "namespace" определяет контекст. Синтаксис переопределения включает элемент "redefine" ("<xs:redefine>") с атрибутом "schemaLocation". Как и в синтаксисах включения и импорта, этот атрибут указывает местоположение и имя адресуемой подсхемы. Кроме того, синтаксис переопределения позволяет некоторые контейнеры и конструкции определить иначе (рис. 8.16).

Синтаксис импорта W3C XML-схемы

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:import schemaLocation="здесь-указывается-адрес-схемы"
        namespace="здесь-указывается-пространство-имен"/>


</xs:schema>
```

W3C XML-схема

Синтаксис импорта W3C XML-Схемы

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <xs:redefine schemaLocation="здесь-указывается-адрес-схемы"
  ...   переопределения сложных типов, простых типов и групп
  ...
  ...

 <xs:redefine>

</xs:schema>
```

W3C XML-схема

Рис. 8.16. Синтаксис импорта и переопределения W3C XML-Схем

Далее описаны важные виды деятельности, связанные с проектированием метаданных и архитектуры данных, которые необходимы для создания гибких, расширяемых и повторноиспользуемых W3C XML-Схем. Затем мы узнаем, каким образом проектировщики данных могут решать стоящие перед ними задачи и как их деятельность интегрируется в процесс разработки приложений.

Глава 9

Проектирование и разработка для проектировщиков данных

До сих пор в центре обсуждения находилось применение принципов моделирования и проектирования к прототипам XML и, что еще более важно, к W3C XML-Схемам. В дополнение к рассмотренным подходам, необходимо рассмотреть процесс проектирования и разработки схемы. Благодаря быстрому принятию языка XML сообществом разработчиков приложений, проектирование и разработка W3C XML-Схем до сих пор ведется по наитию (бессистемно). После того, как принято решение об использовании XML, итоговые структуры и ограничивающие схемы зачастую генерируют из объектно-ориентированной модели, такой как диаграмма классов. Однако сообщество разработчиков приложений может не иметь опыта или времени для присоединения к данным строгих характеристик метаданных, или для применения необходимых стандартов данных. К тому же, может отсутствовать формальное определение зоны ответственности за проектирование и разработку схемы, что ведет к отсутствию строгих метаданных и соответствующей практики проектирования.

Фундаментальное определение и возможности XML необходимо принимать во внимание в ходе любого процесса проектирования и разработки. Повторим, что XML – язык описывающих метаданных, и W3C XML-Схемы обеспечивают способ наложения ограничений на содержимое XML, в соответствии со структурой, организацией и правилами метаданных. Для каждого практика должно быть очевидно, что эффективная разработка схемы требует знаний и опыта в таких дисциплинах проектирования, как метаданные и все, что с ними связано. Без концентрации на метаданных и архитектуре данных мы получим засилье нестандартных и сделанных на скорую руку XML-транзакций и схем, что приведет к возрастанию различий в данных, сложностям в интеграции, и неудачам при попытках снизить затраты на соответствующие технологии.

С другой стороны, проектировщик данных должен также осознавать, что XML-документ (т. е. "реализация", содержащая значения данных) и создается, и используется прикладными программами. Иерархическая природа XML-документа требует для успешной навигации и обработки определенного уровня опыта в разработке приложений. Этот опыт обычно находится вне сферы проектировщика данных и принадлежит сообществу разработчиков программ. Кроме того, существуют потенциальные осложнения, связанные с возможностями и мощью XML. Как уже отмечалось, использование богатого описательного стандарта именования может значительно увеличить размер XML-документа (мы помним, что "теги" XML-документа передаются вместе с данными). Хотя технологии, связанные с XML, постоянно совершенствуются, применение XML-транзакций и процессов в корпоративных приложениях с ограниченным временем реакции, требует адекватной производительности. Эффективный процесс проектирования и разработки схемы должен заимствовать опыт у традиционных методологий разработки, адаптируя лучшие подходы современных процессов (например, объектную ориентацию и итеративность), и обеспечивать сотрудничество между участвующими в разработке сторонами, а не отчетливое разделение на зоны ответственности. Сообщество проектировщиков данных и сообщество разработчиков программ вместе могут обеспечить необходимый для разработки опыт и уверенность в приемлемой производительности.

Процесс проектирования и разработки

Процесс проектирования и разработки объединяет несколько задач, которые заимствованы у традиционных процессов разработки. Сначала процесс проектирования схемы движется к цели, совершая набор определенных шагов, напоминающих методологию водопада (каскадное проектирование). Однако для того чтобы добиться утверждения и бизнес сообществом, и технологическим сообществом предприятия, этот процесс также включает в себя ускоренную разработку приложения, с фокусом на прототипирование, итеративное создание версий, и верификацию (рис. 9.1).

Начальные задачи по проектированию и разработке XML включают определение требований данных и функциональных требований. Деятельность по определению, документированию и проверке требований фундаментально тот же самый процесс, который используется в других проектах по разработке приложений. Также важно определить источники и получателей данных. Именно эта комбинация требований и определения источников и получателей данных, управляет начальной стадией разработки прототипа XML-документа.

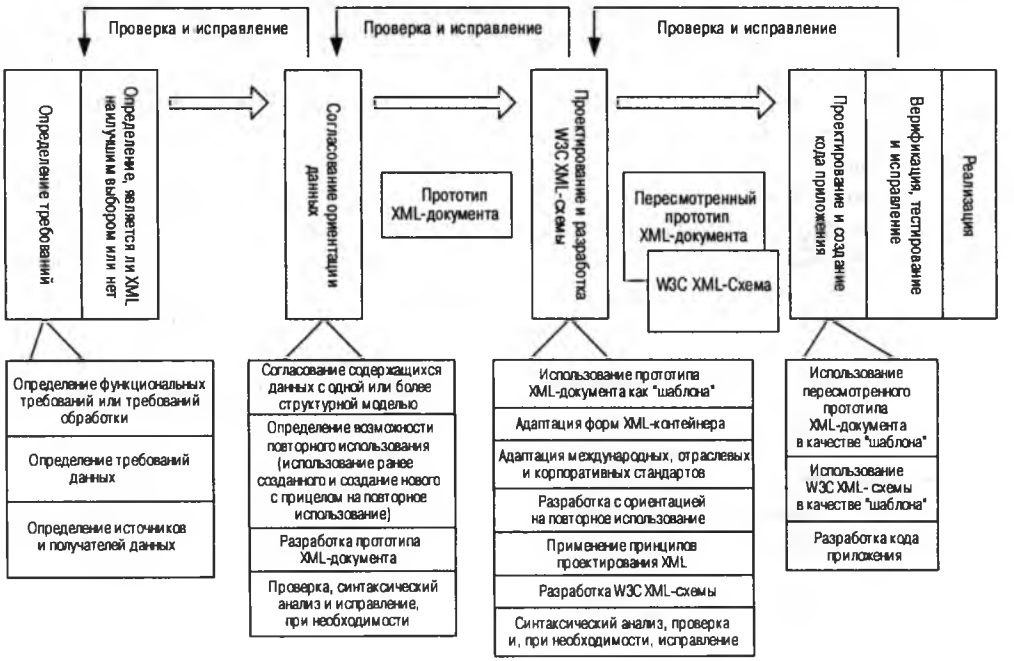


Рис. 9.1. Задачи и процессы проектирования XML

Из-за того, что XML чаще всего будет использоваться для описания документа, транзакции или сообщения, подразумевается "обмен" или "движение" информации между источником и одним или несколькими получателями. Источниками данных для XML-транзакции могут быть: ключевые пользовательские элементы, выборка из базы данных, логика приложения, события, или другие транзакции. В качестве получателей XML-транзакции чаще всего выступают: прикладные программы, обрабатывающие или использующие данные из транзакции; прикладные программы, которые публикуют или распространяют данные из транзакции; прикладные программы, представляющие эти данные для потребителей в наглядном виде (презентация); а также прикладные программы, сохраняющие данные из транзакции.

Когда получателем данных XML-транзакции выступает система хранения информации (например, база данных или система управления файлами), следует иметь ввиду ряд важных обстоятельств. В зависимости от характера использования транзакции, содержащиеся в транзакции данные представлены с использованием той или иной формы и синтаксиса XML. Хранилищем данных, представленных с помощью синтаксиса XML (т. е. посредством описательных контейнеров и в соот-

ветствии с иерархической структурой XML) является, обычно, простой файл, база данных, разработанная специально для хранения XML, или база данных, "допускающая" взаимодействие с XML. Когда XML-структура и содержащиеся данные не подвержены частым изменениям, хранение данных, используя синтаксис XML, – вполне приемлемый подход. Однако, благодаря нынешнему состоянию технологий баз данных и расширений для работы с XML, такие ситуации следует тщательно взвешивать. Может оказаться более предпочтительным опереться на возможности и преимущества традиционных баз данных, включающих поддержку XML и расширения для извлечения данных из XML-документов, нежели хранить данные в виде структуры формата XML. Ожидается, что со временем базы данных, созданные специально для XML, и утилиты доступа к данным будут предоставлять многие из тех возможностей, которыми обладают сегодняшние реляционные и объектно-ориентированные базы данных. Однако в настоящее время в их отношении следует соблюдать осторожность.

Существует одно очевидное отличие от традиционных методологий разработки – определение XML, как наиболее подходящей технологии для описания данных транзакции, возможно уже на ранних стадиях разработки. Хотя XML – мощный и эффективный язык метаданных, существуют сценарии, когда XML не является наиболее подходящим выбором. Традиционные методологии обычно не определяют выбор отдельной технологии, пока процесс разработки не зайдет достаточно далеко. Процесс выбора могут осложнять такие характеристики, как объемы, производительность, вид прикладной обработки, а также частота обмена. Таким образом, необходим некоторый метод ранней проверки того, является ли XML оправданным кандидатом на использование в проекте. XML можно проверить на пригодность с помощью контрольного списка критериев или с помощью более строгого набора оцениваемых критериев.

Возможно, наиболее значительным аспектом процесса разработки XML является создание прототипа XML-документа[1]. Процесс создания прототипа напоминает разработку модели данных. Прототип легче конструировать, заполнять образцами данных и представлять в наглядной визуальной форме, которая проще воспринимается участниками проекта. Ключевые виды деятельности по разработке прототипа включают:

- определение требований, диктуемых данными;
- согласование требований, диктуемых данными с функциональными требованиями;
- согласование требований, диктуемых данными со структурной моделью XML;
- определение возможностей для повторного использования (как внутри, так и вне данного проекта или системы);

[1] Bean J. XML Globalization and Best Practices. Active Education, Colorado, U.S., 2001. – *Примеч. авт.*

- разработка структуры прототипа;
- заполнение структуры прототипа образцами данных, как можно более близкими к итоговой реализации;
- рецензирование и внесение исправлений.

Прототип XML-документа представляет собой приемлемое отражение структуры, которая будет реализована. Он становится "шаблоном" для разработки соответствующей схемы. Структурная модель и архитектурные формы контейнера применяются в ходе процесса создания прототипа. Характеристики содержащихся данных, в сочетании с видом требуемой для XML-транзакции обработки, управляют выбором наиболее приемлемой формы контейнеров. Применение архитектурных форм контейнера помогает претворять в жизнь архитектурные принципы гибкости и повторного использования. Важна также согласованность со стандартами данных и разработкой, ориентированной на повторное использование. По возможности строгое применение стандартов данных поможет обеспечить высокий уровень качества данных и ограничить количество преобразований или трансляций транзакции. Следует определить, какие из существующих, стандартных для предприятия W3C XML-Схем – компонентов можно применить для описания данных в разрабатываемом проекте и использовать их при помощи ссылки (повторное использование). Это не только способствует продвижению стандартов, но и снижает затраты на разработку, поскольку предотвращает создание дополнительных, новых схем. При работе над новыми W3C XML-Схемами следует, по мере возможности, создавать их с прицелом на повторное использование в будущем (разработка, ориентированная на повторное использование).

Затем W3C XML-Схемы проверяют, примеряя их к прототипу XML-документа. Кроме этого, схемы необходимо обсудить с разработчиками приложений, чтобы убедиться, что они интуитивны, гибки и обеспечивают удобную навигацию. Сколь угодно гибкие подходы к разработке, использованные при создании схем, мало что дадут, если итоговый XML-документ нельзя будет легко обрабатывать и использовать. Для обеспечения наиболее эффективного соответствия данным и функциональным требованиям, структура прототипа XML-документа подвергается ревизиям. При принятии прототип XML-документа, итоговые W3C XML-Схемы и функциональные требования управляют разработкой логики приложения.

В описанном процессе отсутствует упоминание традиционных видов деятельности по моделированию данных, проектированию базы данных и моделированию объектов. Это сделано не потому, что они не нужны, просто к ним обращаются при решении соответствующих задач разработки. Как было сказано в предшествующих главах, существует ряд общих видов деятельности между традиционной практикой проектирования данных и проектированием и разработкой W3C XML-Схем. Процессы моделирования данных и проектирования базы

данных предназначены для удовлетворения требований, относящихся к выборкам данных, извлекаемым из и сохраняемым в архитектурах баз данных. Модели процесса и объекта формализуют отношения функция-данные и облегчают разработку логики программы. Как было определено, процесс разработки XML включает ряд задач. Проблема в том, как эффективно использовать ресурсы предприятия и имеющийся опыт для решения этих задач.

Область ответственности проектировщика данных

Одна из спорных и неразрешенных полностью тем в XML-приложениях: разделение зон ответственности между проектировщиком данных и разработчиком приложения. Посылка данной книги заключается в том, что XML и W3C XML-Схемы, фундаментально, есть данные и описывающие их метаданные. XML-документ является файлом, содержащим значения данных. Этот файл представляет документ, транзакцию или сообщение, которые создаются прикладной программой или аналогичным служебным процессом. W3C XML-Схема описывает правила и ограничения, применяемые к ссылающемуся на нее XML-документу. Аналогично, база данных содержит значения данных и заполняется и обрабатывается одной или несколькими прикладными программами. Характеристики метаданных и правила для базы данных определяются моделью, на основании которой эта база была сгенерирована, и управляющим системным каталогом реализуемой архитектуры данных. В обоих случаях существует очевидное ударение на данных и метаданных, которые традиционно являются областью ответственности проектировщика данных.

Исторически, во многих организациях, обязанности по моделированию данных и проектированию баз данных возложены на проектировщика данных. На долю разработчиков досталась разработка логики программы по извлечению, вставке, изменению, удалению, обмену и обработке данных. Проектирование и разработка W3C XML-Схем – процессы, аналогичные традиционным обязанностям проектировщиков данных, но более эффективный процесс проектирования и разработки предполагает сотрудничество с разработчиками программ. Ведь когда этот процесс завершен и технология реализована, XML-документы, являющиеся "конечной продукцией", создаются, обслуживаются и обрабатываются одной или несколькими прикладными программами.

Некоторые организации могут сделать выбор в пользу передачи разработки прототипа XML-документа в руки разработчиков программ. И в некоторых случаях такая практика оправдана. Однако прототип XML-документа также сочетает в себе две формы метаданных: именование и структура. Именование применяется для формализации имен тегов атрибутов и элементов XML, а структура представляет организацию контейнеров с данными и их отношения.

Рекомендация. *Рекомендуется вести процесс проектирования и разработки W3C XML-Схемы совместными усилиями. И проектировщик данных, и разработчик должны вместе принимать участие в создании прототипа XML-документа и соответствующих W3C XML-Схем.*

Таким образом, предлагается осуществлять разработку прототипа XML-документа объединенными усилиями проектировщика данных и разработчика программ. При этом проектировщик данных будет следить, чтобы были добавлены необходимые контейнеры, в полном соответствии с требованиями данных, и эти контейнеры были согласованы с известными источниками и получателями данных. Проектировщик данных также будет организовывать наборы взаимосвязанных контейнеров данных и разрабатывать описательные имена тегов для каждого из контейнеров. Кроме этого, проектировщик данных будет устанавливать возможность применения ранее определенных стандартов (форма повторного использования) и создания новых стандартных структур с прицелом на их повторное использование в будущем. Разработчик приложения будет следить, чтобы прототип XML-документа получился интуитивным и простым в навигации для прикладной программы и чтобы его можно было обрабатывать в соответствии с поставленными функциональными требованиями. Разработчик также обеспечивает руководство в областях, связанных с производительностью и сложностью, тогда как проектировщик данных фокусируется на гибкости, стандартах и повторном использовании.

