Новейшие технологии

CD-ROM

XML и SOAP

программирование для серверов BizTalk™

Брайан Трэвис

РУССКАЯ РЕДАКЦИЯ

Microsoft Press

Миллеру, который недавно научился играть в шахматы и скоро точно станет чемпионом мира.

XML SOAP Programming for BizTalk Servers

Build business-to-business e-commerce solutions using XML, SOAP and BizTalkFramework

Brian E. Travis

Microsoft Press

Новейшие технологии

XML SOAP: программирование для серверов BizTalk

Брайан Трэвис

BHV РУССКАЯ РЕДАКЦИЯ

УДК 004.43
ББК 32.973,26-018
Т59

Трэвис Б.

Т59 XML и SOAP: программирование для серверов BizTalk. Новейшие технологии /Пер. с англ, — М.: Издательско-торговый дом «Русская Редакция», 2001. — 496 стр.: ил.

ISBN 5–7502–0129—5

Книга посвящена технологиям, лежащим в основе электронной коммерции и современного Интернет-бизнеса. Здесь подробно описаны стандарты иерархического структурирования информации, схемы представления данных и идеология BizTalk, освещены вопросы интеграции и создания приложений на платформе BizTalk. Материал представлен на конкретных примерах разработки клиента и сервера SOAP и создания решения для автоматизации обмена XML-документами между несколькими Web-сайтами. Приложения содержат богатый справочный материал, включая окончательные спецификации инфраструктуры BizTalk и протокола SOAP.

Издание адресовано менеджерам Интернет-проектов, разработчикам программного обеспечения электронной коммерции и всем интересующимся современными методами структурирования информации.

Книга состоит из 12 глав, 4 приложений и предметного указателя. Прилагаемый к книге компакт-диск содержит программное обеспечение, примеры приложений, документацию и электронную версию книги.

УДК 004.43
ББК 32.973.26—018

Подготовлено к изданию по лицензионному договору с Microsoft Corporation, Редмонд, Вашингтон, США.
Спецификация Simple ObjectAccess Protocol (SOAP) 1.1 © 2000, DevelopMentor, International Business Machines Corp., Lotus Development Corp., Microsoft Corp., UserLand Software, Inc. Все права охраняются.
BizTalk™ Framework 2.0: спецификация документов и сообщений © 1999-2000, Microsoft. Все права охраняются.
Приложение А основано на статье, предоставленной компанией OmniMark. С 1988-2000, OmniMark Technologies Corporation. Все права охраняются.
ActiveX, JScript, Microsoft, Microsoft Press, MSDN, MS-DOS, Visual Basic, Visual C++, Visual InterDev, Visual SourceSafe, Visual Studio, Win32, Windows и Windows NT являются товарными знаками или охраняемыми товарными знаками корпорации Microsoft в США и/или других странах. Все другие товарные знаки являются собственностью соответствующих фирм.

Все названия компаний, организаций и продуктов, у также имена лиц, используемые в примерах, *вымышлены* и не имеют никакого отношения к *реальным компаниям*, организациям, продуктам и лицам.

© Оригинальное издание на английском языке, Microsoft Corporation, 2000
© Перевод на русский язык, Microsoft Corporation, 2001
© Оформление и подготовка к изданию, издательско-торговый дом «Русская Редакция», 2001

ISBN 0–7356–1126–2 (англ.)
ISBN 5-7502-0129-5

Оглавление

Введение XVI

Часть I Применение XML в электронной коммерции . . 1

Глава 1 Перспективы применения XML в бизнесе 2

Межкорпоративный электронный бизнес 4
Разделение данных и процесса 5
XML и трехуровневая архитектура 6
Уровень данных 7
Клиентский уровень 7
Промежуточный уровень 8
Профиль подписчика 8
SQL и XML 10
HTML как XML 11
Службы электронной коммерции 12
Информационное наполнение 13
Простой протокол доступа к объектам 18
XML для поставщиков информации 18
Стандарты 23
Создание страницы 27
XML и управление содержанием 27
XML и сбор информации 28
Электронный документооборот 32
XML и EDI 33
XML и электронный бизнес 34
Общий язык 34
XML и языки обработки 36

Глава 2 Предыстория XML 39

HTML: достоинства 39
HTML: недостатки 40
SGML: достоинства 42
SGML: недостатки 44
Нужен XML 45
Мифы об XML 46
XML — это язык разметки 46
XML — это только для Web 46
HTML — это подмножество XML 47
XML — это рекламная уловка 47

Глава 3 Стандарты . . . 49
Международная организация стандартов . . . 49
OASIS . . . 50
W3C . . . 50
Стандарты W3C . . . 52
XLink и XPointer . . . 53
Схемы . . . 54
Пространства имен . . . 55
XSL, XSLT и XPath . . . 56
XQL . . . 57
DOM . . . 57
IETF . . . 57
Стандарты де-факто . . . 58
Разработка отраслевых схем . . . 59

Глава 4 Синтаксис XML . . . 61
Правильный документ XML . . . 61
Объявление XML . . . 62
Открывающий и закрывающий тэги . . . 62
Корневой элемент . . . 63
Чувствительность к регистру . . . 63
Вложенность . . . 64
Особые символы . . . 64
Атрибуты . . . 65
Пустые элементы . . . 67
Комментарии . . . 67
Примеры правильных документов . . . 68
Действительные документы XML . . . 68
Определение типа документа (DTD) . . . 70
Пространства имен . . . 73
Схемы . . . 79
XDR — сокращенная версия XML Data . . . 79

Глава 5 Работа с XML из приложений . . . 85
Объектная модель документа . . . 85
Как работает DOM . . . 86
Создание объекта DOM . . . 91
Работа с документом . . . 93
Доступ к элементам документа . . . 94
Упражнение: работа с DOM средствами

Visual Basic . . . 97
Другие операции с документом . . . 105
Событийные модели . . . 112

Глава 6 XSL . . . 115
История XSL . . . 115
Как работает XSL? . . . 116
XS L в реализации Microsoft . . . 118
Выделение спецификаций XSLT и XPath . . . 118
Как работает XSLT . . . 119
Рекурсия . . . 123
Пример использования XSLT . . . 127
Пример: преобразование бизнес-документа . . . 138

Часть II Обмен XML-сообщениями . . . 145

Глава 7 Сетевые службы . . . 146
Значение интеграции . . . 150
Слабосвязанные службы . . . 150

Глава 8 Создание Web-службы с помощью SOAP . . . 155
HTTP . . . 156
Метод GET . . . 156
Метод POST . . . 158
Порты . . . 160
Брандмауэры . . . 161
Структура сообщения SOAP . . . 161
Пример: пишем сервер новостей . . . 162
SOAP-клиент . . . 163
Пишем SOAP-сервер на языке OmniMark . . . 170
Программа SOAPServer.xom . . . 171
Программа getXMLNews.xom . . . 179
Тестирование службы . . . 188

Глава 9 Инфраструктура BizTalk . . . 189
Спецификации документов и сообщений BizTaik . . . 191
Использование BizTalk в транзакциях <<бизнес — бизнес» . . . 192
Подготовка заказа . . . 192
Обработка заказа . . . 194
Эволюция обмена информацией . . . 195
Заказ с использование инфраструктуры BizTalk . . . 196
Уведомление о получении заказа . . . 201

Подтверждение заказа . . . 202
Возможности автоматизации . . . 204

Часть III Сервер BizTalk . . . 207

Глава 10 Требования к серверу BizTalk . . . 208
Надежная пересылка по любому протоколу . . . 209
Зашита информации . . . 209
Маршрутизация документов . . . 210
Документооборот . . . 211
Синхронная и асинхронная связь . . . 211
Очередь сообщений . . . 212
Пакетный режим . . . 212
Отслеживание документов . . . 212
Управление взаимодействием с партнерами . . . 213
Масштабируемость . . . 213
Преобразование документов . . . 213
Встраивание модулей других производителей . . . 214
Взаимодействие с EDI-системами . . . 214
Прикладные интерфейсы . . . 215
Адаптация к изменениям . . . 215
Ориентация на нужды пользователей . . . 216