Когда процесс проектирования и разработки доходит до разработки W3C XML-Схем, проектировщик данных получает более широкие права. Здесь важно определение и формализация правил и характеристик метаданных для каждого из контейнеров прототипа XML-документа. Эти правила реализуются как комбинация типов данных, фасетов и отношений между элементами (например, сложные типы [complexType] и группы [group]). Чтобы разработать W3C XML-Схему, проектировщик данных должен иметь достаточный уровень знаний о синтаксисе W3C XML-Схемы. Если предприятие использует инструменты проектирования и разработки, допускающие работу с XML, вполне вероятно, что первоначальные, черновые схемы могут быть сгенерированы с помощью программного инструмента или утилиты (как при нисходящей разработке), которая сформирует основной необходимый синтаксис. Проектировщику данных после этого предстоит, по необходимости, усовершенствовать, дополнить и изменить эту схему. Разработчик программ просматривает правила и ограничения метаданных с целью определения, какая прикладная логика потребуется в процессе верификации документа с помощью синтаксического анализатора и какие действия следует предпринять в программе, реагируя на сообщения синтаксического анализатора об ошибках.

Проектировщик данных также будет применять необходимые принципы проектирования: структурные модели, формы контейнеров, и практику повторного использования. В зависимости от сложности создаваемой схемы, применение архитектурных форм контейнеров может привести к последующим изменениям структуры прототипа XML-документа. В этом случае разработчик приложений вновь изучает получившуюся структуру, чтобы убедиться, что модифицированный XML-документ все еще интуитивен и прост в навигации. В области повторного использования, проектировщик данных добавляет ссылки на применимые стандартные подсхемы. Кроме того, он изучает возможности выделения вновь создаваемых структур во внешние подчиненные W3C XML-Схемы с целью их дальнейшего повторного использования. Схемы и подсхемы проходят первоначальное тестирование путем верификации синтаксическим анализатором прототипа XML-документа.

После получения W3C XML-Схем, основная ответственность за дальнейшую разработку переходит к разработчику приложений (т. е. разработка прикладной логики). Разработкой логики программ управляет комбинация из прототипа XML-документа, W3C XML-Схем и функциональных требований. Если в ходе этой работы потребуется внести изменения в прототип XML-документа или в W3C XML-Схемы, к изучению этого вопроса и ревизии подключается проектировщик данных (рис. 9.2).

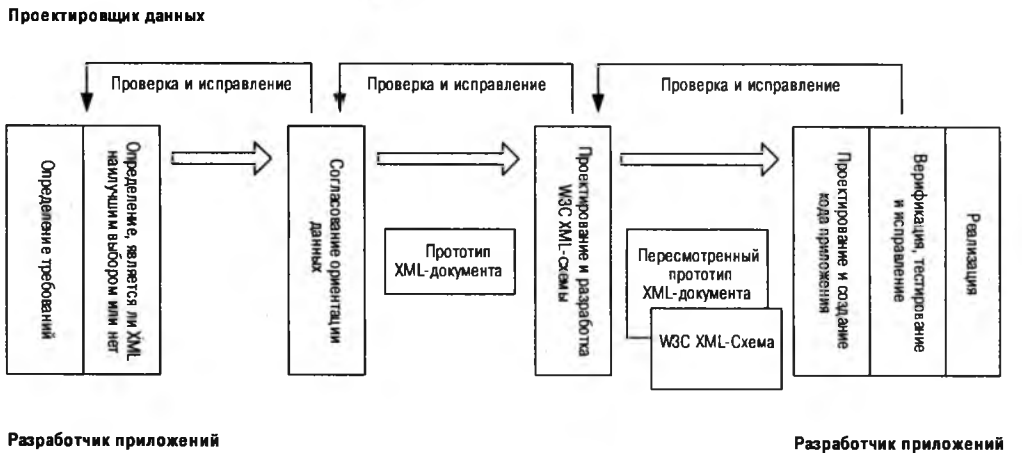


Рис. 9.2. Зоны ответственности в задачах проектирования и обработки XML

Итак, рекомендуемый процесс проектирования и разработки XML на ранних этапах возлагает на проектировщика данных большую степень ответственности и сводит роль разработчика к комбинации консультанта, советника и пользователя. На более поздних стадиях работы главенство переходит к разработчику (особенно

в части задач по разработке прикладной логики). Однако на протяжении всего процесса и проектировщик данных, и разработчик выступают как сотрудники, обозреватели и консультанты. Такое сотрудничество помогает обеспечить то, что XML-документ, создаваемый прикладной программой, будет эффективным, интероперабельным[2], гибким, повторноиспользуемым и завершенным. Это также обеспечивает реализацию знаний и опыта проектировщика данных в форме высоко стандартизированных и повторноиспользуемых W3C XML-Схем.

Вызовы сложности

Несмотря на наличие методологии или процесса разработки, существуют потенциальные сложности и риски, связанные с использованием XML. Как практик и специалист по метаданным, проектировщик данных применяет наилучшие решения для того, чтобы каждый XML-документ (документ, транзакция или сообщение) и каждая схема полностью отвечали требованиям проекта и были спроектированы способом, обеспечивающим гибкость, повторное использование и учет стандартов. Точно так же разработчик приложений проектирует бизнес-логику и разрабатывает прикладную программу в соответствии с функциональными требованиями проекта. Усилия разработчика обычно сосредотачиваются на функциональности, производительности и скорости разработки. Оба набора решений, подходов и целей направлены на достижение наилучшего возможного решения, отвечающего набору заданных требований. Однако характеристики используемой технологии, такой, как XML, могут также внести некоторый уровень сложности.

Тщательно спроектированные и разработанные XML-документы и соответствующие им W3C XML-Схемы немногого стоят, если их нельзя правильно и эффективно обработать. И наоборот, XML-документ и схема, созданные без концентрации на гибкости, повторном использовании и стандартах, мало пригодны (если вообще пригодны) для повторного использования, требуют повышенного обслуживания и могут привести к размножению неправильно представленных данных. Из того, что наиболее часто XML-документ используется для обмена, распространения или перемещения данных, следует настоятельная необходимость при его создании обеспечить применение наилучших подходов проектирования данных. Процессы проектирования и разработки W3C XML-Схемы должны осуществляться в сотрудничестве, и существует несколько областей, где разработчик приложений может предоставить необходимый совет и руководство.

---

[2] В оригинале – "interoperable" (буквально "имеющий возможность взаимодействовать"). В данном контексте означает пригодность для организации взаимодействия различных приложений. – *Примеч. ред.*

Сложность структуры и навигации

Одной из областей сложности и потенциального риска является структура XML-документа. Первоначально ударение делается на проектировании и разработке XML-документа, обладающего гибкостью (т. е. динамически расширяемого или сжимаемого, в соответствии с характеристиками содержащихся данных). Гибкость структуры XML-документа достигается комбинацией вложенности и абстракции (допускает вложение одноименных повторяющихся элементов в качестве дочерних элементов внутри одного родительского элемента). Родительский контейнер-элемент создает некую форму контекста или области видимости для вложенных в него дочерних контейнеров-элементов. При этом повторяющиеся дочерние элементы имеют абстрактное имя и наследуют контекст своего родительского элемента. Это эффективный архитектурный подход, позволяющий достичь изменчивости данных в зависимости от назначения XML-транзакции и вида ее обработки.

Рекомендация. *Как правило, наибольшая глубина вложенности элементов для содержания, ориентированного на транзакции, не должна превышать 10 уровней.*

В связи с этим возникает вопрос о допустимой глубине вложенности для XML-элементов. Спецификация XML (W3C) никак не ограничивает эту глубину, хотя могут существовать ограничения со стороны отдельных синтаксических анализаторов. В результате, можно создать XML-документ, имеющий 20, 30 или более уровней вложения дочерних элементов в родительские. Но при значительных уровнях вложенности сильно усложняется программная навигация по документу. Вложение повторяющихся XML-элементов – мощная техника, но пользоваться ею следует с осторожностью. Вложенность элементов должна оставаться в границах разумного. Как правило, наибольшая глубина вложенности для содержимого, ориентированного на транзакцию, не должна превышать 10 уровней.

Другая область, вносящая сложность в навигацию, связана с повторяющимися элементами. Если контекст повторяющегося элемента не может быть определен его положением в XML-документе (т. е. его родительским элементом), либо присвоенным ему именем, либо иными способами классификации (например, пространством имен), то такой элемент – источник неопределенности при обработке документа. Также, хотя схема позволяет ограничивать кардинальность (minOccurs и maxOccurs), не существует способа известить обрабатывающее документ приложение о том, сколько повторений элемента присутствует в данной реализации XML-документа. Рассмотрим XML-документ, содержащий несколько международных почтовых адресов. В каждом из адресов количество строк для улицы может меняться от одной до нескольких. Можно вызвать метод “getElementsByTagName” синтаксического анализатора и получить от него все адресные строки для улицы, но у этого метода не существует возможности описать, сколько именно адресных строк для улицы существует. Соответствующая прикладная логика должна уметь работать со всеми возможными вариантами.

Техника. *В случае, когда количество повторений некоторого элемента в группе внутри родительского элемента имеет существенное значение для логики навигации, добавьте на уровне родительского элемента атрибут, содержащий число повторений. Но позаботьтесь при этом ввести дополнительную логику для определения числа повторений и инициализации этим числом данного атрибута, а также логику последующего опроса и использования значения атрибута.*

Одним из полезных приемов проектирования является добавление в родительский элемент (содержащий повторяющийся дочерний элемент) атрибута для хранения текущего количества экземпляров повторяющегося дочернего элемента. Приложение, осуществляющее обработку документа, должно содержать логику для извлечения этого значения из атрибута и определения максимальной границы до того, как начнет навигацию по экземплярам повторяющегося элемента. В дополнение к этому, каждый повторяющийся элемент может включать определяемый пользователем атрибут для обозначения порядка следования экземпляра элемента внутри группы повторяющихся экземпляров. Порядок, в котором элементы расположены в XML-документе, должен быть предопределен. Однако могут существовать варианты следования или расположения, диктуемые характером обработки (рис. 9.3).

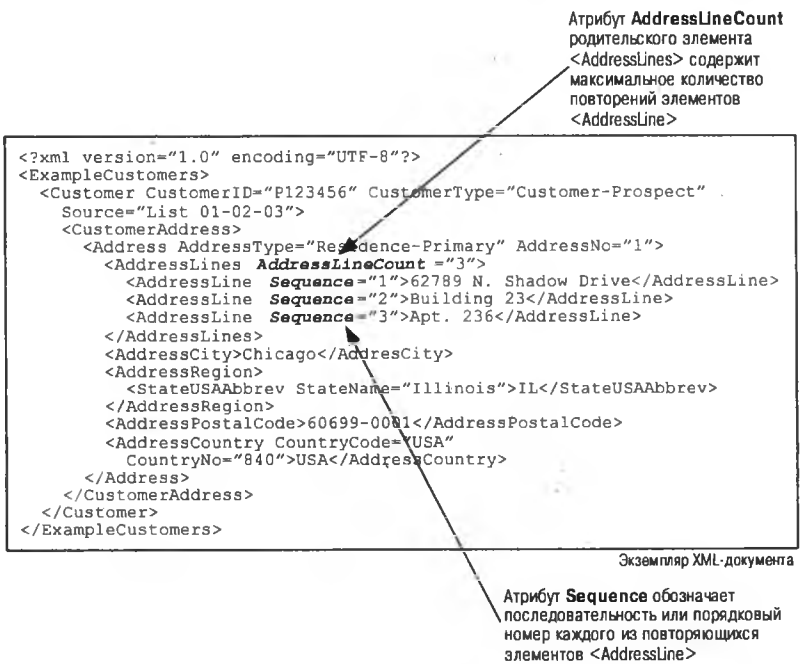


Рис. 9.3. Использование атрибутов помогает уменьшить сложность навигации

Использование определяемых пользователем атрибутов, как было показано в предыдущем примере, – это эффективное проектное решение. Однако и здесь существуют потенциальные риски и сложности. Во-первых, атрибут, описывающий число повторений повторяющегося элемента, – это форма метаданных (т. е. наибольшая степень кардинальности реализации). При таком подходе, обрабатывающее документ приложение должно включать логику опознания атрибута и опроса его значения. Кроме того, внесение подобного атрибута для каждого повторяющегося элемента увеличит общий объем документа, да и прикладной программы – за счет дополнительной логики по работе с этими атрибутами. Перед тем как решиться добавить такие атрибуты в прототип XML-документа, посоветуйтесь с разработчиком приложения. Разработчик должен понять движущие мотивы и определить, какая логика потребуется для заполнения и опроса атрибута.

Рекомендация. *Как правило, старайтесь избегать строго горизонтальных структурных моделей, за исключением случаев, когда существует ограничение на размер создаваемого XML-документа, или когда этот XML-документ не что иное, как простая выборка из реляционной базы данных.*

Другой потенциальной областью структурной сложности являются строго горизонтальные структурные модели. Горизонтальные структурные модели характеризуются преимущественным использованием атрибутов, а не элементов в качестве контейнеров для данных. Само собой, горизонтальная модель не полностью свободна от элементов, поскольку атрибуты не могут существовать сами по себе (XML-атрибут должен определяться внутри конкретного элемента). При синтаксическом разборе XML-документа контейнеры-атрибуты становятся дочерними узлами своего элемента-владельца. Концептуально, они также содержат данные, описывающие свойства узла-владельца. Из-за иерархической природы XML-документа, все атрибуты, определенные для некоторого элемента, логически являются "братьями". Они находятся на одном уровне иерархии.

Ограниченность атрибутов в том, что они не могут быть определены как повторяющиеся. В результате, все атрибуты, определенные для некоторого элемента-владельца, должны иметь уникальные (отличающиеся друг от друга) имена, и их нельзя добавить динамически, если потребуются новые контейнеры. Навигация среди большого числа атрибутов-"братьев" может представлять определенную сложность. Если обрабатывающее документ приложение выборочно обрабатывает разные атрибуты некоторого узла-элемента (а не все атрибуты узла подряд), оно должно быть осведомлено о назначении содержания каждого атрибута. Имя атрибута действует как форма идентификации, и обрабатывающее приложение должно выбирать узлы, соответствующие тем атрибутам, которые отвечают определенной бизнес-логике, или тем, которые отображаются на атрибуты определяющего класса объекта.

Если горизонтальная модель XML-структуры включает определенное количество атрибутов для хранения подобных или взаимосвязанных значений данных, и впоследствии потребуется добавить новый атрибут, в этом случае, скорее всего, придется вносить изменения в XML-документ, в W3C XML-Схему и в обрабатывающее приложение. Хотя каждый из атрибутов можно определить как необязательный (опциональный), такую структуру нельзя динамически расширить в соответствии с изменением данных. Кроме того, каждый из атрибутов должен иметь уникальное имя. У большинства синтаксических анализаторов есть методы, выдающие атрибуты и значения данных атрибутов, но эти методы не избавят обрабатывающее приложение от необходимости индивидуально обрабатывать (или обрабатывать в соответствии с классом) данные атрибута (рис. 9.4).

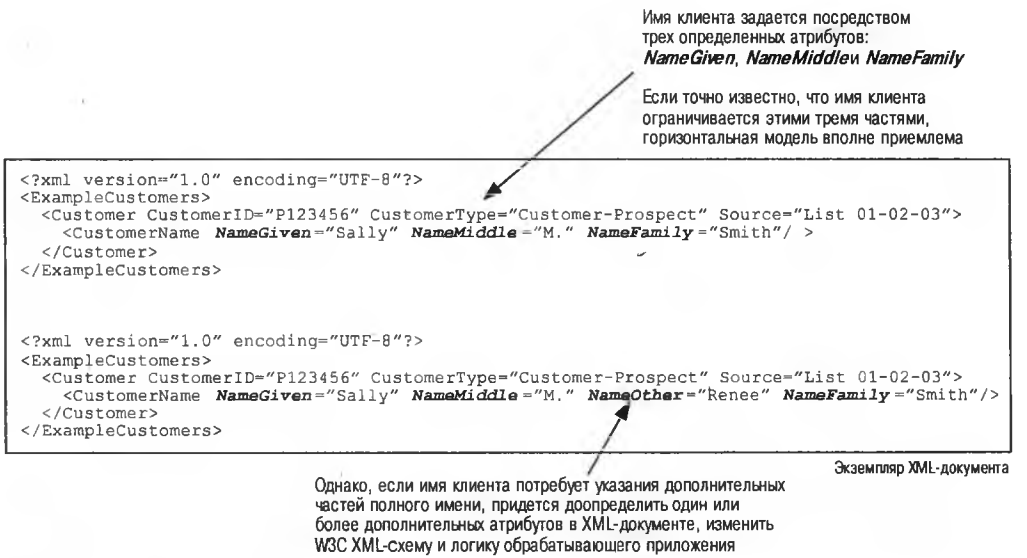


Рис. 9.4. Опасности атрибутов в подверженных расширению структурах, использующих горизонтальную модель

В качестве общей рекомендации: во избежании структурной сложности, проектируйте и разрабатывайте XML-документы и схемы так, чтобы они были простыми, интуитивными, легкими в навигации, гибкими, повторноиспользуемыми и страндартизированными.

Размер документа

Размер XML-документа представляет собой интересную проблему. Независимо от формы или протокола, большинство транзакций с данными перемещается по сети. Характеристики производительности сети могут сильно влиять на возможности обмена и обработки больших объемов данных. При росте общего размера транзакции (размера, определяемого количеством символов, или размера файла в байтах), время, необходимое для передачи и получения транзакции также растет.

Другой проблемой, связанной с размером XML-документа, является применение процесса синтаксического анализа. DOM-синтаксические анализаторы генерируют модель структуры XML-документа и его содержимое непосредственно в памяти. При увеличении размера документа растет размер памяти, необходимой для его обработки. Если размер XML-документа превышает размеры доступной памяти, синтаксический анализатор начинает использовать альтернативные возможности, например страничный обмен с дисковой памятью, а в некоторых случаях синтаксический анализ прерывается с ошибкой.

Одно из существенных преимуществ использования XML для описания содержимого документа, транзакции или сообщения, заключается в применении самоописывающих контейнеров с данными. В сочетании с богатым описательным стандартом именования, содержимое XML-документа становится интуитивно понятным. Однако самоописывающая природа XML имеет свою цену. Описательные имена элементов и атрибутов (теги) передаются в составе XML-документа, транзакции или сообщения, вместе со значениями данных. Если имена элементов и атрибутов становятся чересчур длинными, возрастает и общий размер XML-документа, транзакции или сообщения.

Рекомендация. *Общий размер XML-документа влияет на производительность сети и приложения, осуществляющего его обработку. Необходимо соотнести количество символов, приходящееся на имена элементов и атрибутов с количеством символов, приходящимся на "полезные" данные (содержимое). Если "коэффициент тег/данные" превышает приемлемый уровень, следует пересмотреть необходимость использования XML.*

Ограничение размера XML-документа не должно означать запрет на применение стандартов именования, приводящих к развернутым именам. Однако стандарты именования должны иметь оправданные ограничения на длину имен элементов и атрибутов. Техника сравнения эффективного стандарта именования с размером документа взвешивает количество символов всех имен элементов и атрибутов против количества символов "полезных" данных (содержимого). Если количество символов в именах чрезмерно по сравнению с количеством символов в значениях данных, то транзакция переносит существенно больше метаданных, чем данных. Эффективность применения XML в таком случае ставится под вопрос. Если соотношение имен контейнеров и данных превышает некоторый

порог, использование XML необходимо пересмотреть. Такое соотношение выражается "коэффициентом тег/данные". Очевидно, что число повторяющихся экземпляров внутри XML-документа, пропускная способность сети, и использование технологий сжатия, могут сильно влиять на допустимый размер документа. Также некоторые символьные объемы могут сильно меняться в зависимости от природы содержащихся данных и могут быть неизвестны вплоть до окончания разработки XML-документа и схемы. По возможности, следует использовать оценки и усреднения, сделанные на основе прототипа XML-документа.

Производные и избыточные данные

Традиционная практика проектирования данных уделяет значительное внимание процессу нормализации и избегает определения производных и избыточных данных. Существует несколько правил, управляющих разработкой модели нормализованных данных или базы данных, а также разные интерпретации этих правил. Первые три правила нормализации упоминаются и используются чаще всего. Первое правило нормализации можно интерпретировать так: "все неключевые атрибуты должны зависеть от ключа". Это правило означает, что каждый набор данных должен однозначно определяться уникальным идентификатором или первичным ключом. Второе правило нормализации гласит, что когда "ключ" состоит из нескольких частей (т. е. составной или агрегированный ключ), "неключевые атрибуты должны зависеть от всего ключа в целом". Второе правило нормализации говорит само за себя. Оно постановляет, что когда идентификатор или первичный ключ является комбинацией нескольких значений, ссылка на или зависимость от этого идентификатора, должна учитывать все составляющие его части. Третье правило нормализации гласит, что "не ключевые атрибуты должны находится в зависимости от ключа, полного ключа (всех частей составного ключа) и более ни от чего". Это правило подразумевает, что не ключевые данные не должны зависеть от не ключевых данных или идентификаторов других типов, отличных от первичного ключа.

Проще говоря, применение данных правил нормализации приводит к структуре данных, в которой:

- каждый элемент данных должен иметь формальную зависимость от идентификатора некоторого типа
- несколько элементов данных, представляющих идентификатор, всегда рассматриваются как единое целое
- данные, повторяющиеся в пределах некоторого контекста, будут перемещены в отдельные структуры с отношением, принимающим различные степени кардинальности.
- производные данные не просто разделены, они не должны определяться ни одним из элементов данных.

Как известно любому проектировщику данных, если данные предназначены для постоянного хранения (т. е. хранения в базе данных), соблюдение правил нормализации не просто важно, а критически важно для эффективного проектирования и обработки. Однако существуют специфичные для технологии исключения из нормализации. Например, как вы вероятно знаете, базы данных, разработанные для хранения большого количества элементов данных, со сложными отношениями и очень большими структурами данных, представляют потенциальную сложность в навигации и угрозу производительности. Как и для базы данных, правила нормализации, во многих случаях, можно применять для структуры ориентированного на транзакцию XML-документа. И здесь также существуют потенциальные исключения. Важнейшее отличие XML-транзакции от хранения данных в базе данных заключается в том, что ориентированные на транзакцию XML-документы используются для описания перемещаемых или обмениваемых данных, а не для постоянных (хранящихся) данных.

Проектировщик данных должен применять правила нормализации тогда, когда их использование принесет пользу. К примеру, наборы взаимосвязанных данных, содержащиеся в транзакции, должны также приписываться к идентифицирующему элементу. Когда такой идентификатор состоит из нескольких элементов данных (составной идентификатор), важно переносить отдельные элементы данных из этих частей идентификатора. Здесь также существуют исключения из строгого соблюдения правил нормализации. При обмене данными, ориентированными на транзакцию (что включает перемещение данных транзакции от источника к одному или нескольким приемникам) важным моментом становится производительность, зависящая от полосы пропускания сети и общего размера транзакции. Однако полоса пропускания сети и размер документа не должны быть единственными критериями для определения того, нужно ли применять правила нормализации к структуре XML-транзакции. Кроме этого, проектировщик данных должен учитывать общую сложность транзакции, предполагаемый способ ее использования и потенциальные риски от нарушения правил нормализации (риски, связанные с производительностью, архитектурой, повторным использованием, гибкостью и другими факторами). Проектировщик данных должен рассмотреть важность нормализации применительно к каждой части XML-транзакции, а не только к транзакции как единому целому.

Производные данные определяются как результат применения бизнес-правил или логики к одному или более элементам данных. Производные данные могут иметь несколько форм (агрегат, декомпозиция, производное). Наиболее общий подход описания полного имени человека заключается в том, чтобы использовать данные полного имени как набор составных частей имени, хранящихся в отдельных элементах данных (фамилия, имя, дополнительное имя и т. д.). Альтернатива этому – полное имя человека, являющееся результатом сцепления в единое целое

всех частей имени и хранимое в одном элементе данных – как раз является примером производных данных. Другим примером является полная стоимость покупки, когда налоги, отгрузка, транспортировка и страховка добавлены к цене изделия (рис. 9.5).

Производное полное имя человека

Comlete Person Name (полное имя)		Given (First) Name (Данное имя)			Additional (Middle) Name (Дополните-льное имя)			Family (Last) Name (Фамилия)
Sally M. Smith	=	Sally	+	Пробел	+ M.	+	Пробел	+ Smith

Производная полная стоимость покупки

Item Price (цена изделия)	14.32
Local Tax (налог)	1.43 (*10%)
Shipping (отгрузка)	5.95
Handling (транспортировка)	2.50
Insurance (страховка)	3.00
Total Price (полная стоимость)	27.20

Рис. 9.5. Примеры производных данных

Техника. *Для некоторых сценариев обработки, может иметь смысл определить в XML-транзакции производные данные. Однако здесь требуется тщательная оценка общего размера документа, сложность производных, и количество детализированных элементов данных.*

В случае электронной коммерции, транзакция может создаваться пользователем с помощью некоторой формы Web-интерфейса и проходить через серию приложений и обработок, с порождением одной или более дополнительных транзакций и ответов, циркулирующих между клиентом и бизнес-предприятием. В случае интеграции предприятия, транзакции перемещаются и обрабатываются приложениями внутри предприятия. В обоих случаях, безусловно, важна производительность. В зависимости от степени детализации, объемов и предполагаемого использования транзакции, может иметь смысл помещать в транзакцию и производные данные. Если можно один раз аккуратно создать производные данные на стороне источника и затем передать их нескольким получателям, приложения на стороне получателей смогут избежать дополнительной обработки и ответить быстрее.

Передача производных данных может принести пользу в разных ситуациях. Если несколько наборов данных, передаваемых в составе одно транзакции, предназначены для последовательной обработки, производные данные вполне уместны. Если одна и та же транзакция используется в нескольких обработках разного вида, может оказаться полезным наряду с детализированными элементами данных передавать и производные данные. В качестве примера можно привести XML-транзакцию, содержащую несколько экземпляров адресных данных, предназначенных для формирования списка рассылки и создания отчета по продажам. Если приложение, формирующее список рассылки, работает с именами и адресами в их полном виде, не применяя к ним никакой бизнес-логики (обработка заключается в извлечении данных имени и адреса и печати наклеек для почтовой рассылки), то наличие элемента данных, содержащего производное "полностью сформированное" имя клиента, может оказаться очень полезным. Если эта же транзакция предназначалась для повторного использования в разных прикладных процессах, которые сортируют и отбирают группы клиентов по фамилиям, в ней необходимо иметь отдельные части полного имени. Таким образом, XML-транзакция, предназначенная для повторного использования в различных обработках, должна включать и детализированные элементы данных – части полного имени, и производные данные – сцепленное в единое целое полное имя (рис. 9.6).

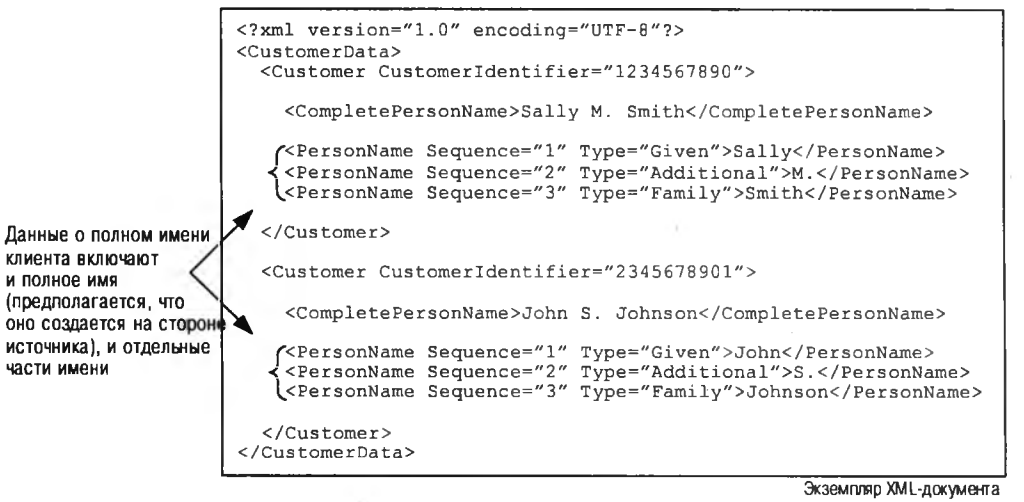


Рис. 9.6. Производные и избыточные XML-данные о полном имени

Данный пример описывает сценарий, в котором производные данные могут быть определены в одной, повторноиспользуемой структуре XML-транзакции, и обработаны приложениями разных типов. Однако производные и избыточные данные также могут являться причиной рисков и осложнений. Как уже отмечалось, предметом нашей озабоченности является общий размер XML-транзакции. Будучи по своей природе самоописывающим, каждый контейнер имеет теги с именем. Количество символов составляющих каждый тег увеличивает суммарный объем XML-транзакции. В результате, перенос дополнительных элементов или атрибутов, к тому же содержащих концептуально избыточные данные, может существенно увеличить размер транзакции, что может повлиять на производительность обмена и общей обработки XML-транзакции. Использование производных данных также подразумевает применение бизнес-правил или логики, и эти данные, в ряде случаев, не могут быть разлагаемы без более детальных элементов данных и логики, посредством которой они создавались. В результате, необходимо тщательно оценивать общий объем документа, сложность создания производных данных и количество детализированных элементов данных.

Итак, мы познакомились с языком XML в контексте документа или транзакции. К последним технологическим нововведениям относится концепция совместной обработки (collaborative processing), в которой XML также играет важную роль. К примеру, заслуживающему обсуждения, относятся Web-сервисы на основе XML.

Глава 10

Web-сервисы – введение в будущее

Web-сервисы относятся к недавним технологическим нововведениям во Всемирной паутине (WWW или просто Web). В общих чертах, Web-сервис – это прикладная программа, работающая на базе Web и имеющая определенный интерфейс, с помощью которого она получает некоторые запросы и, после выполнения необходимой обработки, возвращает ответы инициаторам запросов. Web-сервис не связан напрямую с каким-либо определенным запрашивающим приложением (т. е. Web-сервис слабосвязанный). В некотором отношении Web-сервис напоминает модель клиент-сервис, где Web-сервис играет роль сервиса. Однако Web-сервис является программой-антагонистом понятия "платформа", доступ к нему осуществляется из Web. Web-сервис обладает следующими характеристиками.

- Web-сервис поддерживает единые методы обращения к нему.
- Интерфейс Web-сервиса обычно не зависит от платформы.
- Способы внутренней реализации Web-сервиса обычно неизвестны инициатору запроса (клиенту).
- Функциональность, обеспечиваемая Web-сервисом, предполагает некоторый контекст, но не ограничена использованием лишь в каком-то одном приложении или виде приложений.

Как правило, но совсем необязательно, типичный (внешний) Web-сервис работает для всей Всемирной Паутины. Для обращения к нему может быть использован почти любой коммуникационный протокол или форма сообщения (т. е. Web-сервис является интероперабельным). В настоящее время наиболее распространенным протоколом в этой сфере является протокол передачи гипертекста (HTTP), но вовсе не обязательно использовать именно его. Фактически к Web-

сервису можно обратиться, используя протокол передачи файлов (FTP), вызов удаленных процедур (RPC) или аналогичные способы. Очевидное преимущество интероперабельности заключается в том, что приложения, использующие HTTP для передачи информации, могут быть легко адаптированы для переноса сообщений запросов и ответов Web-сервиса.

Интерфейс Web-сервиса также не связан с какой-либо одной платформой. Если Web-сервис определен как открытый (общедоступный), потенциально он может использоваться приложением любого вида, из любой точки Web. Поскольку в XML используется кодировка Unicode (а чаще ее подмножества UTF-8 и ASCII), он вполне подходит как способ описания интерфейса Web-сервиса и запроса, посылаемого при обращении к Web-сервису. В действительности, многие общедоступные Web-сервисы используют XML для описания провайдера Web-сервиса, характеристик сервиса, и интерфейса для взаимодействия с сервисом.