Глава 11 Создание сервера BizTalk . . . 217
Информационные потоки . . . 219
Архитектура . . . 221
Базы данных . . . 222
Установка системы . . . 225
Ввод заявки на покупку . . . 226
prEnter.asp . . . 226
prProcess.asp . . . 230
ROPE и SDL . . . 234
Программа генерации заказа на покупку . . . 238
Процедура Form.Load . . . 238
Процедура createPO_Click . . . 239
Функция sendMsg . . . 242
Обработка заказа на покупку . . . 244
Автоматизированная обработка заказа на покупку . . . 245
Обработка подтверждения заказа . . . 256

Глава 12 Microsoft BizTalk Server 2000 261
BizTalk Editor 262
Пример: создание схемы заказа 262
BizTalk Mapper 266
Пример: отождествление двух схем заказа 266
Другие средства BizTalk Server 2000 273
Средства обработки данных BizTalk Server 2000 274
Размещение заказа 274
Проверка наличия товара и заказ у производителя 276
Пересылка подтверждения заказчику 277

Часть IV Справочные материалы 279

Приложение А OmniMark для нетерпеливых 280
Потоковый язык программирования 281
Обработка XML 284
Создание областей ввода 285
Создание областей вывода 286
Компоненты OMX 289
Переменные и типы данных 291
Ввод-вывод и переменные 293
Структуры данных 296
Ссылки 298
Условные операторы 300
Операторы цикла 302
Арифметические операторы
и операторы сравнения 303
Сопоставление с образцом 304
Переменные шаблона 308
Функции 309
Генерация и обработка исключений 311
Перехват программных ошибок 312
Обработка исключений 315
Передача дополнительной информации 316
Очистка памяти после завершения работы 317
Группы правил 321
Константы и макросы 323
Финансовые вычисления 325
Сетевое программирование 326
Простое приложение-сервер 328

Ошибки и восстановление . . . 332
Простая программа-клиент . . . 334
Клиенты для распространенных серверов . . . 335
Программирование CGI . . . 335
Псевдокомментарий . . . 335
Конфигурирование Web-серверов для выполнения программ CGI, написанных на OmniMark . . . 337
Internet Information Services . . . 337
Personal Web Server для Windows 95/98 . . . 338
Web-сервер Apache . . . 339
Apache для Windows 2000 . . . 339
Apache для UNIX/Linux . . . 340
Местонахождение системных библиотек OmniMark . . . 340
Создание CGI-программ ни OmniMark . . . 341
Библиотека CGI OmniMark . . . 341
Получения данных от Web-сервера . . . 341
Методы GET и POST . . . 341
Доступ к информации методов GET и POST . . . 341
Доступ к переменным среды CGI . . . 342
Передача результатов Web-серверу . . . 343
Формирование заголовка HTTP . . . 343
Передача двоичных файлов . . . 344
Небуферизированный стандартный ввод и вывод . . . 344
Повышение надежности CGI-программ . . . 345
Перехват программных ошибок . . . 345
Предотвращение просмотра исходного кода . . . 346
Соберем все вместе . . . 346
Использование простых форм . . . 346
Доступ к внешним источникам данных . . . 347
Файлы параметров и проектов OmniMark . . . 349
Файл параметров OmniMark . . . 349
Использование псевдокомментариев в файле параметров . . . 350
Файл проекта OmniMark . . . 350
Компиляция CGI-программ . . . 351
Дополнительная информация об OmniMark . . . 352

Приложение Б BizTalk™ Framework 2.0: спецификация документов и сообщений . . . 353

1. Введение . . . 353

2. Область действия и структура спецификации . . . 355

2.1 Взаимосвязь со спецификацией BizTalk Framework 1.0 . . . 356

2.2 Модель версий . . . 357

3. Использование типов данных XML-схем . . . 357
4. Концепции . . . 358
4.1 Терминология . . . 358
4.2 Логическая структура . . . 361
5. Структура документа BizTalk . . . 362
6. Тело документа BizTalk . . . 363
7. Элементы заголовка документа BizTalk . . . 365
7.1 Источник и пункт назначения документа . . . 365
7.2 Идентификация документа и его свойства . . . 366
7.3 Доставка . . . 369
7.4 Каталог документа . . . 370
7.5 Управление обработкой документа . . . 372
8. Гарантированная доставка документов BizTalk . . . 374
8.1 Структура и содержание уведомлений . . . 375
8.1.1 Уведомление о доставке . . . 375
8.1.2 Подтверждение . . . 377
8.2 Поведение отправителя и получателя . . . 379
8.2.1 Поведение отправителя . . . 380
8.2.2 Поведение получателя . . . 381
8.2.3 Сроки доставки, уведомления и подтверждения . . . 382
9. Документы BizTalk с вложениями . . . 384
9.1 Структура составного документа MIME . . . 384
9-2 Структура декларации . . . 386
10. Защита документов и сообщений BizTalk . . . 386
10.1 S/MIME . . . 389
11. Транспортировка . . . 391
11.1 HTTP . . . 391
11.1.1 Пример BizTalk-сообщения . . . 392
11.1.2 Пример BizTalk-сообщения с вложениями . . . 393
11.2 SMTP . . . 395
12. Цитированная литература . . . 396
i 2.1 Нормативные источники . . . 396
12.2 Ненормативные источники . . . 397
Дополнение. Схемы документа BizTalk . . . 397
1.1 XDR-схемы . . . 397
1.1.1 Элемент заголовка <endpoints> . . . 397
1.1.2 Элемент заголовка <properties> . . . 398
1.1.3 Элемент заголовка <services> . . . 399

1.1.4 Элемент заголовка <manifest> 399
1.1.5 Элемент заголовка <process> 400
1.1.6 Элемент заголовка <deliveryReceipt> 401
1.1.7 Элемент заголовка <commitmentReceipt> 401
1.1.8 Конверт SOAP 1.1 для документов BizTalk 402
1.2 XSD-схемы 403
1.2.1 Элемент заголовка <endpoints> 403
1.2.2 Элемент заголовка <properties> 404
1.2.3 Элемент заголовка <services> 405
1.2.4 Элемент заголовка <manifest> 406
1.2.5 Элемент заголовка <process> 407
1.2.6 Элемент заголовка <receipt> 407
1.2.7 Конверт документа BizTalk 408

Приложение В Web-сайты, посвященные XML и BizTalk 411

BizTalk.org 411
Прямая поддержка 411
Домашняя страница Microsoft BizTalk 412
XML.com 413
OASIS 413
XML.ORG 414
Информационный бюллетень <TAG> 414
XMLSOFTWARE 415

Приложение Г Simple Object Access Protocol (SOAP) 1.1 416

Резюме 416
Статус 417
1. Введение 417
1.1 Цели проекта 418
1.2 Обозначения 418
1.3 Пример сообщения SOAP 418
2. Модель обмена сообщениями SOAP 420
3. Связь с XML 421
4. Конверт SOAP 421
4.1.1 Атрибут SOAP encodingStyle 423
4.1.2 Версии конверта 423
4.2 Заголовок SOAP 424
4.2.1 Атрибуты заголовка 424