Внутренняя реализация Web-сервиса может быть осуществлена с использованием почти любого языка программирования. Подойдет Visual Basic (VB), Java, C, C#, C++, Perl и другие языки. К основным условиям, которым должны отвечать язык программирования и технология, пригодные для создания Web-сервиса, относятся:

- взаимодействие с коммуникационным протоколом Web;
- обработка входных параметров как части запроса;
- возвращение ответа инициатору запроса.

Функциональность, предоставляемая Web-сервисом, не отличается от функциональности специализированной программы, функции, метода, или подпрограммы. Идея заключается в отчуждении (выносе на внешний уровень) одной или нескольких функций, для их последующего использования одной или несколькими прикладными программами (инициаторами запросов). В силу предоставления конкретной функциональности, Web-сервис подразумевает некоторый контекст применения. Web-сервис может быть разработан для обеспечения определенной функциональности или услуг, в пределах контекста, в соответствии с параметрами, определенными в его интерфейсе. К примеру, крупные Web-сервисы, ориентированные на потребителей, могут обеспечивать следующую функциональность:

- предоставлять информацию о фондовом рынке (для заданного рынка – текущее состояние рынка, историю активности рынка, тренды рынка и т. п.);
- предоставлять информацию о погоде (для заданной точки – текущую температуру, давление, видимость, скорость ветра и т. п.);
- предоставлять информацию о товарах и ценах (для заданного типа товара – розничные точки продажи, наличие товара, цена товара и т. п.).

Хотя название "Web-сервис" подразумевает функционирование сервиса во Всемирной паутине, это совсем не обязательно. Фактически можно создавать Web-сервисы, работающие в пределах предприятия, во внутренней сети. В такой форме Web-сервис является "корпоративным" Web-сервисом. Задачи, решаемые традиционными подпрограммами: преобразование данных, пересчет валют, доступ к данным, обмен данными между системами и т. п., представляют собой широкое деятельности по их реализации на основе внутренних Web-сервисов. Могут быть определены и другие, более мелкие Web-сервисы, обеспечивающие специфическую функциональность для прикладной программы или предназначенные для связи между прикладными программами.

Возможность. *Хотя большинство Web-сервисов предназначены для работы с клиентами, Web-сервисы также могут быть использованы для интеграции предприятия и в качестве брокеров данных.*

Глядя на предыдущие примеры и определение Web-сервиса, проектировщик данных может задаться вопросом: "А зачем Web-сервисы нужны мне?" Ответ двойной. Во-первых, Web-сервис в области интеграции предприятия обладает даже большими возможностями, чем в традиционной для него работе с клиентами и отработке бизнес-логики. Для многих бизнес-предприятий характерна ситуация, когда важная информация "замкнута" в пределах разных узкоспециализированных систем. Структура и характеристики одних и тех же данных могут быть разными от системы к системе. Совместное использование данных и обмен данными при помощи транзакций, описанных на XML, становится наиболее предпочтительным способом для интеграции предприятия. Однако такой подход часто реализуется посредством прямых интерфейсов (связи "точка-точка").

В качестве альтернативы бизнес-приложениям, Web-сервисы могут быть разработаны как интерфейс для обслуживания запросов от системы к системе на доступ к данным, преобразование данных и обмен данными. Для обеспечения интеграции, Web-сервис действует как один или более брокеров данных. Брокеры данных обращаются к источникам данных, выбирают данные, отображают и преобразовывают данные, и перемещают данные от источников к получателям. Для эффективного преобразования данных из одной формы в другую, критическое значение имеет отображение метаданных источника на метаданные получателя. Для обеспечения такого отображения и другой необходимой поддержки интеграции предприятия, можно использовать W3C XML-Схемы, реализованные в виде общих трансформирующих словарей предприятия. В отображении источника данных на получателя и создании основанных на W3C XML-Схеме словарей предприятия существенную роль играет проектировщик данных.

К числу важных вопросов относится содержание запроса к Web-сервису и его ответа. Как уже отмечалось, Web-сервис должен быть интероперабельным. Вследствие этого, предпочтительным методом описания содержания запроса и ответа Web-сервиса является XML. Интерфейс любого Web-сервиса, независимо от контекста и характера использования, соединяет в себе входные (запрос) и выходные (ответ) элементы данных. Определяют характеристики этих элементов данных – метаданные. И здесь опять на первый план выдвигается проектировщик данных, разрабатывающий необходимые схемы.

XML и Web-сервисы

Для эффективной разработки и использования Web-сервиса, необходимо выполнить ряд важных действий. Уже созданный Web-сервис необходимо "опубликовать" для той аудитории, которой он предназначается. Эта аудитория может быть открытой (широкой) (т. е. выставляемый Web-сервис может использоваться любым приложением без всяких ограничений), или закрытой (Web-сервис может использоваться только одним или несколькими доверенными приложениями). Кроме того, до потенциальных пользователей необходимо довести характеристики Web-сервиса. К этим характеристикам может относиться описание предоставляемой функциональности, описание интерфейса (особенно сообщений запроса и ответа), и способов, используемых для обращения к Web-сервису (например, идентификатор ресурса или аналогичное обозначение, допустимые коммуникационные протоколы и необходимая для связи информация). Чтобы Web-сервис можно было "найти", информация Web-сервиса должна быть опубликована в каталоге, доступном для потенциальных пользователей.

После установления необходимого Web-сервиса, обращающееся приложение должно сформировать запрос, отвечающий ожиданиям данного сервиса. Аналогично, это приложение должно также подготовиться к приему и обработке ответа Web-сервиса. Содержание запроса и ответа формируется в соответствии с предопределенной структурой, и должно включать значения данных определенного типа. Описанные действия подразумевают три различных части инфраструктуры Web-сервиса:

1. каталог, в котором опубликована и сделана доступной информация Web-сервиса;
2. описание интерфейса и способа обращения к Web-сервису;
3. взаимодействие (запрос и ответ), отвечающее интерфейсу Web-сервиса.

Каждая из этих частей входит в инфраструктуру посредством которого Web-сервис может быть описан, найден, вызван (привязан) и обработан (рис. 10.1).

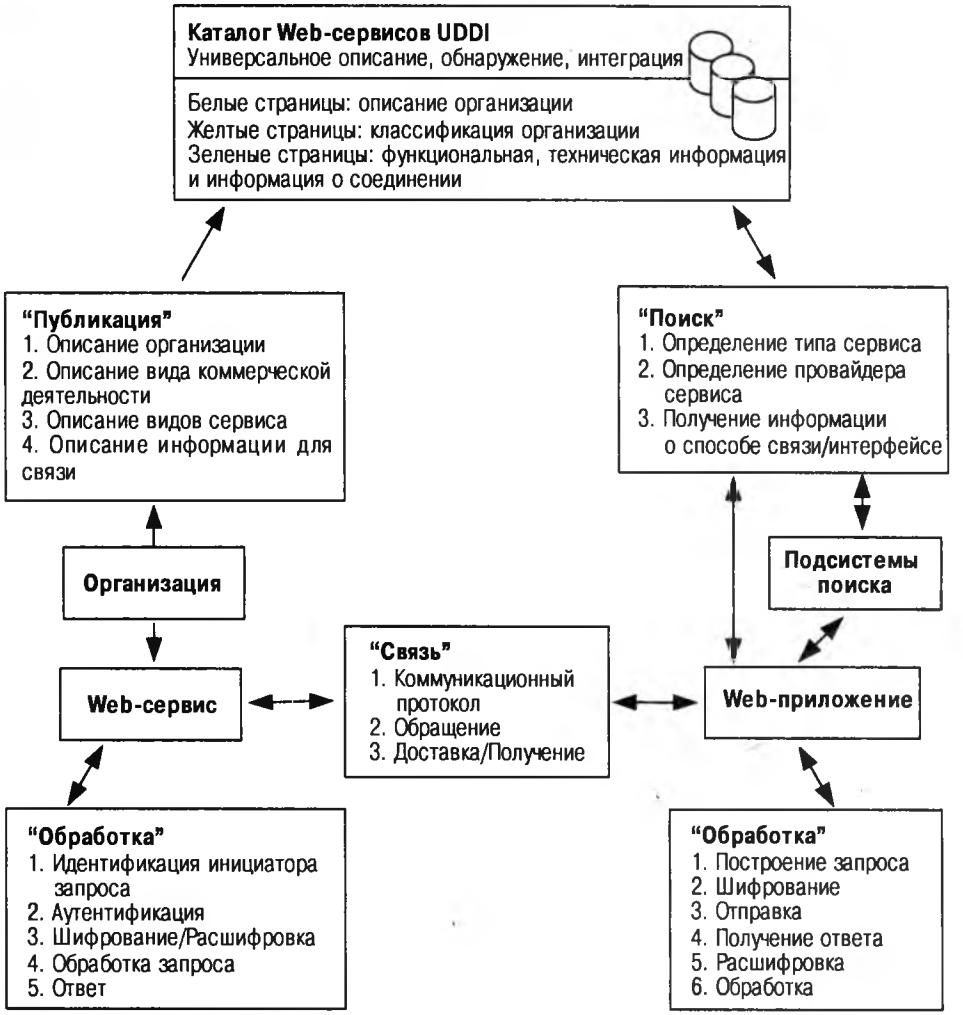


Рис. 10.1. Базовая инфраструктура Web-сервисов

Каталог UDDI

Важным аспектом Web-сервиса является то, каким образом он описывается и публикуется в Web. Чтобы Web-сервис мог быть использован, идентификация, функциональность и интерфейс Web-сервиса должны быть известны приложению, обращающемуся с запросом. Описания зарегистрированных Web-сервисов

хранятся в каталоге UDDI [1](Universal Description Discovery and Integration – универсальное описание, обнаружение и интеграция). То есть когда Web-сервис разрабатывается для широкомасштабного использования в WWW, характеристики провайдера этого сервиса и самого сервиса публикуются в каталоге UDDI. Приложения могут обращаться к этому каталогу для поиска доступных Web-сервисов. Публикация информации Web-сервисов в каталоге UDDI и доступ к этой информации могут быть выполнены с использованием XML (рис. 10.2).

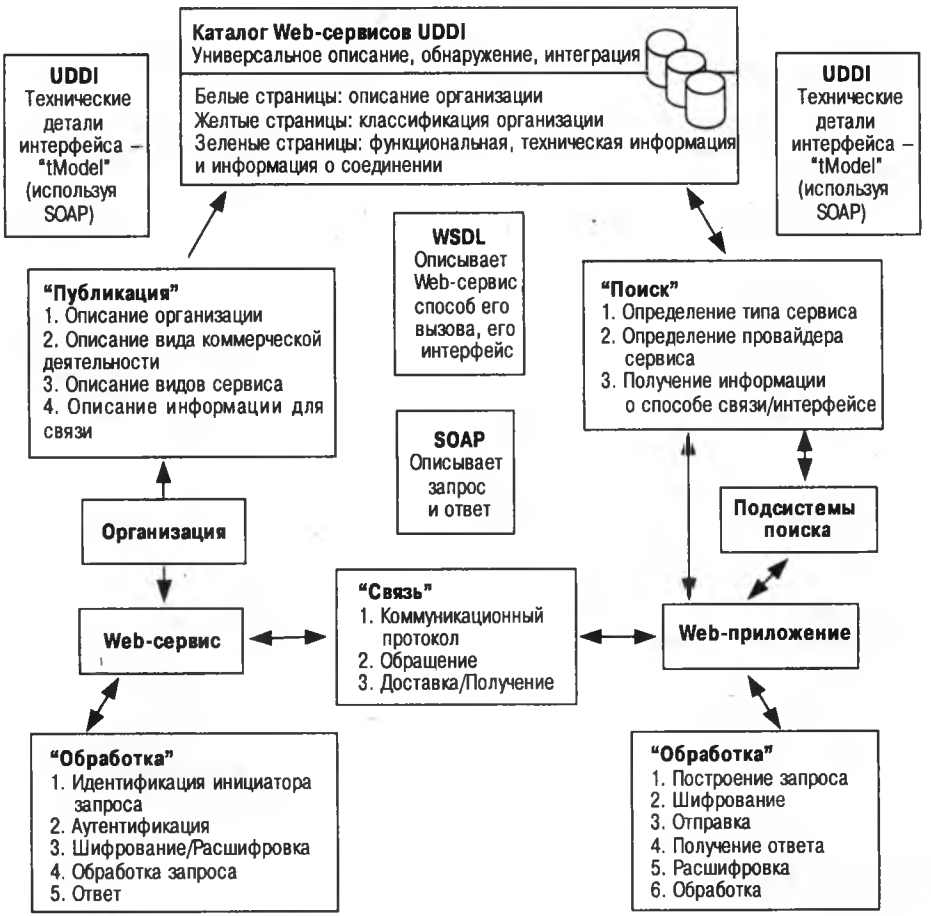


Рис. 10.2. Web-сервисы и XML

[1] Universal Description Discovery and Integration of Web Services. Ariba, International Business Machines, and Microsoft. UDDI, 2000. Доступно по адресу http://www.uddi.org/. – *Примеч. авт.*

Когда Web-сервис предназначен для использования внутри предприятия, могут существовать ограничения, относящиеся к функциональности и доступу к сервису. Очевидно, его публикация в открытом и доступном из всех точек WWW каталоге UDDI – не лучшее решение. Однако потенциальные пользователи должны иметь возможность найти этот сервис. В этом случае более подходящим решением будет использование каталога, созданного внутри предприятия, или такого каталога как LDAP (Lightweight Directory Access Protocol – облегченный протокол доступа к каталогу). Использование внутреннего каталога может быть ограничено определенным кругом пользователей и приложений и соответствующим образом настроено. Однако следует учитывать, что настройка такого каталога может потребовать дополнительных усилий разработчиков.

После того как Web-сервис идентифицирован как доступный для использования и описанная функциональность отвечает требованиям обращающихся приложений, следующий шаг – определение того, как обратиться к сервису и взаимодействовать с ним.

Язык описания Web-сервисов WSDL

WSDL (Web Services Definition Language – язык описания Web-сервисов) обеспечивает способ описания технической информации о Web-сервисе[2], а также о его интерфейсе (т. е. сообщениях запроса и ответа). Важно отметить, что совсем не обязательно использовать именно WSDL для описания Web-сервиса. Если способ обращения и формат интерфейса доводится до заинтересованного приложения иным методом, использование WSDL под вопросом. Однако это также означает, что существует некоторая форма собственной архитектурной привязки обращающегося приложения к Web-сервису. В результате, изменение функциональности, интерфейса, или способа обращения сервиса, может сбить с толку сторону, обращающуюся с запросом. Хотя WSDL технически не обязателен, он важен, и если нет иной формы описания Web-сервиса, требуется тщательно изучить вопрос о применении именно его.

WSDL-документ состоит из нескольких частей, но касательно архитектуры данных, заслуживает обсуждения секция определения типов (секция "types") (рис. 10.3). Секция "types" WSDL-документа описывает характеристики метаданных интерфейса Web-сервиса. Мощь WSDL заключается в том, что для этой секции можно использовать W3C XML-Схемы, которые обеспечивают строгую поддержку типов данных. Связанные секции WSDL-документа связывают типы с именованными элементами сообщения и со структурой сообщения (входное или выходное).