4.2.2 Атрибут actor . . . 424
4.2.3 Атрибут mustUnderstand . . . 425
4.3 Тело сообщения SOAP . . . 426
4.3.1 Связь элементов Header и Body . . . 426
4.4 Элемент Fault . . . 426
4.4.1 Коды ошибок SOAP . . . 427
5. Представление данных в SOAP . . . 428
5.1 Представление типов данных в XML . . . 429
5.2 Простые типы . . . 433
5.2.1 Строки . . . 434
5.2.2 Списки . . . 435
5.2.3 Байтовый массив . . . 435
5.3 Полиморфные ссылки . . . 436
5.4 Составные типы . . . 436
5.4.1 Составные значения, структуры и ссылки на значения . . . 436
5.4.2 Массивы . . . 440
5.4.3 Другие составные типы . . . 445
5.5 Значения по умолчанию . . . 447
5.6 Атрибут root . . . 447
6. SOAP и HTTP . . . 447
6.1 HTTP-запрос SOAP . . . 448
6.1.1 Поле заголовка HTTP SOAPAction . . . 448
6.2 HTTP-ответ SOAP . . . 448
6.3 Инфраструктура расширения HTTP . . . 449
6.4 Примеры использования SOAP и HTTP . . . 449
7. SOAP как средство удаленного вызова методов . . . 450
7.1 Удаленный вызов процедур и тело сообщения SOAP . . . 450
7.2 Удаленный вызов процедур и заголовок сообщения SOAP . 451
8. Вопросы защиты . . . 452
9. Цитированная литература . . . 452
А. Примеры конвертов SOAP . . . 453
А.1 Пример запроса вызова метода . . . 453
А.2 Примеры ответов . . . 454

Предметный указатель . . . 457

Об авторе . . . 467

Благодарности

Вообще-то издание этой книги было моей идеей, но сейчас я чувствую себя скорее дирижером хорошо сыгранного оркестра. Именно сотрудники издательства Microsoft Press превратили мои идеи в связный текст.

Я хотел бы особо поблагодарить Викки Тульмана, который, как заправский голливудский продюсер, умело балансировал на грани между «дарованием», т. е. мной, и суровой правдой издательского бизнеса. На протяжении всей работы Викки держал меня в курсе событий и ненавязчиво, но эффективно управлял всем процессом написания и редактирования.

Брайан Джонсон, технический редактор, блестяще отслеживал все мои действия, не давая мне слишком глубоко зарыться в технические вопросы. Kpovre того, Брайан убедил меня зарегистрировать доменные имена, упоминавшиеся в книге. Теперь, если вы попадете на сайт *myduckjokes.com* или *duckbarjokes.com*, вы точно будете знать, кто виноват во всем этом безобразии.

Первым читателем моей книги была Ребекка Маккей. Поскольку эта книга писалась в дюжине разных гостиничных номеров и множестве самолетных кресел, связность повествования несколько пострадала. Объяснить тебе, читатель, что я в действительности имел в виду, должна была именно Ребекка. К примеру, я часто употребляю слова «этот» и «тот». В редакторском экземпляре мы обнаружили множество комментариев Бекки вроде: «Автор опять употребляет слово *этот*. Что он имеет в виду на *сей* раз?»

Команда Microsoft Press — это предел мечтаний любого автора. Я надеюсь поработать с ними еще раз!

В работе над этой книгой мне очень помогли сотрудники компании OmniMark Technologies. Это естественно, так как она рекламирует их ведущий продукт. Технический директор Джон Макфэйден употребил все свое влияние, чтобы достать мне работающую версию OmniMark, которая поддерживает документы и схемы XML на мировом уровне. Как всегда, OmniMark Technologies удалось сделать сложное простым. А невозможное просто занимает у них чуть больше времени. Марк Бейкер из OmniMark тихо и спокойно помог мне написать приложение А «OmniMark для нетерпеливых», не обращая внимания на суету, обычную перед сдачей книги. Я в общем-то почти не трогал его текст, просто добавил мои любимые анекдоты про гусей,

Сотрудники Microsoft помогли мне разобраться в технологиях, упомянутых в книге. Кевин Макколл, проповедник BizTalk, вообще никогда не спит. Он ездит по миру, рассказывая людям, как с помо-

шью BizTalk решать проблемы сферы «бизнес — бизнес». Именно Кевин обратил мое внимание на существенный пересмотр спецификации BizTalk Framework, причем по закону Мэрфи это случилось уже после сдачи текста в издательство. К счастью, Викки дал мне еще несколько дней, чтобы переписать часть книги. В той же роли, но уже относительно SOAP выступил Джон Монтгомери — неиссякаемый источник информации о взаимосвязи BizTalk и SOAP. Роберт Колридж — ведущий архитектор ROPE в Microsoft — держал меня в курсе всех новостей. Кроме того, я хотел бы поблагодарить всех остальных сотрудников Microsoft, помогавших мне.

Мехтаф Озкан помогла мне разобраться с неприятной ошибкой в ASP, которая приводила к потере данных, передаваемых методом POST. Она работала не покладая рук, чтобы достичь правильного сочетания функций преобразования символов и избежать потери данных. Я также очень благодарен ей за моральную поддержку на самой тяжелой, заключительной стадии работы.

Джон Обст из Ассоциации американской печати помог мне разобраться в их подходе к использованию схем и других средств XML для обмена новостями.

За последние несколько лет я обучил XML и разработке приложений электронной коммерции тысячи людей. Многочисленные лекции и семинары позволили мне протестировать новый материал и работоспособность продуктов. Ответы моих учеников дали мне массу нового материала для книги, включая кучу свежих анекдотов.

И, наконец, я благодарю вас за то, что вы все-таки купили эту книгу. Я надеюсь, что, читая ее, вы получите такое же удовольствие, какое я получил от работы над ней.

Введение

XML, или extensible Markup Language, — это простая технология. ввергнувшая компьютерный мир в водоворот изменений- Еще до формального одобрения XML глава корпорации Microsoft Билл Гейтс назвал новый язык «технологическим прорывом», поскольку видел грандиозные перспективы описания структуры данных на независимом переносимом языке. С тех пор на основе XML создано множество стандартов обмена данными, каждый для своей области, но все на базе этого простого синтаксиса.

BizTalk и упрошенный протокол доступа к объектам (Simple Object Access Protocol, SOAP) — открытые инициативы на базе XML, цель которых — упростить взаимодействие Web-систем электронной коммерции. Обе инициативы используют XML как язык обмена данными благодаря его удивительной способности точно и непротиворечиво описывать структуру данных.

Я давно хотел написать книгу об XML. Моя книга «Руководство по использованию SGML» (*The SGML Implementation Guide, Springer Verlag, 1995} в* середине 90-х пригодилась тем, кто занимался внедрением SGML-решений. Когда на горизонте появился XML, мне в голову пришла идея немного переписать мою первую книгу, сделав ее «XML-совместимой». Это оказалось непросто. Гибкость, простота и гораздо меньший объем XML позволяют делать то, что и не снились пользователям SGML, поэтому новая книга должна была быть не просто XML-совместимой, но полностью посвящена этому языку.

Я человек занятый, поэтому так и не собрался написать хотя бы небольшую книжку об XML. В прошлом году полки книжных магазинов просто ломились от книг по самым разным аспектам XML. А мне, естественно, не хотелось, чтобы моя книга затерялась среди них.

Потом на свет появились замечательные продукты на базе XML — BizTalk и SOAP, и я понял, что пробил мой час. У BizTalk и SOAP много общего, но хватает и различий. В этой книге я надеюсь рассказать вам о них и проиллюстрировать свой рассказ примерами из жизни.

BizTalk

Microsoft запустила проект BizTalk для решения проблемы обмена документами и данными между партнерами по электронной коммерции. BizTalk предлагает изящный метод передачи компьютерных данных через Интернет в виртуальном конверте, действующий так же, как обычная почта, пересылающая обычные письма в бумажных конвертах.

На самом деле у BizTalk четыре ипостаси. Первая — независимая спецификация документов инфраструктуры BizTalk (BizTalk Framework Independent Document Specification). Инфраструктура BizTalk — это набор тэгов, позволяющий описывать адресацию сообщений и их обработку. Фактически инфраструктура BizTalk образует тот «конверт», в котором мы пересылаем документы и данные нашим партнерам.

Вторая часть проекта — сайт BizTalk.org, предоставляющий пользователю возможность лучше изучить XML и современные технологии электронной коммерции вообще и BizTalk в частности.

В-третьих, для работы с документами BizTalk нужен BizTalk-совместимый сервер (BFC-сервер). BFC-сервер — это программа, которая умеет «читать» документы BizTalk и «обладает интеллектом», необходимым для их обработки. В зависимости от того, какие системы имеются в вашем распоряжении, и от того, что нужно сделать с документом, BFC-сервер должен проявлять разную степень «сообразительности».

И, наконец, есть еще Microsoft BizTalk Server 2000 — BizTalk-совместимый сервер для Microsoft Windows 2000. Так как спецификация инфраструктуры BizTalk открыта, вы можете создавать и отправлять сообщения BizTalk через BFC-сервер, написанный на Java и работающий под управлением Linux, а обрабатывать их на компьютере под управлением Microsoft Windows 2000 Advanced Server, используя BizTalk Server 2000.

Многим BizTalk представляется как разработанный Microsoft протокол только для платформ Microsoft. На самом деле это не так. Фактически первые три составные части BizTafk не принадлежат компании Microsoft. Два Java-приложения могут обмениваться данными посредством BizTalk. Компании, использующие рабочие станции Unix, могут пользоваться XML-схемами, опубликованными на сайте BizTalk.org. Кроме того, BizTalk-сервер можно написать на Perl или другом языке программирования со встроенными средствами для работы с Интернетом.

В этой книге мы напишем сервер BizTalk на OmniMark — языке программирования со встроенными средствами для поддержки XML и для работы с Интернет.

SOAP

SOAP начал свою жизнь как менее требовательный способ активизации DCOM-методов, чем обычные COM-вызовы. По мере развития он превратился в открытую спецификацию, позволяющую вызывать

методы в распределенных системах, связанных разными протоколами Интернета (HTTP, FTP и т. д.). SOAP и BizTalk сходны в подходе к обмену информацией, но оптимизированы для транзакций разного типа.

SOAP лежит в основе другой инициативы Microsoft — Web-служб (Web Services). Web-службы — это принципиально новый взгляд на Интернет. Распространение обозревателей заставило авторов представлять информацию в виде HTML-страниц, чтобы сделать ее доступной потребителям. Если же избавиться от присущих HTML команд форматирования, оставив в силе только требования к структурированию данных, можно построить новый мир взаимодействующих сетевых служб, обменивающихся данными и доставляющих их пользователям. Простейший пример — онлайновые метеорологические службы. Доступ к их услугам не помешал бы сайтам, ориентирующимся на путешественников, однако им нужны не HTML-страницы с прогнозом, а доступ к метеоданным в реальном времени. Как сформатировать такой поток прогнозов, каждый сайт решит сам. Единственный язык, способный решать такие задачи обмена данными, — XML.

SOAP так же независим и переносим, как и BizTalk. Вы можете отправить сообщение SOAP Perl-сценарию на машине под управлением Solaris, который вызовет метод Java-класса для извлечения информации из БД Oracle. Ответ поступит в виде документа XML, с которым вы сможете работать в своем приложении. Вам не нужно заботиться о том, где и как выполняется метод объекта, который вы вызываете.

Чтобы понять, как работают BizTalk и SOAP, нужно сначала разобраться в XML-синтаксисе, который делает их работу возможной, Фактически BizTalk — это надстройка нал SOAP, использующая SOAP для пересылки бизнес-документов по протоколу HTTP.

Первая часть этой книги — введение в XML с технической и деловой точки зрения, цель которого — рассказать читателям о перспективах обмена деловой информации с использованием XML. Фактически первая часть — это та книга про XML, которую я всегда хотел написать. Даже если вас совершенно не трогает SOAP или BizTalk, надеюсь, что вы все-таки найдете что-нибудь интересное в первой части книги.

Вторая часть книги посвящена инфраструктуре BizTalk и использованию XML и BizTalk для создания приложений электронной коммерции. В третьей части мы с вами займемся делом, создав свой BFC-сервер с помощью инструментария Microsoft и других компаний.

Компакт-дисккниги

На прилагаемом к этой книге компакт-диске вы найдете несколько приложений, с которыми, по-моему, вам стоит познакомиться. Кстати, диск снабжен простой программой, облегчающей процесс установки приложений, полной электронной версии этой книги и моей книги *OmniMark At Work, Volume 1: Getting Started* в формате PDF.

Требования к системе

Для работы с приложениями и примерами, хранящимися на диске, вам понадобится операционная система Microsoft Windows 98, Windows NT 4.0 или Windows 2000.

Чтобы поэкспериментировать с проектом Microsoft Visual Basic poGen.vbp, требуются Microsoft Visual Basic 6 или Microsoft Visual Studio 6 (профессиональная или корпоративная редакция).

Для работы с программами на OmniMark понадобятся OmniMark C/VM 6.0 и интегрированная среда разработки OmniMark IDE 2 (необязательно).

Чтобы загрузить примеры, просто вставьте компакт-диск в дисковод и запустите программу StartCD.exe,

Электронные версии этой книги и книги «OmniMark At Work» находятся в папке eBook.

Примеры приложений

По умолчанию все файлы с примерами приложений устанавливаются в папку C:\TravisBook\ChXX\, где XX — номер соответствующей главы. Некоторые программы на OmniMark выполняются как CGI-программы в среде Web-сервера I1S, поэтому сначала нужно связать исполняемые файлы OmniMark с соответствующими расширениями. Инструкции см. в разделе «Настройка Web-сервера для использования OmniMark» приложения А.

Программное обеспечение

В этом разделе описано программное обеспечение, включенное в состав компакт-диска.

Пробная 120-дневная версия Microsoft BizTalk Server 2000

В состав компакт-диска включена пробная версия Microsoft BizTalk Server 2000.

Вы можете установить пробную версию в клиентском или в серверном варианте. Клиентская версия содержит множество инструментов для пользователей и разработчиков, но не поддерживает вы-

полнение транзакций BizTalk. Серверная версия позволяет обмениваться сообщениями BizTalk с другими серверам BizTalk по нескольким транспортным протоколам.

Для работы клиентской версии необходима одна из следующих операционных систем:

- Microsoft Windows 98;
- Microsoft Windows NT Workstation 4.0;
- Microsoft Windows NT Server 4.0;
- Microsoft Windows 2000 Professional с установленной файловой системой NTFS и компонентами доступа к данным Microsoft Data Access Components (MDAC) 2.5;
- Microsoft Windows 2000 Server или Microsoft Windows 2000 Advanced Server с установленной файловой системой NTFS, компонентами доступа к данным Microsoft Data Access Components (MDAC) 2.5 и поддержкой DCOM 1.3.

Для работы серверной версии требуются:

- ОС Microsoft Windows 2000 Server или Microsoft Windows 2000 Advanced Server с установленной файловой системой NTFS;
- библиотека компонентов доступа к данным Microsoft Data Access Components (MDAC) 2.5;
- поддержка DCOM 1.3.

Кроме того, для работы Microsoft BizTalk Server 2000 необходима СУБД Microsoft SQL Server (версия 7.0 с серверным пакетом 2 или Microsoft SQL Server 2000). Пробную 120-дневную версию Microsoft SQL Server 2000 можно получить с сервера Microsoft: http://www.microsoft.com/sql/productinfo/evaluate.htm. Кстати, такая версия находится на компакт-диске, прилагаемом к книге Д. В. Артемова «Microsoft SQL Server 2000. Новейшие технологии».» — М.: Издательско-торговый дом «Русская Редакция», 2001. — 576 стр.: ил. ISBN 5 — 7502 — 0154

Программа установки пробной версии BizTalk находится в папке BizTalk.