[2] Web Services Definition Language (WSDL) версия 1.2. World Wide Web Consortium (W3C) 2002. Доступно по адресу http://www.w3.org/TR/wsdl12/. – *Примеч. авт.*

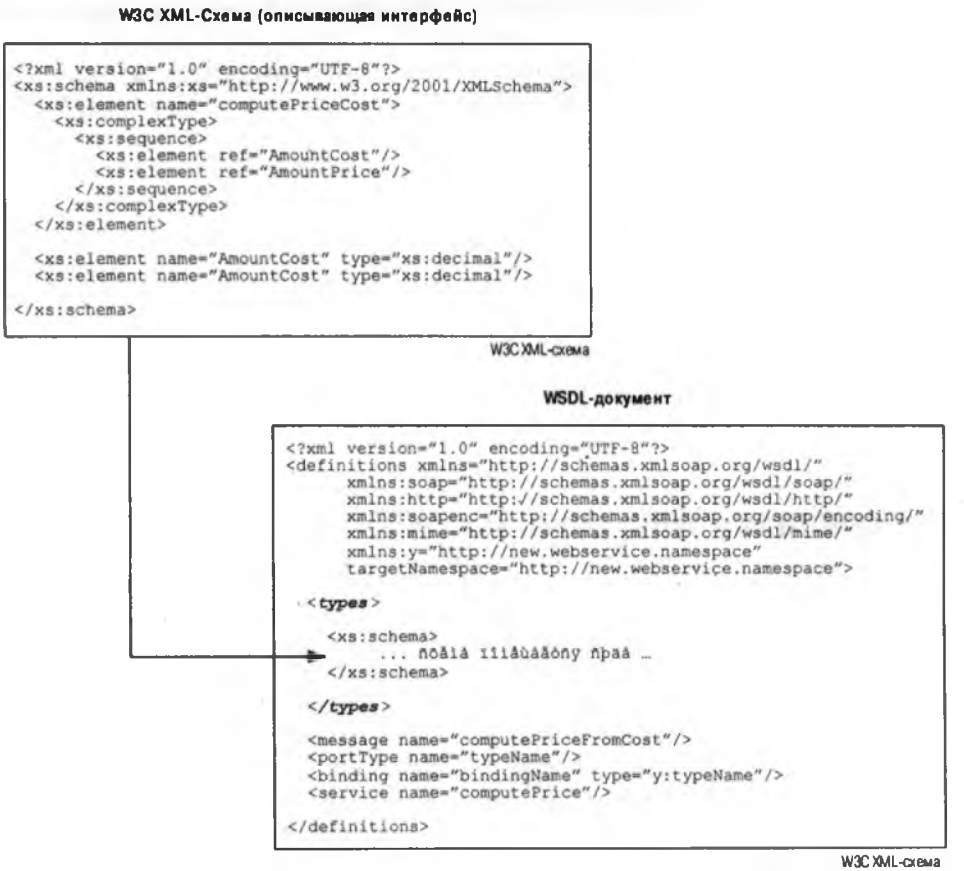


Рис. 10.3. W3C XML-Схемы и секция "types" WSDL-документа

После того как способ обращения известен и интерфейс определен, Web-сервис можно использовать. Если способ обращения включает применение HTTP, запрос к Web-сервису аналогичен запросу Web-страницы. Указывается идентификатор и место размещения ресурса, и содержимое запроса отсылается. Концептуально, содержимое запроса к Web-сервису представляет собой набор параметров и значений данных, который должен соответствовать ожиданиям Web-сервиса. После получения запроса, Web-сервис производит его аутентификацию и авторизацию (имеет ли данный запрос и его инициатор необходимые полномочия), а затем осуществляет проверку корректности содержания запроса. Процесс проверки может включать простейший синтаксический анализ, за которым следует извлечение значений параметров запроса

и обработка данных сообщения запроса, или этот процесс может включать широкую проверку, в соответствии со схемой, чтобы убедится, что содержание сообщения отвечает набору правил из метаданных. Необходимо отметить потенциальную возможность применения шифрования-расшифровки. В зависимости от возможностей Web-сервиса и инициатора запроса, а также конфиденциальности данных интерфейсного сообщения, часть данных сообщения может быть зашифрована.

Простой протокол доступа к объектам SOAP

Обращение к типичному Web-сервису включает сообщение-запрос, посылаемое сервису и сообщение-ответ, которое сервис возвращает инициатору запроса. Из-за необходимости интероперабельности сервиса, эти сообщения часто определяют с помощью некоторой формы XML. SOAP [3](Simple Object Access Protocol – простой протокол доступа к объектам), являющийся диалектом XML, используется в качестве синтаксиса для описания сообщений, посылаемым к и получаемых от Web-сервиса. Протокол SOAP включает три основные части:

1. оболочка SOAP;
2. заголовок SOAP;
3. тело SOAP.

Концептуально, оболочка SOAP – это набор общих описателей и контейнер для сообщения. Подобно базовому пространству имен W3C XML-Схемы, оболочка SOAP включает пространство имен, ссылающееся на используемую версию SOAP. По мере эволюции SOAP, номер версии может использоваться Web-сервисом, для определения того, какого вида интерпретацию и обработку необходимо применить. Оболочка SOAP заключает в себе две другие части: заголовок SOAP и тело SOAP.

Заголовок SOAP содержит информацию о данном конкретном сообщении. Он может включать обозначение инициатора запроса, точку исхода или источник данного сообщения, аутентифицирующую информацию, информацию, относящуюся к шифрованию и безопасности, и прочие технические данные, касающиеся того, как данный запрос необходимо обработать (например, в случае ошибки). Заголовок SOAP допускает наличие нестандартных расширений, если это необходимо для работы Web-сервиса.

Возможность. *Проектирование интерфейса Web-сервисов (особенно тела SOAP) подразумевает необходимость в описывающих метаданных. Поэтому, несмотря на то что основная нагрузка при создании Web-сервиса ложится на специалиста по разработке приложений, в проектировании структуры сообщения и схемы, описывающей применяемые характеристики метаданных, должен принимать участие проектировщик данных.*

[3] Simple Object Access Protocol, SOAP версия 1.2. W3C, 2002. Доступно по адресам: http://www.w3.org/TR/soap12-part1/, http://www.w3.org/TR/soap12-part2/ и http://www.w3.org/TR/soap12-af/. – *Примеч. авт.*

Тело SOAP определяет содержимое сообщения. Эта секция сообщения SOAP представляет собой набор контейнеров (часто являющихся контейнерами-элементами) и значений данных в них содержащихся. Значения данных сообщения-запроса SOAP подобны параметрам. Структура тела SOAP зависит от того, к какому серверу направляется запрос. Эту структуру определяет интерфейс, поддерживаемым данным Web-сервисом (возможно этот интерфейс описан с помощью WSDL-документа). И это и есть определение для тела SOAP, и характеристики метаданных этого определения безусловно важны. Если элементы данных в том виде, как они заданы в теле сообщения-запроса SOAP не соответствуют ожидаемой структуре (т. е. перечню требуемых контейнеров, заданному порядку их следования и необходимым типам данных), Web-сервис не сможет обработать данный запрос. Проектирование тела SOAP похоже на разработку прототипа XML-транзакции и ограничивающей схемы. Хотя в данном случае основная ответственность лежит на разработчике приложений, при создании тела SOAP важно подключить к сотрудничеству проектировщика данных. Кроме того, проектировщик данных также должен разрабатывать соответствующую W3C XML-Схему. Если для описания интерфейса используется WSDL, типы из этой W3C XML-Схемы затем станут частью группы элементов "types" в WSDL (рис. 10.4).

Зачем компании разрабатывать и публиковать Web-сервисы?

Некоторые могут задаться вопросом, зачем компании создавать и развертывать Web-сервис, который будет использоваться другими. Опираясь на посылку, что Web-сервисы реализуют бизнес-модель, основанную на совместной работе (collaborative business model), трудно сформулировать оправдание разработки Web-сервисов. В настоящее время существуют три основные бизнес-модели, реализуемые Web-сервисами:

1. альтруистическая;
2. приносящая непосредственный доход:
   - за каждый запрос;
   - за лицензию на определенный период;
3. приносящая косвенный доход:
   - маркетинговый сбор и анализ информации о запрашиваемых данных (поток щелчков мышью, интересующие темы, и инициаторы запроса).

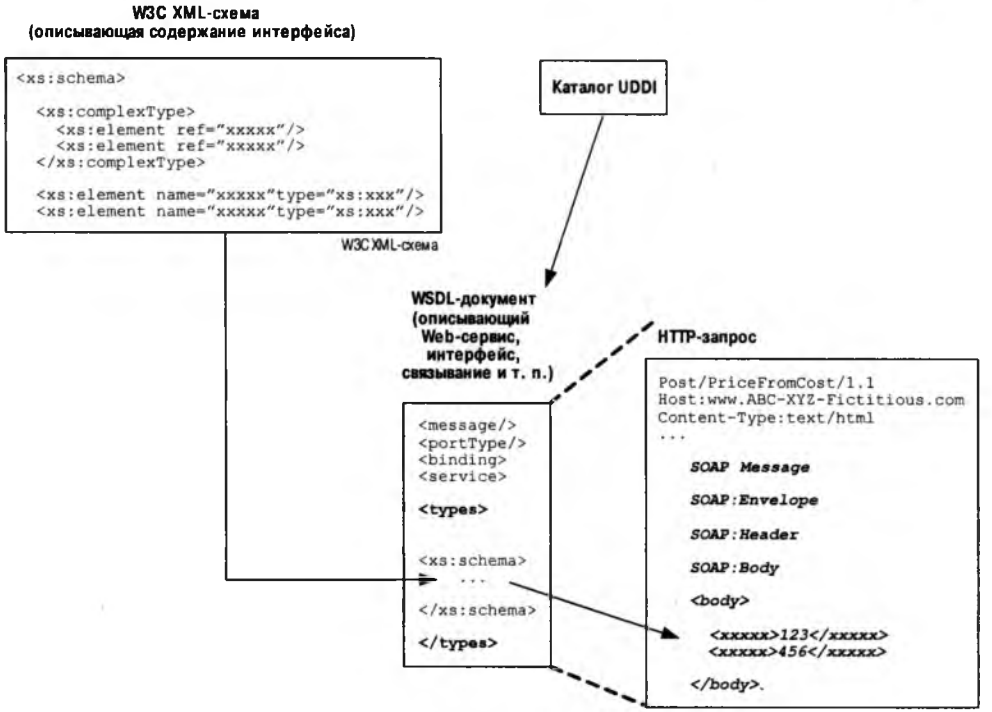


Рис. 10.4. Сообщение для Web-сервисов, использующее HTTP и SOAP

С точки зрения выгоды, альтруистическая модель наиболее уязвима. Как следует из названия, такие Web-сервисы создаются и развертываются для свободного или бесплатного использования. Эти Web-сервисы развертываются в открытой среде, где пользователь не должен вносить плату за их использование. Могут существовать разные варианты данной модели, в которых права на интеллектуальную собственность принадлежат разработчику и копирование интерфейса или функциональности Web-сервиса запрещено. Большинство предприятий, основанных на получении дохода, имеют трудности с обоснованием разработки альтруистической модели. Однако можно подойти к этому вопросу шире, представив альтруистический Web-сервис, как жест доброй воли, или как некоторый стимулирующий сервис, ведущий к дополнительному доходу. Эта бизнес-модель продвигает "открытую" Всемирную паутину.

Модель, приносящая непосредственный доход интуитивна и понятна большинству бизнесов. Доход – вот внутреннее содержание такого Web-сервиса. Две основные вариации этой модели: оплата за каждый запрос и продажа лицензии на некоторый период (лицензии на использование Web-сервиса на определенный период времени). В этом случае должен существовать устойчивый рынок для Web-сервиса. Ударение на словах доход и оплата может быть понято некоторыми как препятствие для открытой и совместной обработки. Такой Web-сервис также должен включать логику по аутентификации обращающегося с запросом (т. е. определение того, имеет ли он лицензию или необходимые полномочия) и некоторую форму регистрации, для учета каждого запроса.

Третья основная бизнес-модель Web-сервиса – косвенное извлечение дохода. Эта модель похожа на сбор идентификаторов пользователей, адресов электронной почты, и данных о потоке щелчков от посетителей Web-страниц, используемых для определения предпочтений среди товаров и услуг, и, потенциально, для последующей перепродажи. Отслеживание потока щелчков напоминает подглядывание за тем, какие страницы и объекты Web посещают пользователи. Из-за того что Web-сервис обеспечивает функциональность для определенного контекста или целей, связывание инициатора запроса с видом запрошенного сервиса подразумевает возможность выполнения определенных информационных исследований. Можно предположить, что обратившийся к сервису определенного типа, может заинтересоваться аналогичными товарами и услугами. Риски, связанные с этой бизнес-моделью, относятся к охране частной жизни и ответственности за разглашение данных об инициаторе запроса.

Перечисленные бизнес-модели в основном фокусируются на Web-сервисах, развертываемых для использования в WWW и, возможно, сторонними потребителями. Обоснование необходимости во внутренних Web-сервисах (внутри предприятия) обычно базируется на уменьшении стоимости разработки и повторном использовании. Как и стандартная подпрограмма, внутренний Web-сервис может быть создан для обеспечения фунукциональности, необходимой предприятию, но предоставляемой независимым от платформы способом. Такой Web-сервис можно развернуть во внутренней сети совершенно аналогично тому, как развертывается внешний Web-сервис, при этом бизнес-логика и методы программы будут скрыты от инициаторов запроса. Как и для любого повторноиспользуемого технологического актива, здесь существует эффект снижения стоимости разработки (например, не нужно заново создавать логику приложения, которая уже существует) и уменьшения времени разработки.

Рекомендация. *Если сообщение с запросом или ответом Web-сервиса содержит конфиденциальную информацию, следует применять соответствующую технику шифрования и обеспечения безопасности.*

С разработкой и развертыванием Web-сервисов могут быть связаны некоторые проблемы и риски. Если Web-сервис создается и развертывается в Web для использования другими, существуют затраты и проблемы, связанные с технической поддержкой и доступностью сервиса. Если Web-сервис страдает от плохой доступности или имеет низкую производительность, или изобилует ошибками, вряд ли он сможет выжить как способ генерации дохода. Существуют также риски, связанные с неавторизованным использованием, возможностью вторжения, взлома конфиденциальной информации и т. п., причем независимо от того, является этот сервис внешним или внутренним. И хотя все эти проблемы важны, они не являются непреодолимыми.

Будущее Web-сервисов

Как уже упоминалось, Web-сервис подобен подпрограмме или функции сервера. Клиенты (к ним могут относиться пользователи, бизнесы, или прикладные программы) обращаются к Web-сервису с запросом и затем обрабатывают полученный ответ. Эта модель не является революционной для разработки приложений. Будущее Web-сервисов еще только определяется, но направление развития – координируемые сочетания Web-сервисов, как интеллектуальных агентов. В запрос к Web-сервису может быть помещен набор некоторых критериев, и сервис будет обрабатывать его, и вернет ответ с различными опциями. Приложение, обращающееся с запросом, является при этом сборкой функций-компонентов, которые оценивают ответы сервисов, определяют необходимые действия и сотрудничают между собой. Чтобы избежать сложностей, возникающих при использовании разных платформ и технологий, сотрудничество (совместная обработка) предполагает строгое использование XML.

Как упоминалось ранее, Web-сервис может быть разработан, чтобы действовать как основной интерфейс брокера данных для интеграции предприятия. Добавление интеллекта в виде набора сотрудничающих функций может расширить концепцию брокера данных до обозначения основного источника данных предприятия, и обеспечения маршрутизации и выборки из альтернативных источников данных в случае, когда основной источник недоступен. Данные, выбранные брокером, также могут быть приведены в соответствие с форматом, качеством и объемами, диктуемыми метаданными. Информационный брокер данных может обращаться к другим Web-сервисам для осуществления необходимых преобразований и чисток. В итоге, основной Web-сервис выдает ответ на запрос, связанный с интеграцией данных, но в зависимости от обработок и ответов, выполненных подчиненными сервисами, для этого могут быть предприняты разные варианты действий и обращений. Запрашивающей стороне не нужно знать о предпринятых действиях, за исключением случая, когда итоговый ответ не отвечает ее ожиданиям.

Точно также могут быть разработаны и другие подобные приложения. Расширения Web-сервиса, которые определяют брокера данных предприятия, могут "узнавать", когда к информации предприятия добавляются новые данные. Если профиль новых данных (вытекающий из их стандарта именования, контекста и характеристик метаданных) соответствует другим требованиям поддержки решений, могут быть активизированы дополнительные Web-сервисы для очистки и преобразования, с целью подготовки хранилища информации к новому наполнению. Если логика очистки и преобразования находится в Web-сервисах вне предприятия, могут быть задействованы и эти внешние сервисы. Очевидно, эти примеры приложений Web-сервиса чисто теоретические и требуют строгого анализа и проектирования. Кроме того, должны быть разрешены традиционные вопросы безопасности, производительности и доступности. Однако потенциальные возможности применения Web-сервисов почти безграничны. Возможность разрабатывать прикладные программы, сотрудничающие между собой в Web и, в некоторой форме, применяющие абстрактный интеллект, основываясь на динамических событиях и ответах, представляет широкие перспективы для Web-сервисов.

Приложение А

Факты, рекомендации, техники и возможности

Ключевой особенностью данной книги является выделение важной информации в отдельные фрагменты по рубрикам:

- факты;
- рекомендации;
- техники;
- возможности.

Факты основываются на достоверных источниках или многочисленных наблюдениях. Рекомендации носят субъективный характер и являются результатом моего личного опыта и исследований. Техники определяют тактические приемы, а возможности помогают определить, когда необходимо применить ту или иную технику или избежать потенциальных ловушек.

Эти фрагменты присутствуют в тексте соответствующих глав и дополнительно приводятся здесь, сгруппированные по темам.

Обоснованность

Факт. *XML-самоописывающий язык (поддерживает описательные теги элементов и атрибутов).*

Факт. *XML независим от платформы (по умолчанию используется кодировка Unicode и UTF-8, поддерживающая основной набор символов ASCII).*

Факт. *XML является повторноиспользуемым (W3C XML-Схемы могут быть спроектированы как модульные компоненты).*

Факт. *XML – гибкий язык (XML-структуры можно разрабатывать в расчете на оперативное расширение или сокращение).*

Факт. *XML расширяем (XML-структуры и схемы могут быть расширены сами или добавлены для расширения другого).*

Типы схем

Рекомендация. *DTD-схемы могут эффективно использоваться как метод описания и введения ограничений для простого документоориентированного содержания. Однако во многих случаях вместо них с тем же успехом можно использовать и W3C XML-Схемы.*

Рекомендация. *W3C XML-Схемы, в общем случае, прекрасно подходят для описания содержимого, ориентированного на транзакции (как вне, так и внутри предприятия).*

Рекомендация. *W3C XML-Схемы прекрасно подходят для описания содержимого, ориентированного на сообщения.*

Факт. *В зависимости от характеристик данных, их назначения и количества обрабатывающих их приложений, может потребоваться использовать более одного типа схемы.*

Рекомендация. *Если необходимо использовать только один тип схемы, рассмотрите в качестве первого претендента W3C XML-Схемы.*

Стандарт именования

Возможность. *Когда существующие на предприятии стандарты и процессы именования данных являются описательными, интуитивными, общепризнанными и не нарушают синтаксических правил XML, они могут быть адаптированы для присвоения имен атрибутам и элементам XML.*

Рекомендация. *Имя элемента или атрибута XML должно быть интуитивно понятным ("правило интуитивности").*

Рекомендация. *Избегайте использования артиклей, предлогов, местоимений и имен собственных в составе имен XML.*

Рекомендация. *Избегайте использования акронимов и аббревиатур, за исключением легко узнаваемых и общераспространенных.*

Рекомендация. *Используйте слова класса только тогда, когда они коротки, представляют понятное, и исчерпывающе определяют ограничение.*

Рекомендация. *Если не существует обязательных ограничений (например, в виде уже существующих технологий и инструментальных средств, поддерживающих лишь символы одного регистра), старайтесь в имени элемента или атрибута XML использовать символы обоих регистров.*

Рекомендация. *Разделители между составными частями имени используются только для обеспечения читабельности имен элементов и атрибутов.*

Факт. *Для визуального деления имени элемента или атрибута в XML можно использовать символы подчеркивания ("_"),тире ("–"), и точки (".").Однако каждое вхождение в имя символа разделителя увеличивает его длину еще на одну позицию.*

Рекомендация. *Использование в именовании подхода "camel case#?", при котором каждый начальный символ каждой составной части имени находится в верхнем регистре, а остальные символы – в нижнем, визуально разделяет составные части. Отпадает необходимость введения в имя дополнительных символов-разделителей.*

Рекомендация. *Имена атрибутов и элементов XML должны иметь оправданную длину (быть не слишком многословными, но и не слишком сокращенными).*

Рекомендация. *Длина имен атрибутов и элементов XML должна быть оптимизирована и согласована с большинством общих источников данных и получателей данных.*

Возможность. *Абстрактные элементы и атрибуты XML, из имени которых исключены внутренние "роли" или "классификации" и для которых уточнение их текущего назначения производится с помощью их родительских элементов или описательных атрибутов, хорошо подходят для повторного использования.*

Факт. *Слишком абстрактные имена элементов и атрибутов XML могут привнести излишнюю сложность и даже стать причиной неверного толкования.*

Возможность. *Проектировщик данных обычно хорошо разбирается в стандартах именования и процессах именования, и прекрасно знает информационные активы предприятия (их определение и использование). Несмотря на уровень абстракции или определенности, и прочие процессы стандарта именования в XML, проектировщики данных имеют реальную возможность принять в них участие и применить все свои навыки.*

Типы данных

Факт. *Несмотря на возможное совпадение названий типов данных в различных продуктах баз данных, типы данных не всегда реализовываются и поддерживаются одинаковым образом.*

Факт. *Наиболее часто данные, переносимые посредством транзакций или интерфейсных файлов, в той или иной форме, берутся из базы данных и, так или иначе, сохраняются в базе данных.*

Факт. *W3C XML-Схемы обеспечивают обширную поддержку типов данных для даты и времени (например, даты, времени, продолжительности, часового пояса и отдельных частей даты). Однако платформы баз данных могут не обеспечивать поддержку типов данных для даты и времени в соответствии с ISO 8601. Поэтому применение некоторых типов данных даты и времени в W3C XML-Схемах может потребовать использования символьных или числовых типов баз данных для хранения таких данных в базе данных, а также применения соответствующей логики преобразования.*

Факт. *По умолчанию, десятичные (decimal) типы данных и некоторые из целочисленных типов W3C XML-Схем не имеют ограничений в виде минимального и максимального значений (они "неограниченные"). Однако в большинстве баз данных десятичные и целочисленные типы имеют определенные верхние и нижние пределы. При обмене числовыми данными следует быть осторожными и следить за тем, чтобы не выйти за пределы ограничений базы данных.*

Техника. *Синтаксическая форма назначения типа данных W3C XML-Схемы такова: "type="тип_данных"", здесь "тип_данных" – один из поддерживаемых в W3C XML-Схеме типов, например, "integer", "date", "dateTime", "decimal", или "boolean".*

Рекомендация. *Идентифицирующие данные, являющиеся критически важными для обработки транзакции или сообщения XML, необходимо включать как элемент или атрибут и адекватно описывать с помощью имени или местоположения внутри транзакции (или сообщения).*

Рекомендация. *В XML-транзакции должен использоваться тот же уровень детализации и состав уникальных идентификаторов, что определен источником данных или пунктом назначения. Если идентификатор является комбинацией отдельных элементов данных (составной ключ), для них должны быть определены аналогичные (источнику или пункту назначения) контейнеры.*

Рекомендация. *Использование типа данных – уникальный идентификатор ("ID") W3C XML-Схемы, должно ограничиваться только теми элементами и атрибутами, которым требуется уникальность содержащихся в них значений в пределах одного XML-документа. Из-за ограничений на допустимые значения, тип "ID" невозможно использовать для описания первичных ключей базы данных или аналогичных уникальных идентификаторов. Проверка уникальности идентификаторов базы данных следует оставить самим системам баз данных.*

Фасеты

Факт. *Фасеты служат для задания пределов, ограничений и описания типов данных. В некоторых случаях, в зависимости от типов данных, фасеты можно комбинировать друг с другом.*

Техника. *Фасеты символьной длины можно использовать для разрешения отличий в характеристиках метаданных между источником и получателем данных, путем ограничения числа символьных позиций до наименьшего общего значения.*

Техника. *Фасет minLength может использоваться для контроля обязательности присутствия значения данных в элементе или атрибуте XML. Если его значение равно "1", верифицирующий синтаксический анализатор проконтролирует, что значение данных соответствующего контейнера, по крайней мере, равно одному пробелу (непустое). Если не будет совсем никакого значения (что эквивалентно "null") будет возбуждена ошибка верификации.*

Техника. *В некоторых случаях фасеты ограничения значений (minInclusive (>=), maxInclusive (<=), minExclusive (>), maxExclusive (<)) можно использовать для разрешения расхождений между источниками и получателями данных путем ограничения допустимых значений общим для них диапазоном .*

Техника. *Цифровые фасеты можно использовать для разрешения различий в характеристиках метаданных между источником и получателем данных, путем ограничения количества цифр и символьных позиций до наименьшего общего.*

Техника. *Фасеты перечисления (список допустимых значений) могут использоваться для поддержания внутренних стандартов и, в некоторых случаях, для разрешения различий в характеристиках метаданных между источником и получателем данных, путем ограничения допустимых значений набором, согласованным с обеими системами.*

Техника. *Фасеты перечисления (список допустимых значений) можно использовать для описания стандартов данных, таких как наборы кодов (например, коды стран, валют, штатов США). Однако следует соблюдать осторожность в случаях, когда списки перечислений определяются внутри нескольких схем, а не во внешнем подмножестве схем, на которые можно ссылаться*[1].

Техника. *Списки перечислений, обычно полезные, не всегда являются лучшим архитектурным решением для проверки допустимых значений, особенно если размеры списков велики (более 200 значений), или эти списки часто обновляются (чаще 1-2 раз в месяц).*

Техника. *Фасеты шаблонов можно использовать для ограничения допустимых символов и возможных позиций этих символов в данных (наподобие формата отображения). Также как и списки перечислений, шаблоны необходимо выделять во внешние схемы и затем ссылаться на них, чтобы избежать лишних затрат на поддержание схемы.*

Рекомендация. *В случаях, когда содержимое XML-документов не является документно-ориентированным, следует избегать использования фасетов пробельных символов. В зависимости от синтаксического анализатора и процесса верификации, использование фасетов пробельных символов может привести к тому, что извлеченные получателем данные будут отличаться от первоначального содержимого XML-документа.*

Структурные модели

Факт. *Все XML-структуры являются иерархическими (и читаются сверху вниз, слева направо).*

Факт. *Вертикальные структурные модели компонуются преимущественно из контейнеров-элементов. В сочетании с повторением элементов (кардинальность или мультипликативность), вертикальные модели структур XML являются наиболее гибкими. Они могут быть динамически расширены или сжаты в соответствии с характеристиками содержащихся данных.*

---

1. Имеется ввиду, что при дублировании списков перечислений в двух или более схемах можно получить недостоверные данные, когда при сопровождении XML-Схем списки перечислений будут подвергаться модификации, и внесенные в них изменения не будут отражены одинаково во всех копиях. Этой ситуации можно избежать с помощью определения списков перечислений во внешней подсхеме и дальнейшего использования этой подсхемы через ссылки на нее. – *Примеч. ред.*

Рекомендация. *В зависимости от количества контейнеров-элементов, объема содержащихся данных и стандарта именования, используемого для присвоения имен элементам, вертикальные структурные модели могут привести к чрезмерной многословности и существенному размеру документа. Если очень важна производительность приложения, или ограничена пропускная способность канала передачи XML-документа, проектировщику данных придется рассмотреть возможность сжатия, обратиться к другим структурным моделям, или применить другой подход проектирования.*

Факт. *В горизонтальных структурных моделях используется значительное количество атрибутов. Поэтому они имеют скорее "горизонтальную" протяженность, нежели вертикальную.*

Рекомендация. *Когда суммарный объем XML-документа или транзакции слишком велик, или когда пропускная способность канала его передачи ограничена, возможно имеет смысл использовать горизонтальную структурную модель.*

Факт. *Использование компонентных структурных моделей обеспечивает хорошие предпосылки для повторного использования. Применяя внешние подсхемы, которые можно, с помощью ссылок, повторно использовать в разных схемах, получаем высоко стандартизированные и модульные по конструкции группы контейнеров.*

Рекомендация. *Наиболее эффективно использовать XML-атрибуты в качестве контейнеров для порядкового номера элемента в последовательности повторяющихся элементов, для описательной классификации элемента, для значений стандартных кодов, описания функции или вида деятельности и для хранения отдельных частей элемента, содержащего данные.*

Факт. *Гибридные структурные модели сочетают характеристики трех других структурных моделей (вертикальной, горизонтальной и композитной). При правильном использовании гибридные модели приводят к созданию гибких, хорошо определенных и повторно используемых XML-структур.*

Архитектурные формы контейнера

Рекомендация. *Жесткие архитектурные формы контейнера рекомендуется применять для XML-документов, используемых для обмена информацией с внешними для предприятия сотрудничающими группами, а также для определения интерфейса Web-сервиса. Во избежание ошибочной интерпретации, внешние сущности обычно требуют использования стандарта именования создающего конкретизированные описательные имена.*

Факт. *Жесткие архитектурные формы контейнера могут упростить навигацию и обработку структуры XML-документа.*

Факт. *Жесткие архитектурные формы контейнера, как правило, не отличаются гибкостью. Добавление новых контейнеров с уникальными именами обычно требует изменения структуры XML-документа, соответствующих схем и, возможно, прикладных программ, обрабатывающих документ.*

Рекомендация. *Абстрактные формы контейнера рекомендуется применять для структур, в которых присутствует шаблон данных, несколько раз повторяющийся в одном или в похожем контексте (например, строки для улицы в почтовом адресе, части полного имени человека и телефонные номера разных видов).*

Факт. *Абстрактные формы контейнера основываются на концепциях повторения и кардинальности.*

Техника. *Абстрактные формы контейнера используют, до некоторой степени, абстрактные имена элементов. Один из основополагающих принципов абстрактной формы контейнера – имена элементов должны, с одной стороны, достаточно хорошо описывать содержимое элементов, а с другой стороны, должны быть достаточно абстрактными, для того, чтобы было возможно повторение. Если же все еще остается неясность относительно содержимого повторяющихся элементов, родительский элемент повторяющейся группы должен обеспечивать дополнительный контекст.*

Факт. *Абстрактные формы контейнера часто включают атрибут для описания порядкового номера повторяющегося элемента в группе таких элементов.*

Рекомендация. *За исключением горизонтальных структурных моделей, рекомендуется использовать атрибуты только для: указания порядкового номера, вида или классификации, стандартных кодов, закрепленной функции или вида деятельности, отдельных частей значения данных.*

Техника. *При правильном применении, задание значений атрибутам minOccers и maxOccurs для повторяющихся элементов является мощной возможностью W3C XML-Схем по указанию степени кардинальности. Однако следует соблюдать осторожность, если кардинальность или характеристики повторяющихся элементов подвержены изменениям. Такие изменения потребуют модификации соответствующих W3C XML-Схем.*

Факт. *Абстрактные архитектурные формы контейнеров могут внести сложность в навигацию и обработку структуры XML-документа. Для опроса характеристик атрибутов может потребоваться дополнительная логика.*

Рекомендация. *С некоторыми исключениями, гибридная форма контейнера – рекомендуемая архитектурная адаптация для структуры XML_транзакции.*

Повторное использование

Факт. *Разработка повторноиспользуемой XML-схемы – это набор практических приемов, технических подходов и мероприятий, необходимых для разработки W3C XML-Схемы или схемы-компонента, с учетом необходимости ее повторного использования.*

Факт. *Собственно повторное использование XML-схемы – процесс, включающий действия по определению возможности повторного использования, проверке, и реализации повторного использования W3C XML-Схем, подсхем или схем-компонентов.*

Рекомендация. *Все контейнеры-элементы XML следует определять глобально, чтобы их можно было использовать повторно. Это не касается только тех элементов, которые намеренно защищаются от повторного использования.*

Рекомендация. *Наборы взаимосвязанных контейнеров-элементов, специально предназначенные для повторного использования, следует определять как группу ("group"). За исключением случаев, когда очевидны преимущества использования для этих целей сложных типов ("complexType").*

Техника. *Простой тип (simpleType) W3C XML-Схемы является мощным методом определения стандартных типов данных предприятия и ограничений на допустимые значения контейнеров (элементов и атрибутов).*

Факт. *Повторное использование W3C XML-Схемы или подсхемы – концептуальная форма разработки методом сборки. Основная W3C XML-Схема собирается с помощью ссылок на содержимое других, определенных вовне W3C XML-Схем.*

Факт. *Наиболее общие виды повторноиспользуемых подсхем включают стандартизированные структуры, стандартные коды и допустимые значения, а также нестандартные (собственные) типы данных.*