Пробная версия Microsoft Visio 2000

Для работы пробной версии Microsoft BizTalk Server 2000 необходимо установить Microsoft Visio 2000. Программа установки пробной 120-дневной версии Microsoft Visio 2000 находится в папке Visio.

SOAP Toolkit для Visual Studio 6.0

SOAP Toolkit для Visual Studio 6.0 выпушен Microsoft для разработчиков ПО. В этот пакет включены инструменты для создания файлов описания служб на языке SDL, необходимых для доступа к Web-сер-

висам с помощью SOAP, и средства поддержки представительского сервиса удаленных объектов (Remote Object Proxy Engine,ROPE), реализующего клиентскую и серверную поддержку приема и отправки документов SOAP.

Программа установки SOAP Toolkit находится в папке Software\SOAPTK\.

XML-редактор XRay компании Architag

XM L-редактор XRay компании Architag — XML-редактор со средствами проверки правильности документов. XRay проверяет XML-документы по мере ввода, сообщая о сделанных ошибках. XRay, разработанный для обучения слушателей курсов компании Architag (*http://architag.com/university),* — превосходное средство изучения синтаксиса XM L. Он поддерживает правильные и действительные XML-документы, а также два вида синтаксиса схем: DTD и XML Data Reduced (XDR).

Кроме того, XRay поможет в отладке приложений, использующих XML. Часто при этом приходится изучать создаваемые такими приложениями XML-документы, и здесь вам не обойтись без хорошего редактора. Он потому и XRay, что «просвечивает» документы, как рентген.

У редактора XRay есть еще две полезные функции: поддержка преобразований XSLT и предварительный просмотр HTML. Окно XSLT позволяет преобразовать открытый XML-документ, связав его с шаблоном XSLT. Если вы преобразуете документ в формат HTML, результат можно посмотреть в окне предварительного просмотра HTML-документов.

Программа установки XRay находится в папке Software\Xray\.

OmniMark C/VM

Язык программирования OmniMark понравился мне как средство быстрой разработки кросс-платформенных масштабируемых сетевых приложений. Компания OmniMark Technologies предоставила мне последнюю версию свободно распространяемого продукта OmniMark C/VM (компилятор и виртуальная машина). Язык OmniMark подробно описан в приложении А.

OmniMark работает на большинстве платформ. На компакт-диске приведена версия для сред Microsoft Win32; версии для других платформ см. на сайте *http://www.omnimark.com.* Компания OmniMark предлагает разработчикам техническую поддержку, справочную информацию и новые версии ПО. Если вы программируете на OmniMark, подумайте о подписке.

Программа установки OmniMark C/VM 6.0 находится в папке Software\OmniMark\. Поддержку и свежие версии ПО OmniMark см. на сайте *http://www.omnimark.com.*

Интегрированная среда разработки OmniMark

Интегрированная среда разработки OmniMark предназначена для создания и отладки программ на языке OmniMark. Виртуальная машина OmniMark (OmniMark VM) тесно интегрирована со средой разработки, так что в вашем распоряжении все средства интерактивной отладки, включая контрольные точки, просмотр значений переменных, условные выражения и т. д.

На компакт-диске находится версия для среды Win32. Это 10-дневная пробная версия; вам придется обратиться на Web-сайт OmniMark за идентификационным ключом к программе. Если вы планируете разрабатывать программы на OmniMark, обязательно познакомьтесь с интегрированной средой.

Программа установки OmniMark IDE 2.0 находится в папке Software\OmniMark\.

Adobe Acrobat Reader

Компакт-диск содержит ПО Adobe Acrobat Reader, чтобы вы смогли прочесть электронную версию моей книги об OmniMark и еще несколько файлов на диске. Вы можете установить Adobe Acrobat Reader на свой компьютер из каталога \Software\Acrobat\.

Техническая поддержка изданий и продуктов Microsoft Press

Издательство Microsoft **Press** приложило все усилия, чтобы книга и компакт-диск не содержали ошибок. Если вы все же найдете неточность, сообщите нам по адресу *http://mspress.microsoft.com/suppon/.*

Если у Вас появились какие-либо замечания, предложения, идеи или вопросы после прочтения этой книги или просмотра диска и у Вас не доступа к сети Интернет, свяжитесь с нами по адресу:

Microsoft Press
Attn: XML and SOAP Programming for BiZiTaBc Servers Editor
One Microsoft Way
Redmond, WA 98052-6399
MSPINPUPT@MICROSOFT. COM

Часть I

Применение XML в электронной коммерции

По сути *расширяемый язык разметки* (extensible Markup Language, XML) – это просто синтаксис, позволяющий создавать языки разметки *(схемы),* которые описывают данные. На практике, однако, XML — нечто большее. XML предоставляет возможности общения с партнерами, недостижимые при использовании любых других технологий.

Для создания приложений, способных задействовать эти возможности, надо хорошо понимать базовый синтаксис XML и несколько связанных с ним стандартов. В первой части этой книги мы обсудим XML с точки зрения перспектив создания приложений электронной коммерции и связанных с этим стандартов. Мы также поговорим о *правильных* и *действительных* XML-документах и схемах. рассмотрим вопросы, которые нужно учитывать при программировании с применением объектной модели документа XML, и обсудим специфику преобразования документов из одной схемы в другую с использованием языка преобразования (XSLT) *расширяемого языка стилей* (Extensible Stylesheet Language, XSL).

Глава 1

Перспективы применения XML в бизнесе

Объемы электронного бизнеса огромны и растут день ото дня. Но что именно понимается под этим термином? Для кого-то это необходимость предоставить данные о своей кредитной карточке на Web-сайте, для других — наличие на нем электронной корзины для покупок. Куда ни глянь, везде электронный бизнес представляется как настоящее и будущее человеческих отношений.

Мне кажется, большинство понимает под электронным бизнесом исключительно электронную торговлю. Торговля, разумеется, подразумевает деловые отношения: люди встречаются, договариваются, принимают решения и т. д. Адам Смит, отец современной экономической науки, как-то сказал, что капитализм — это то, чем люди занимаются, когда государство оставляет их в покое. Коммерция (commerce) — настолько естественное занятие, что «Словарь американского наследия»[1] среди прочих предлагает такие значения:

2. Интеллектуальный обмен или социальные взаимоотношения.
3. Половой акт.

Последнее, правда, затруднительно осуществить через Интернет, но это лишь еще один довод в пользу того, что бизнес — это то, чем занимаются люди, когда имеют дело друг с другом.

Люди занимаются бизнесом тысячи лет. С первых дней цивилизации мы вели дела напрямую, лицом к лицу. Только относительно не-

[1] Толковый словарь английского языка, издаваемый фондом «Американское наследие» (American Heritage Dictionary). — *Прим. ред.*

давно мы начали применять разнообразные технические средства, которые оттеснили непосредственное общение с ведущих позиций в деловых отношениях.

Состоятельные люди давно использовали для связи гонцов, но заключение важных сделок почти всегда требовало личной встречи.

В середине XIX в. в США обычные люди наконец-то смогли позволить себе заниматься бизнесом на расстоянии благодаря появлению разветвленной сети доставки почты «Пони-экспресс». Это радикально изменило практику покупок товаров и значительно удешевило их, поскольку покупатель получил возможность выбора продавца. В духе современных веяний «Пони-экспресс» можно было бы назвать «П-коммерцией».

По мере развития страны железные дороги облегчали доставку товаров на Запад, а обширная почтовая сеть все больше их удешевляла, позволяя продавцам и покупателям находить друг друга (звучит как история изобретения «Всеамериканской паутины»). К тому же времени относится и рождение торговли по почте. «Сирс, Робак и К°» заработали немало денег, принимая заказы и рассылая товары по почте. Необходимость присутствия при заключении сделки отпала, как только люди стали доверять репутации компании.

Потом появился телефон и номер 1-800-Flowers: сняв трубку, можно выбрать букет в подарок почти мгновенно, и никаких бумаг! Это уже операция, проведенная при помощи электроники (телефон!). Разве это не электронный бизнес?

Вспоминается Экклезиаст (1:9): «Что было, то и будет; и что делалось, то и будет делаться, и нет ничего нового под солнцем».

Сегодня я могу зайти на сайт www. *1800flowers.com* (цветы — только начало) и купить что угодно. Нет необходимости разговаривать с живым продавцом и объяснять, что именно меня интересует. Если только вы не прожили последние пять лет в лесной хижине, процедура должна быть вам знакома: выбираете нужную вещь по фотографии на экране компьютера и добавляете ее в свою электронную корзину для товаров. Когда вы закончите выбор покупок, вам будет любезно предложена для заполнения форма с информацией для доставки товара и номером вашей кредитной карточки. Два-три дня — и вы сможете порадоваться обновкам.

Думаю, большинство определило бы этот процесс как сделку в рамках электронного коммерции. Есть ли другие мнения? Если торговля представляет собой процесс взаимодействия продавца и покупателя, является ли сайт, торгующий информацией по подписке, предприятием электронной коммерции? Реальная продукция отсут-

ствует, но такие сайты обеспечивают людей, платящих деньги, чем-то, имеющим для них ценность. Операция происходит каждый раз, когда кто-то заходит на сайт. Относится ли это к электронной коммерции?

А как насчет «бесплатных» сайтов? Я помещаю слово «бесплатные» в кавычки потому, что еще мама говорила, что бесплатных обедов не бывает. (Еще она говорила «не бери деревянных центов»[2]. Понять это было тяжелее всего — я представлял себе заговор фальшивомонетчиков, норовивших всучить деревянные деньги несмышленым детишкам. До сих пор точно не знаю, что она имела в виду.)

Когда кто-то что-то вам дает бесплатно, этот кто-то получает что-то взамен. Возьмите Интернет — сколько бесплатно доступной информации! Но, каждый раз, когда вы просматриваете ее, вашему вниманию предлагаются баннеры, сообщения от спонсоров или просто логотипы сайтов и сообщения от их владельцев. Те же спонсоры и владельцы сайтов считают, что ваш взгляд, «зацепивший» их сообщения, стоит денег, которые они и платят персоналу сайтов за привилегию быть представленными вам. Этот косвенный «налог на взгляд» так же реален для рекламодателя, как и наличные, так что и здесь имеет место бизнес. Считать ли это электронной коммерцией?

Межкорпоративный электронный бизнес

Пока что под словами «электронная коммерция» я подразумевал отношения «поставщик — покупатель» (business-to-consumer, B2C). Этот вид электронного бизнеса особенно привлекает внимание прессы. Когда мы слышим о проблемах Amazon.com, связанных с хранением заказанных в виртуальном пространстве вполне реальных товаров, это проблемы сектора **B2C.** Если речь заходит об отключении вследствие перегрузки серверов eBay в то время, когда люди вернулись с работы, мы тоже имеем дело с огромным неудовлетворенным спросом в секторе B2C.

Но это только часть электронного бизнеса. Есть еще невидимый нам товарооборот между компаниями, производящими продукцию, реализуемую через Web. Возьмем обычную ручку. Когда вы покупаете ее в «онлайновом» магазине, происходит сделка. Для снижения стоимости этой операции магазин может применить соответствующие технологии. Допустим, продавец сэкономил 2 цента на стоимости операции. Эта экономия заслуживает внимания, но это всего лишь пара процентов от стоимости ручки.

[2] Американская идиома «Don't take any wooden nickels» употребляется при прощании в смысле «будь осторожен». — *Прим ред.*

С другой стороны, для изготовления ручки нужны сотни операций. Производитель ручек должен был купить сталь, чернила, пружины и множество других материалов у фирм-изготовителей. Он должен оплатить производство, маркетинг и управление. Каждая из этих операций чего-то стоит. Если бы изготовитель ручек мог экономить пару центов на каждой операции между ним и партнерами, себестоимость продукции снизилась бы отнюдь не на 2%.

В этой книге я хочу сосредоточиться на секторе «бизнес — бизнес» (Business-to-Business, B2B) и на том, как язык XML поможет снизить стоимость бизнес-транзакций. Сейчас этот подход не просто стал актуальным, но уже реализуется в некоторых отраслях. Я хочу обсудить существующие подходы и подробнее остановиться на одном из них, BizTalk, чтобы вы могли видеть, как работают типичные B2B-транзакции.

Разделение данных и процесса

XML позволяет снизить себестоимость и повысить доходность. Для авторов это возможность структурировать содержание документов. Разработчикам больших приложений XML предоставляет синтаксис для обмена данными между объектами. Для тех, кто создает решения электронного бизнеса в сфере, B2B — это возможность обеспечить точность и своевременность выполнения транзакций.

Однако всем категориям специалистов XML предоставляет минимум одно общее преимущество: *он позволяет отделить данные от процессов, оперирующих этими данными*. Подробные иллюстрации этого важнейшего преимущества вы будете встречать по всей книге.

В разделении данных и процессов, манипулирующих ими, нет ничего нового. Номер заказа на покупку, который существует как поле в реляционной базе данных, имеет специальное назначение. Он может быть задействован в разных документах: в заказе, счете-фактуре, отчете и т. д. Ни один проектировщик баз данных никогда не стал бы задумываться о хранении этого номера вместе со списком команд, указывающих, как представить его в том или ином специфическом документе.

Но когда это происходит с другими типами информации, мы часто смешиваем данные и сведения о форматировании. Текстовые процессоры — хороший пример. Рассмотрим заголовок этой главы, набранный шрифтом, который имеет такие характеристики: PragmaticaCondC, 28 пунктов, выравнивание по центру. Человек, читающий строку, отформатированную таким образом, на бумаге или на экране компьютера, может заключить, что перед ним — название главы. Но

чтобы компьютер нашел в тексте название главы, мы должны специально подчеркнуть этот факт — ведь компьютер не человек и не умеет распознавать по образцам. Способность человека иметь дело с неоднозначной информацией — это то, чем компьютеры, увы, пока не обладают. Например, фраза на иностранном языке, набранная в тексте курсивом, для человека подчеркивает факт использования иного языка. Это происходит благодаря нашей способности различать образцы и понимать, с какой целью используется форматирование. В случае же с компьютером для того, чтобы установить отличие фразы на иностранном языке от любой другой, набранной курсивом, мы должны явно отметить каждую такую фразу.

XML предоставляет нам синтаксис, позволяющий однозначно определить каждый информационный объект. Поступая таким образом, вы можете структурировать информацию, а потом обрабатывать ее разными приложениями в зависимости от конкретных требований.

XML и трехуровневая архитектура

В разработке современных Web-сайтов XML используется для разных целей. В этом разделе я хотел бы обсудить Web-сайт, использующий XML везде, где это необходимо, и продемонстрировать превосходство XML над другими технологиями.

Современный Web-сайт обеспечивает посетителей персонализированной информацией. Минули те дни, когда компании хватало нескольких статических HTML-страниц для демонстрации партнерам и потенциальным заказчикам. Компании установили, что первое посещение сайта может быть случайным, но, если при следующем визите заказчик не обнаружит изменений, он больше не вернется, логично предположив, что на этой странице никогда ничего не меняется.

Сегодня преуспевающий сайт предоставляет посетителям полезную информацию, заставляя их возвращаться вновь и вновь. Повторные посещения позволяют снабжать посетителей новыми данными каждый раз. когда этого требуют интересы дела. XML обеспечивает персонализированное представление данных разными способами.