Рекомендация. *Внешнее имя файла повторно используемой W3C XML-Схемы (подсхемы) должно быть интуитивно понятным, в оправданной степени конкретным. В нем должны использоваться символы обоих регистров (стиль "camel case") и не должно быть пробелов. Длина имени не должна превышать 32 символа.*

Рекомендация. *Внешнее имя файла повторноиспользуемой W3C XML-Схемы (подсхемы) должно включать номер версии. Номер версии может быть префиксом или суффиксом имени, в зависимости от стандартов предприятия.*

Рекомендация. *Повторноиспользуемые W3C XML-подсхемы должны содержать в имени ту или иную форму классификации или вида (например, "STD" [стандартная структура], "CODES" [список значений кода или перечисления], или "TYPES" [нестандартные типы данных]).*

Рекомендация. *Повторноиспользуемая W3C XML-Схема, предназначенная для обмена данными (интеграции предприятия), должна включать в себя несколько разных структур и форматов, учитывающих потенциальные варианты использования и обработки данных.*

Рекомендация. *Когда основная (ссылающаяся W3C XML-Схема) имеет единый контекст (т.е. пространство имен), для ссылки на определенную вовне подсхему лучше использовать включение ("include").*

Техника проектирования

Рекомендация. *Рекомендуется вести процесс проектирования и разработки W3C XML-Схемы совместными усилиями. И проектировщик данных, и разработчик должны вместе принимать участие в создании прототипа XML-документа и соответствующих W3C XML-Схем.*

Рекомендация. *Как правило, наибольшая глубина вложенности элементов для содержания, ориентированного на транзакции, не должна превышать 10 уровней.*

Техника. *В случае, когда количество повторений некоторого элемента в группе внутри родительского элемента имеет существенное значение для логики навигации, добавьте на уровне родительского элемента атрибут, содержащий число повторений. Но позаботьтесь при этом ввести дополнительную логику для определения числа повторений и инициализации этим числом данного атрибута, а также логику последующего опроса и использования значения атрибута.*

Рекомендация. *Как правило, старайтесь избегать строго горизонтальных структурных моделей, за исключением случаев, когда существует ограничение на размер создаваемого XML-документа, или когда этот XML-документ ни что иное, как простая выборка из реляционной базы данных.*

Рекомендация. *общий размер XML-документа влияет на производительность сети и приложения, осуществляющего его обработку. Необходимо соотнести количество символов, приходящееся на имена элементов и атрибутов с количеством символов, приходящимся на "полезные" данные (содержимое). Если "коэффициент тег/данные" превышает приемлемый уровень, следует пересмотреть необходимость использования XML.*

Техника. *Для некоторых сценариев обработки, может иметь смысл определить в XML-транзакции производные данные. Однако здесь требуется тщательная оценка общего размера документа, сложность производных, и количество детализированных элементов данных.*

Web-сервисы

Возможность. *Хотя большинство Web-сервисов предназначены для работы с клиентами, Web-сервисы также могут быть использованы для интеграции предприятия и в качестве брокеров данных.*

Возможность. *Проектирование интерфейса Web-сервисов (особенно тела SOAP) подразумевает необходимость в описывающих метаданных. Поэтому, несмотря на то, что основная нагрузка при создании Web-сервиса ложится на специалиста по разработке приложений, в проектировании структуры сообщения и схемы, описывающей применяемые характеристики метаданных, должен принимать участие проектировщик данных.*

Рекомендация. *Если сообщение с запросом или ответом Web-сервиса содержит конфиденциальную информацию, следует применять соответствующую технику шифрования и обеспечения безопасности.*

Приложение Б

Примеры синтаксиса W3C XML-Схемы

Синтаксис W3C XML-Схемы для определения элемента (локальное объявление)

(Пример фрагмента W3C XML-Схемы)

```
<xs:complexType>
  <xs:sequence>
    <xs:element name="ElementName" type="xs:integer"/>
  </xs:sequence>
</xs:complexType>
```

Данный элемент определяется в той же точке, где он первоначально объявляется. Такие локально объявленные элементы обычно недоступны для повторного использования за пределами области видимости их первоначального объявления.

Синтаксис W3C XML-Схемы для определения сложного типа (complexType) (локальное объявление)

(Пример фрагмента W3C XML-Схемы)

```
<xs:element name="ParentofcomplexType">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ElementName1" type="xs:integer"/>
      <xs:element name="ElementName2" type="xs:integer"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Данный сложный тип определяется в той же точке, где он первоначально объявляется. В этом фрагменте применен компоновщик “sequence”. Локально объявленный сложный тип обычно недоступен для повторного использования за пределами области видимости его первоначального объявления.

Синтаксис W3C XML-Схемы для определения элемента (глобальное объявление)

(Пример фрагмента W3C XML-Схемы)

```
<xs:complexType>
  <xs:sequence>
    <xs:element ref="GlobalElementName"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="GlobalElementName" type="xs:integer"/>
```

Объявление данного поименованного элемента действует в пределах всей схемы (в соответствии с термином "глобальное"). Поэтому этот элемент можно неоднократно повторно использовать в любой точке W3C XML-Схемы при помощи ссылки (в нашем случае ссылка расположена внутри сложного типа).

Синтаксис W3C XML-Схемы для определения группы (group) (глобальное объявление)

(Пример фрагмента W3C XML-Схемы)

```
<xs:group ref="GlobalGroupName"/>

<xs:group name="GlobalGroupName">
  <xs:sequence>
    <xs:element name="ElementOneName" type="xs:string"/>
    <xs:element name="ElementTwoName" type="xs:integer"/>
  </xs:sequence>
</xs:group>
```

Объявление данной поименованной группы содержит два дочерних элемента, упорядоченных компоновщиком "sequence". Группа определена для всей схемы (т. е. "глобально"). Поэтому эту глобально объявленную группу можно использовать повторно в любой точке схемы при помощи ссылки.

Синтаксис W3C XML-Схемы для определения простого типа (simpleType) (глобальное объявление типа данных)

(Пример фрагмента W3C XML-Схемы)

```
<xs:element name="ElementName" type="GlobalsimpleTypeName"/>

<xs:simpleType name="GlobalsimpleTypeName">
  <xs:restriction base="xs:string">
    <xs:minLength value="3"/>
    <xs:maxLength value="10"/>
```

```
    </xs:restriction>
  </xs:simpleType>
```

Объявление данного именованного простого типа вводит нестандартный тип данных. Этот тип базируется на типе "string" W3C XML-Схемы, к которому применены два ограничивающих фасета: minLength и maxLength (при желании можно добавить и другие фасеты). Данный простой тип определен для всей схемы (т. е. "глобально"). Поэтому этот глобально объявленный простой тип можно использовать повторно в любой точке схемы при помощи ссылки. В таком случае ссылающийся на него элемент становится элементом этого типа.

Синтаксис W3C XML-Схемы для определения простого типа (simpleType) (глобальное объявление перечисления)

(Пример фрагмента W3C XML-Схемы)