Рассмотрим гипотетический Web-сайт (рис. 1-1), использующий современную трехуровневую архитектуру и XML одновременно для качественного представления информации.

Чтобы получить от сайта максимальную отдачу, мы используем трехуровневую архитектуру. Иногда ее называют n-уровневой, так как между начальным и конечным уровнями может находиться столько промежуточных слоев, сколько требуется для решения задачи. Microsoft иногда применяет термин *распределенная архитектура межсе-*

тевого взаимодействия (Windows Distributed interNet Architecture, Windows DNA).

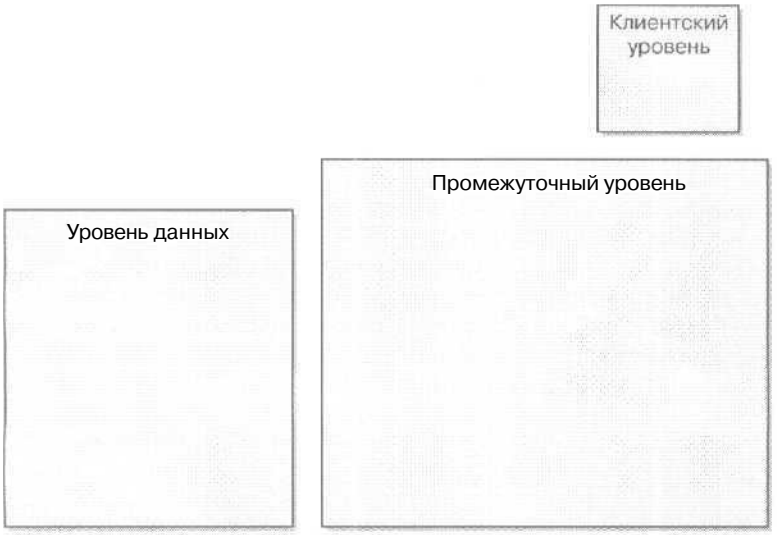


Рис. 1-1. *Трехуровневая* ***Web-архитектура: уровни данных, промежуточный и клиентский.***

Уровень данных

Источник данных находится на уровне данных. Я не сказал «база данных» — ведь существуют и другие варианты. В нашем примере мы используем в этом качестве другие сайты и информацию наших партнеров. Говоря об источнике данных, расположенном на уровне данных, я не обязательно имею в виду его существование на отдельном физическом устройстве. Это просто концептуальное обозначение для места, располагаясь в котором, источник данных доступен для процесса, выполняющегося на другом уровне. Является ли уровень данных отдельным компьютером, компьютерной сетью или виртуальным HTTP-соединением, в данном случае значения не имеет.

Клиентский уровень

Затем следует клиентский уровень. Мы рассмотрим только «тонкий» клиент — обычный обозреватель. «Тонким» он называется потому, что большая часть обработки данных выполняется на сервере, а обозревателю передаются только результаты для отображения. Возможности обработки информации у современных обозревателей больше, чем

обычно требуется. Мы рассмотрим эти дополнительные возможности при изучении использования XML для передачи данных клиенту.

Промежуточный уровень

Располагается между уровнем данных и клиентским. На этом уровне бизнес-процедуры применяются к данным, полученным с уровня данных, а результаты передаются на клиентский уровень. Иначе говоря, данные обрабатываются именно здесь.

Наиболее общим элементом промежуточного слоя является программное обеспечение (ПО), которое отвечает на клиентские запросы об инициализации процессов, — HTTP-сервер или, как его обычно называют, Web-сервер. Web-сервер не выполняет других задач, кроме приема клиентских запросов и запуска приложений промежуточного уровня. Последние могут быть написаны на разных языках программирования и реализуют бизнес-логику, необходимую для достижения определенных результатов.

В нашем примере используется несколько серверных приложений, которые устанавливают соединение с уровнем данных, получают необработанные данные, обрабатывают их и представляют результат конечному пользователю.

Профиль подписчика

Обращаясь с запросом к серверу, обозреватель посылает поток данных, содержащих полезную информацию. Web-сервер может использовать эту информацию для определения типа обозревателя, ОС, установленной на клиентской машине, и возможностей клиента по отображению информации. Некоторые обозреватели могут даже предоставить сведения о том, откуда посетитель попал на наш сайт.

На нашем уровне данных мы создадим базу данных с информацией о профилях подписчиков (рис. 1-2).

Рис. 1-2. ***База данных*** *профилей* ***подписчиков с примерами записей,*** ***проиндексированных*** *по* ***уникальному*** *полю* ***VisitorID.***

Обратите внимание на поля с информацией об имени и адресе электронной почты, а также на поля, содержащие дату и время первого и последнего визитов посетителя на наш сайт.

Мы можем использовать и другой прием: поместить *жетон* (cookie) на компьютер пользователя (если он явно не запретил этого). Так как жетон останется на его компьютере заданное нами время, мы обойдем ограничения протокола HTTP, не позволяющего определить статус соединения. Если жетон был размешен на компьютере посетителя во время предыдущего визита, его обозреватель будет сообщать хранящуюся в файле информацию по запросу сервера. Например, полю VisitorlD (рис. 1-2) соответствует значение, сохраненное нами в жетоне на машине посетителя. Когда сервер проделывает эту операцию лля нового посетителя. в БД, находящуюся на уровне данных, заносится новая запись. При этом поле VisitorlD становится ключом, используя который, мы можем сохранять и выбирать информацию о конкретном подписчике (рис. 1-3),

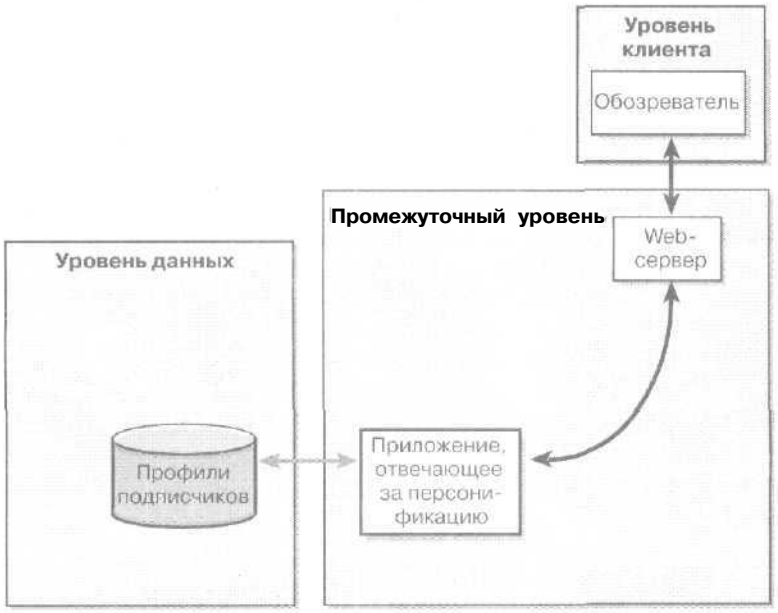


Рис. 1-3. Web-сервер передает информацию, содержащуюся в HTTP-заголовке клиентского запроса, приложению, отвечающему за идентификацию посетителеи. Оно в свою очередь извлекает информацию о посетителе из базы данных. Поле VisitorlD используется для поиска информации о конкретном посетителе.

Таким образом, имея возможность читать файлы-жетоны и находить информацию в БД, мы можем установить время последнего по-

сещения сайта. Это требуется для определения наших дальнейших действий. Заметьте: обработка идет без участия посетителя.

SQL и XML

Структурированный язык запросов (Structured Query Language, SQL), созданный для извлечения информации из реляционных БД, используют сотни тысяч программистов и администраторов БД. Все основные реляционные СУБД поддерживают SQL с некоторыми расширениями, что только подчеркивает мощь стандарта: программист, работавший в среде Oracle, перейдя на Microsoft SQL Server, будет использовать свои знания SQL в новой среде.