```
<xs:element name="ElementName" type="GlobalsimpleTypeName"/>

<xs:simpleType name="GlobalsimpleTypeName">
  <xs:restriction base="xs:string">
    <xs:enumeration value="ABC"/>
    <xs:enumeration value="DEF"/>
    <xs:enumeration value="GHI"/>
    <xs:enumeration value="JKL"/>
  </xs:restriction>
</xs:simpleType>
```

Объявление данного именованного простого типа вводит нестандартный тип данных. Этот тип базируется на типе "string" W3C XML-Схемы, к которому применены фасеты ограничения допустимых значений. Каждый из фасетов "enumeration" содержит одно значение. А вместе они определяют полный набор допустимых значений для данного простого типа. Данный простой тип определен для всей схемы (т. е. "глобально"). Поэтому этот глобально объявленный простой тип можно использовать повторно в любой точке схемы при помощи ссылки. В таком случае ссылающийся на него элемент становится элементом этого типа.

Глоссарий

Абстрактная кардинальность (abstract cardinality) подобна неограниченной кардинальности. Это форма кардинальности, в которой степень кардинальности (наибольшее возможное количество повторений) неизвестна. С точки зрения схемы XML, это означает, что значение атрибута maxOccurs в этом случае равно "unbounded".

Абстрактная форма контейнера (abstract container form) – архитектурная форма или шаблон, включающий повторяющиеся контейнеры-элементы с одинаковыми именами, в определенном контексте. Абстрактная форма контейнера подразумевает использование до некоторой степени абстрактных имен элементов (имен "тегов"). В качестве примера абстрактного имени можно привести имя "<PersonName>"(полное имя), используемое вместо более конкретного имени "<EmployeePersonName>"(полное имя сотрудника).

Архитектурная форма контейнера (architectural container form) – применение или адаптация заслуживающих внимания и повторяющихся шаблонов к структурной модели XML.

Атрибут (attribute) – именованный контейнер для данных. XML-атрибут синтаксически похож на HTML-атрибут (пара "имя-значение"). Важно отметить, что XML-атрибут определяется внутри конкретного элемента и не может повторяться.

Привязка (binding) – сцепление или интеграция объектов. Часто применяется в концепции данных и презентации, где данные привязываются к Web-странице.

B2B (Business-to-Business – бизнес для бизнеса) описывает тип Web-сайта, создаваемого некоторым бизнесом, основными клиентами которого являются другие бизнесы.

B2C (Business-to-Consumer – бизнес для потребителя) описывает тип Web-сайта, создаваемого некоторым бизнесом, и предназначенного для обычных потребителей (физических лиц или групп)[1].

Camel case – способ использования регистра символов в фразе, слове, или терме. В зависимости от источника, техника использования может отличаться. Чаще всего он предполагает следующее: первый символ каждого выделяемого слова находится в верхнем регистре (заглавный), а все остальные – в нижнем регистре (прописные).

---

[1] B2B, B2C – модели бизнеса, определяющие то, на каких клиентов некоторый конкретный бизнес будет ориентирован. В B2B-модели в качестве клиента бизнеса выступает другой бизнес (в терминах информационных систем данная модель подразумевает взаимодействие систем различных компаний). В B2C-модели в качестве клиента бизнеса выступает потребитель, например физическое лицо. Типичным примером B2C-модели является электронный магазин. – *Примеч. ред.*

Дочерний элемент (child element) – дочерний или подчиненный элемент некоторого родительского элемента.

Слово класса (class word) – часть имени, обозначающая класс или тип. При использовании в имени элемента, слово класса чаще всего предназначено для описания отдельных допустимых значений, диапазона допустимых значений, или типа данных.

Поток щелчков (click stream) напоминает путь к ресурсу, вводимый пользователями при обращении к Web-сайту. Информация о потоке щелчков иногда собирается и используется предприятием для маркетинговых исследований, связанных с Web-сайтом.

Компонент (component) – составная часть более крупного объекта или концепции. В примере с почтовым адресом компонентом является, например, название города. Концептуально, сочетание всех непосредственно входящих компонентов представляет собой исходный объект, частями которого они являются.

Компонентная структурная модель XML (component XML structure model) – модель структуры, состоящая из наборов подобных или взаимосвязанных контейнеров, объединенных в именованные группы. XML-схемы, описывающие эти группы элементов, становятся кандидатами на повторное использование в качестве подсхем.

Контейнер (container) – в контексте XML, это обычно элемент, или атрибут. То есть объект, содержащий данные, другой контейнер, то и другое вместе или ничего.

Содержимое или контент (content) – значения данных, содержащиеся в XML-документе.

Контекст (context) – определение или прикладное использование объекта, информации, набора или элемента данных. Контекст может быть явным (четко обозначенным и описанным), или неявным (подразумеваемым по некоторому набору элементов данных). В XML контекст часто представлен схемой-словарем или родительским контейнером.

Междоменное повторное использование (cross-domain reuse) – широкомасштабное повторное использование. Повторное использование информационного актива в разных прикладных системах или доменах.

CRM (Customer Relationship Management – управление связями с клиентами) – управление связями с клиентами объединяет такие бизнес-функции, как маркетинг, продажи, персонализация, обслуживание клиентов, удержание клиентов, и анализ. Возможность определять, адресоваться, привлекать, обслуживать, захватывать, наращивать, и удерживать клиентов – вот основные задачи управления связями с клиентами.

Концепция данных (data concept) – общая абстрагирующая классификация для типа или набора данных. К примерам концепций данных относятся “почтовый адрес”, ”полное имя”, ”телефонный номер” и т. д.

Составляющая концепции данных (data concept particle) – определенный набор составных частей, фрагментов информации, или компонентов концепции данных. К примеру, имя и фамилия – составляющие концепции данных "полное имя".

Диспаритет данных (data disparity) – характерные различия между концепциями данных, элементами данных и характеристиками их метаданных. Диспаритет данных – заметная отличительная черта между похожими, но не идентичными частицами информации. При движении к интеграции данных, диспаритет данных необходимо устранять.

Составная часть данных (data particle) – часть, фрагмент, или компонент концепции данных. Составная часть данных часто рассматривается как "атом" (т. е. представляет собой наименьший возможный результат декомпозиции данных).

Стандарт данных (data standart) – распространенные и общепринятые: определение, набор кодов, форма, структура, или набор допустимых значений. Стандарты данных можно применять к концепциям данных, составляющим концепций данных, или составным частям данных.

Степень кардинальности (degree of cardinality) – заданный предел, наименьшее или наибольшее возможное количество повторений элементов данных.

Документ (document) – в контексте XML, это документ, транзакция, сообщение, файл, или другой подобный объемлющий контейнер для информации.

DTD (Document Type Definition – определение типа документа) – вид XML-схемы, базирующийся на SGML. Синтаксис DTD-схемы отличается от синтаксиса XML и W3C XML-Схемы.

Домен (domain) – область видимости, контекст или набор подобных или взаимосвязанных концепций данных, функций, структур данных, приложений или систем.

Элемент (element) – именованный контейнер для данных. XML-элемент имеет ту же синтаксическую форму, что и HTML-элемент ("<имя-элемента>...</имя-элемента>", или "<имя-элемента/>").

Интеграция предприятия (enterprise integration) – цель и процесс согласования и интеграции информационных активов различных систем предприятия.

Предприятие (enterprise) – бизнес или объединение взаимосвязанных бизнесов, действующих как единое целое. Предприятие включает ресурсы (людские и др.), процессы, информацию, цели и стратегии.

XML (eXtensible Markup Language – расширяемый язык разметки) – язык, возникший как согласованное подмножество SGML. XML часто называют самоописывающим языком метаданных. XML получил популярность как язык, используемый в Web-приложениях и язык для определения транзакций электронной коммерции. За дополнительной информацией обращайтесь к Web-сайту WWW-консорциума (www.w3c.org).

XSL, XSLT (eXtensible Stylesheet Language – расширяемый язык стилей) – XML-технология, позволяющая осуществлять навигацию по ссылающемуся XML-документу, и применение к нему презентационного формата или логики преобразования.

Гибкость (flexible) – архитектурная характеристика; когда структура может быть расширена или сжата.

G-CRM (Global Relatioship Management – глобальное управление связями с клиентами) расширяет концепцию управления связями с покупателями, включая в нее учет глобальных и локальных (локализация) характеристик и предпочтений покупателя.

Горизонтальная структурная модель XML (horizontal XML structure model) – модель структуры, в которой для хранения данных используются преимущественно контейнеры-атрибуты. Ее называют также "плоской".

Гибридная форма контейнера (hybrid container form) – архитектурная форма контейнера, в которой сочетаются лучшие характеристики и возможности всех остальных форм контейнера.

Гибридная структурная модель XML (hybrid XML structure model) – модель XML-структуры, сочетающая лучшие характеристики и возможности всех остальных моделей структуры.

HTML (HyperText Markup Language – язык разметки гипертекста) – приложение SGML. Разработан для упрощения презентации информации в Web.

HTML-форма (HTML form) – форма, описанная на языке HTML. Расширение HTML, позволяющее осуществлять взаимодействие с пользователем (ввод данных, выбор, ограниченную верификацию и передачу данных Web-серверу или аналогичному приложению). В HTML-формах используются декларативные элементы (<INPUT/>, <TEXTAREA/>, и <SELECT/>). Элементы HTML-форм также называют "полями HTML-форм".

Поле HTML-формы (HTML form field) – неофициальный синоним "элемента HTML-формы". Используемые на HTML-форме поля позволяют вводить данные, делать выбор, и производить ограниченную верификацию в Web-браузере. Поля HTML-форм включают: <INPUT/>, <TEXTAREA/>, и <SELECT/>.

Экземпляр (instance) – некоторая реализация. Экземпляр документа – это XML-документ, заполненный конкретными данными.

Интеграция (integration) – процесс, ведущий к порождению, комбинированию, объединению, обмену, преобразованию и обработке информационных активов несколькими системами предприятия.

Интероперабельный (interoperable) – позволяющий обеспечивать связь, совместную обработку или совместное использование на или между различными операционными системами, сетевыми окружениями, базами данных или другими технологиями.

Локальные параметры (locale) – комбинация характеристик, описывающих географические особенности и предпочтения международного пользователя. Не существует четко определенных стандартов, относящихся к локальным параметрам. Однако в большинстве случаев, они могут быть описаны, как комбинация географических признаков (страна, область, город и т. д.), предпочитаемого языка общения, вида валюты и метрической системы.

Метаданные (metedata) – характеристики, форма, допустимые значения, стандарты и правила, описывающие и определяющие данные. Часто упоминаются как "данные о данных".

Модальность (modality) описывает концепцию обязательности или необязательности (опциональности).

Родительский элемент (parent element) – элемент-владелец (или вышестоящий) для одного или нескольких дочерних элементов.

Синтаксический анализатор (parser) – в случае XML, служебная программа, проверяющая синтаксис XML-документа и позволяющая приложениям обрабатывать документ (осуществлять навигацию, извлекать данные и вставлять данные).

Шаблон (pattern) – характерная форма или оболочка, которую можно адаптировать для конкретных нужд и неоднократно использовать.

RDBMS (Relational DataBase Management System – система управления реляционными базами данных) – тип базы данных, основанный на матрицах, кортежах и реляционной алгебре. Появление концепции реляционных баз данных чаще всего приписывают работе Кодда. Реляционная база данных позволяет не только сохранять информацию, но и поддерживает возможность связывать данные между собой и видеть отношения между данными.

Повторное использование (reuse) – возможность использовать что-либо более одного раза. Более узко – возможность использовать более одного раза то, что было разработано специально с прицелом на возможность повторного использования.

Разработка с прицелом на повторное использование (reuse engineering) – проектирование, разработка и развитие информационного актива, выполняемые специально с расчетом на возможность его неоднократного использования.

Собственно повторное использование (reuse harvesting) – установление возможности, проверка, и использование повторноиспользуемого информационного актива.

Жесткая форма контейнера (rigid container form) – архитектурная форма контейнера, складывающаяся из элементов с конкретизированными именами, определения которых не предполагают возможности повторения этих элементов.

Корневой элемент (root element) – элемент самого верхнего уровня XML-документа.

Сервер (server) – компьютерное устройство или прикладной процесс, обеспечивающий обслуживание по запросам. Могут существовать серверы самого разного типа (Web, приложений, данных, и предприятия).

SOAP (Simple Object Access Protocol – простой протокол доступа к объектам) – протокол на основе XML, используемый для запросов к Web-сервисам.

Заданная кардинальность (specific cardinality) – кардинальность, степень которой определенна конечными границами или порогами. В синтаксисе XML-схемы существуют атрибуты manOccurs и maxOccurs, значения которых и определяют степень кардинальности повторяющихся элементов.

Строго типизированные данные (strongly typed data) – данные, которые были описаны или определены с детальными характеристиками, такими, как тип данных (например, string, integer, date и т. д.).

Тег (tag) – синтаксическая форма XML для задания имени контейнера или структуры.

Стандарт именования (taxonomy) – схема и форма классификации, часто применяемая для создания имен.

UDDI (Universal Description Discovery and Integration – универсальное описание, обнаружение и интеграция) – подход к описанию Web-сервисов на основе репозитория.

Пользовательский интерфейс (user interface) – способ взаимодействия с пользователем, реализованный посредством одной или нескольких технологий. Пользовательский интерфейс определяется или располагается на уровне представления.

Верификация (validation) – для XML, процесс, выполняемый синтаксическим анализатором, и сравнивающий XML-документ со схемой, на которую он ссылается. Сообщения о нарушениях и ошибках передаются обратно в приложение, вызвавшее синтаксический анализатор. Для выполнения данного процесса требуется "верифицирующий" синтаксический анализатор.

Вертикальная структурная модель XML (vertical XML structure model) – структурная модель, складывающаяся преимущественно из контейнеров-элементов. Визуально такая модель может быть представлена как направленная сверху вниз.

Web-страница (Web page) – определенный набор информации (также называемый контентом или содержимым), идентифицируемый как ресурс WWW и доступный, таким образом, пользователям. Web-страница обычно обслуживается Web-сервером. Web-сайт может содержать большое количество Web-страниц.

Web-сервер (Web server) – разновидность сервера. Web-сервер выдает или обслуживает Web-страницы, отвечая на запросы на страницы обычно имеющие форму универсального указателя ресурса (URL).

Web-сервис (Web service) – программа, функция или логика, обозначенная как ресурс WWW, к которому можно обратиться посредством сообщения. Web-сервис получает запросы и возвращает ответы на них.

WSDL (Web Service Description Language – язык описания Web-сервисов) – язык, основанный на XML, который можно использовать для описания Web-сервиса, его интерфейса и методов обращения к нему.

Web-сайт (Web site) – набор Web-страниц или аналогичных Web-ресурсов. Web-сайт обычно идентифицируется универсальным указателем ресурсов (URL).

Внутридоменное повторное использование (within-domain reuse) – повторное использование информационного актива внутри определенного контекста, прикладной системы или домена.

XML-атрибут (XML attribute) – вид контейнера XML, который определяется внутри XML-элемента. XML-атрибут не может существовать сам по себе (без соответствующего элемента). XML-атрибут может содержать данные или ничего не содержать (пустой). XML-атрибут не может содержать другой атрибут или элемент.

XDR (XML Data Reduced – ограничение XML данных) – вид XML-схемы, предшествующий разработке W3C XML-Схемы.

XML-документ (XML document) – файл, документ, транзакция или сообщение, определенные с использованием синтаксиса XML. Объемлющий контейнер для контейнеров XML и значений данных.

XML-элемент (XML element) – наиболее часто используемый тип контейнера XML. XML-элемент определяется в форме имени тега, заключенного в угловые скобки. XML-элемент можно определить как содержащий данные, другие элементы, то и другое вместе, или нечего. XML-элемент можно также определить как пустой, то есть элемент, который не может содержать никакого значения.

XML-Схема (также известная как W3C XML-Схема) (XML Schema) – основанный на XML вид схемы, включающий правила и ограничения для соответствующего XML-документа. XML-Схема может включать расширенный набор характеристик метаданных (формат, структура, правила и допустимые значения). Как и определено в майских 2001 года рекомендациях WWW консорциума.

Структурная модель XML (XML structure model) – модель структуры, применяемая к XML-документу, транзакции или файлу.

Библиография

American National Standards Institute (ANSI), X3 Working Group. X3.138 Information Resource Dictionary System. *Data Entity Naming Conventions,* Special Publication 500-149.

Bean J. *Currency Data Concept, Web Globalization Guide Framework.* James Bean ©1996-2002. Находится на *http: //www.globalwebarch.com/*

Bean J. *Engineering Global E-Commerce Sites.* Morgan Kaufmann, San Francisco, 2003.

Bean J. *XML Globalization and Best Practices.* Active Education, Colorado, U.S., 2001.

Crawford M, Egan D, Jackson A. *Federal Tag Standards for Extensible Markup Language.* GS018T1, p 27. Logistics Management Institute (LMI), June 2001. Находится на *http://xrnl.gov/*

IBM DB2 Universal Database, SQL Reference, Version 71, SC09-2974-01. IBM, 1993, 2001. DB2 Universal Database for OS/390, z/OS, SQL Reference, Version 7, SC26-9944-01. IBM, 1982, 2001. Находится на *http: //www. ibm. com/*

International Standards Organization (ISO). ISO 3166, Country Codes, TC46 Technical Committee. Находится на *http://www.iso.ch/iso/en/stdsdevelopment/tc/tclist/TechnicalCommitteeDetailPage.Technical CommitteeDetail?COMMID=1757*

International Standards Organization (ISO). ISO 3166, Country Codes, MA Maintenance Agency. Находится на *http://www.din.de/gremien/nas/ nabd/iso3166ma/ a3ptnorm. html*

International Standards Organization (ISO). ISO 4217, Currency Codes, TC68 Technical Committee. Находится на *http://www.iso.ch/iso/en/stdsdevelopment/tc/tclist/TechnicalCommitteeDetailPage. TechnicalCommitteeDetail?COMMID=2183*

International Standards Organization (ISO). ISO 4217, Currency Codes, UN/ECE United Nations Economic Commission for Europe. Находится на *http://www.unece.org/cefact/rec/rec09en.htm*

International Standards Organization (ISO). ISO 8601, Date and Time, TC154 Technical Committee. Находится на *http://www.iso.ch/iso/en/stdsdevelopment/tc/tclist/TechnicalCommitteeDetailPage.Technical-CommitteeDetail?COMMID=382 7*

Internet Explorer 6. Microsoft, 1995-2000 (использовался для создания различных сопровождающих текст рисунков). Находится на *http://www.microsoft.com*

Karlsson, E-V. *Software Reuse—A Holistic Approach.* John Wiley & Sons, New York, 1995.

Manual for Data Administration, Special Publication 500-208, InterNational Committee for Information Technology Standards (INCITS). Находится на *http://www.x3.org/incits/*

Microsoft SQL Server, Transact-SQL Reference (on-line), Data Types, Updated September 2001. Находится на *http: //msdn.microsoft.com/ library/default.asp?url=/library/en-us/tsqlref/ts_da-db_7msw.asp*

Object Management Group. OMG Modeling and Metadata Specifications. XML Metadata Interchange (XMI®). Находится на *http: //www. omg. org/*

Oracle 9i Applications Developer Guide, release 1 (9.0.1), part A88876-02. Oracle Corporation, 1996-2001. Находится на *http: //www. oracle. com/*

PowerDesigner, Инструмент моделирования данных. Sybase. Находится на *http: //www.sybase.com*

Simple Object Access Protocol, SOAP Version 1.2. World Wide Web Consortium (W3C), 2002. Находится на *http://www.w3.org/TR/soapl2-part//, http://www.w3.org/TR/soapl2-part2/, и http://www.w3.org/TR/soap12-af/*

Sybase Adaptive Server Enterprise 12.5. Transact-SQL, Content ID 1009196, revised Feb. 1, 2002. Sybase Inc., 2002. Находится на *http: //www.sybase.com/*

Universal Description Discovery and Integration of Web Services. Ariba, International Business Machines, and Microsoft. UDDI, 2000. Находится на *http: //www.uddi.org/*

Web Services Description Language (WSDL) version 1.2. World Wide Web Consortium (W3C), 2002. Находится на *http://www.w3.org/TR/wsdl12/*

World Wide Web Consortium (W3C). eXtensible Markup Language (XML) 1.0, 2nd ed. W3C Recommendation, October 6, 2000. Находится на *http://www. w3. org/TR/2000/REC-xml-20001006*

World Wide Web Consortium (W3C). HTML 4.01 Specification. W3C Recommendation, December 24, 2001. Находится на *http://www.w3.org/ TR/1999/REC-html401-19991224*

World Wide Web Consortium (W3C). XHTML™ 2.0. W3C Working Draft, August 5, 2002. Находится на *http://www.w3.org/TR/2002/WD-xhtml2-20020805*

World Wide Web Consortium (W3C). XML-Data (например, "XML Data Reduced"). Note, January 5, 1988. Находится на *http://www.w3.org/TR/1998/NOTE-XML-data-0105/*

World Wide Web Consortium (W3C). XML Schemas. W3C Recommendation, May 2, 2001. Находится на *http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/*(structures); *http://www. w3.org/TR/2001/REC-xmlschema-2-200105021* (data types).

XML Spy, Инструмент разработки и редактирования XML . Altova, Inc. Находится на *http://www.xmlspy.com*

Предметный указатель

П

T

Содержание

ИЗДАТЕЛЬСТВО
«КУДИЦ-ОБРАЗ»

Тел.: (095) 333-82-11; ok@kudits.ru, http://books.kudits.ru

ПРИОБРЕТАЙТЕ КНИГИ У НАШИХ ПАРТНЕРОВ

Великий Новгород
ул. Б. Санкт-Петербургская, 44,
(81622) 73-188 доб. 34

Вологда
ООО "Венал" *Оптово-розничная торговля*,
ул. Челюскинцев, д. 9 (8172) 75-21-43

Воронеж
"Книжный мир семьи", пр-т. Революции, 58,
(0732) 51-28-90

Донецк
ЧП Карымов Ратмир Гибадулович,
(10-380-62) 381-9232

Екатеринбург
"Книжный мир", ул. 8 Марта, 8г, (3432) 71-18-87
ООО "КДК Дом книги", ул. Блюхера, 51, 359-41-04

Иркутск
"Знание" ул. Ленина, 15, (3952) 24-28-05
"Продалит" ПБОЮЛ Перевозников,
(3952) 23-28-62, 59-13-80, 59-09-90

Калининград
ООО "ПРОМЭКСПОРТИМПОРТ" (0112) 35-37-66

Киев
"Микроника", ул. М. Расковой, 13, (044) 517-73-77
"Технокнига", (044) 268-53-46

Краснодар
"Мир книги", ул. Буденного, 147
"Колос", ул. Красная, 100, (8612) 59-41-32

Минск
Издательство "ТетраСистемс" ул.Железнодорожная,
дом 9, (375-17) 219-73-90, 219-74-01
Книжный магазин books@tut.by, (375-17) 219-73-88
Интернет-магазин http://www.oz.by

Москва
"ОПТИМА+",
(095) 333-65-67, ok@kudits.ru; http://books.kudits.ru

Новосибирск
"Книжный пассаж", ул. Ленина, 10а, (3832) 29-50-30
"Сибирский Дом Книги",
Красный пр-т, 153, (3832) 26-62-39
"Книжный мир", пр-т К. Маркса, 51

Омск
"Книжный Мир", ул. Ленина, 17/19, (3812) 24-32-54

Пермь
"Образование", ул. Солдатова, 37, (3422) 45-96-55
ООО "АКМА" *Оптово-розничная торговля* (3422) 90-93-02,
41-24-60

Ростов-на-Дону
"Мир книги", Ворошиловский пр-т, 33; (8632) 62-54-61
"Деловая литература", (8632) 62-36-55

Самара
"Техническая книга" (8462) 24-29-80

Санкт-Петербург
"Санкт-Петербургский Дом книги"
Невский проспект, 28, (812) 312-01-84
издательство "Наука и Техника", пр. Обуховской Обороны, 107,
(812) 567-70-25, 567-70-26

Саратов
"Книжный Мир"; пр-т Кирова, 32, (8452) 32-98-14

Ставрополь
"Книжный Мир", ул. Мира, 337, (8652) 35-47-90

Таганрог
"Компьютерная книга", ул. Чехова, 31, (8634) 37-13-12

Томск
"Книжный Мир", ул. Ленина, 141, (3822) 51-07-16

Уфа
ООО ПКП "Азия", тел./факс: (3472) 50-39-00
Оптовая торговля Ул. Зенцова, 70
Розничная торговля Магазин "Оазис", ул. Чернышевского, 88
Магазин "Книжник", пр. Октября, 106

Ханты-Мансийск
Магазин "Книги", ул. Ленина, 39

Харьков
Книжный рынок "Райский уголок", ул. Клочковская, 28,
(0572) 549-116

Челябинск
"Книжный Мир", ул. Кирова, 90, (3512) 33-19-58

Ярославль
Магазин "Наука", ул. Володарского, 63, (0852) 25-95-04

ЗАКАЗ КНИГ НАЛОЖЕННЫМ ПЛАТЕЖОМ

Издательство «КУДИЦ-ОБРАЗ» осуществляет рассылку книг по почте.

Заказы принимаются по адресу: 121354, Москва, а/я 18; или через Интернет-магазин http://books.kudits.ru.

Заказы из регионов России с авиадоставкой, а также заказы из стран ближнего и дальнего зарубежья обслуживаются только по предварительной оплате.