Увы. история открытого стандарта на этом и завершается. Хотя SQL позволяет формировать запрос к большинству реляционных СУБД, каждая СУБД определяет свой вариант ответа на этот запрос. Ответом на запрос является *набор записей* (recordset), однако двоичная структура этого набора различна для разных СУБД.

Неплохо было бы иметь возможность посылать стандартный запрос (SQL) и получить ответ на каком-то стандартном языке, правда? (Могу я предложить XML?) Многие производители СУБД работают над решением этой проблемы. Давайте рассмотрим подход Microsoft.

В объектной модели Microsoft ADO (ActiveX Data Objects) результирующий набор записей может быть сохранен в формате XML с фиксированной схемой или набором схем (подробнее о схемах см. главу 4). Допустим, мы выполнили средствами ADO такой SQL-запрос к СУБД:

```
SELECT Name, Joke FROM Students
```

Следующий документ представляет собой данные в формате XML, полученные в результате выполнения SQL-запроса с помощью *компонента доступа к данным Microsoft* (Microsoft Data Access Components. MDAC).

```
<xml xmlns:s='uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882'
   xmlns:dt='uuid:C2F41010-65B3-11d1-A29F-00AA00C14882'
   xmlns:rs='urn:schemas-microsoft-com:rowset'
   xmlns:z='#RowsetSchema'>

<s:Schema id='RowsetSchema'>
   <s:ElementType name='row' content='eltOnly'>
      <s:AttributeType name='Name'
         rs:number='1' rs:nullable='true' rs:write='true'>
```

```
            <s:datatype dt:type='string' rs:dbtype='str'
               dt:maxLengtn='50'/>
         </s:AttributeType>
         <s:AttributeType name='Joke'
            rs:number='2' rs:nullable='true' rs:write='true'>
            <s:datatype dt:type='string' rs:dbtype='str'
               dt:maxLengtn='2147483647' rs:long='true'/>
         </s:AttributeType>
         <s:extends type='rs:rowbase'/>
      </s:ElementType>
   </s:Schema>
   <rs:data>
      <z:row Name='Брайан Трэвис' Joke='Два арахиса ползли по дороге.
      Одному крепко насолил автомобиль.'/>
      <z:row Name='Бен Кинесси' Joke='Человек пришел к врачу и говорит.
      &#x22;Доктор, я везде вижу пятна.&#x22;.
      Доктор отвечает,&#x22;А врача вы видели?&#x22;
      Человек отвечает,&#x22;Нет,только пятна.&#x22;'/>
   </rs:Gata>
</xml>
```

Заметьте, документ содержит два раздела: один описывает схему для возвращенных полей, второй содержит сами записи. Схема документа фиксирована схемой БД, но ее легко преобразовать в любую другую при помощи *языка преобразования* XSLT (см. главу 6).

HTML как XML

Зная время последнего посещения пользователем нашего сайта, мы можем найти нужные сведения, сохраненные в другой БД: статические страницы HTML, содержащие пресс-релизы, рекламу новой продукции или другие данные. Учитывая дату последнего посещения, мы можем предложить посетителю только новые или изменившиеся после его последнего визита материалы.

Мы запрашиваем БД для получения информации, помешенной после последнего посещения пользователем нашего сайта. В такой ситуации XML можно задействовать как стандартный синтаксис для форматирования данных, полученных из БД. Взгляните на работу маркетингового приложения, извлекающего информацию с уровня данных и возвращающего ее Web-серверу (рис. 1-4).

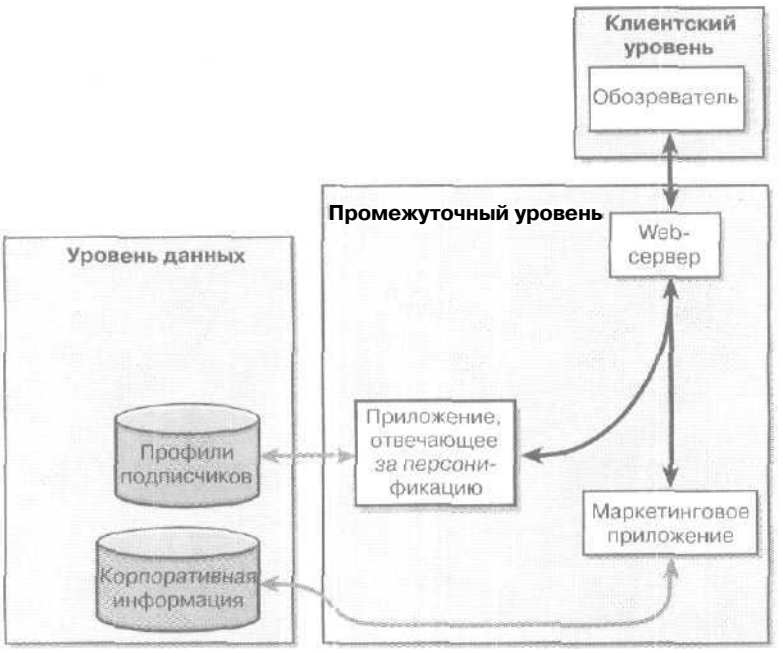


Рис. 1-4. Маркетинговое приложение обращается к информации о дате последнего посещения сайта пользователем.

Как вы убедитесь в дальнейшем, HTML — это почти XML. HTML-документ может считаться XML-документом, если он отвечает требованиям правильности в смысле XML.

Службы электронной коммерции

Главная цель нашего сайта — поддержка бизнеса, поэтому мы будем использовать приложение, работающее с каталогом продукции и услуг, предлагаемых нашей компанией. Этот каталог представляет собой базу данных (рис. 1-5). Наш онлайновый каталог использует информацию из профиля подписчика для формирования списка товаров и услуг, предлагаемых конкретному посетителю.

Рис. 1-5. Добавление дополнительной базы данных на уровень данных. Онлайновый каталог использует ту же технику преобразования данных SQL-XML, что и БД профилей подписчиков.

Информационное наполнение

Теперь мы должны предоставить посетителю нужные ему сведения. Если мы не позаботились об их персонализации, посетитель может больше не вернуться на наш сайт.

Другие сайты могут обеспечить нас богатыми информационными ресурсами. Мы можем задействовать XML для запроса информации и получения ответов, чтобы потом обработать полученные данные. Некоторые службы, например метеорологическая или служба информации о состоянии дорожного движения, предоставляют данные за плату. Они обеспечивают нас сведениями, которые мы можем преобразовать в удобный для посетителя вид. Наша задача, таким образом, — собрать нужную информацию на нашем сайте и предоставить ее посетителю. Вот как используются внешние источники информации в трехуровневой архитектуре (рис. 1-6).

Рис. 1-6. ***Внешние источники становятся частью нашего уровня данных, а агрегирующее приложение объединяет данные на промежуточном уровне.***

Допустим, я хочу добавить на свой сайт сведения о погоде в режиме реального времени. Текущее состояние и прогнозы меняются раз в час. Я могу собрать такую информацию самостоятельно, но установка метеостанций по всей стране выходит за рамки моих финансовых возможностей и бизнес-плана.

Я могу получать бесплатные данные из Национального центра атмосферных и океанских исследований (National Oceanic and Atmospheric Administration, NOAA), составной частью которого является Национальная служба погоды (National Weather Service). Эта организация собирает информацию с множества метеостанций, расположенных по всей стране, и предоставляет ее заинтересованным лицам, но в необработанном виде, поэтому для создания прогноза данные нуждаются в дополнительной обработке. Этот род деятельности также не входит в мой бизнес-план.

И все же есть организация, специализирующаяся на сборе, прогнозировании и предоставлении всем желающим метеосводок, — The

Weather Channel. Помню, как я впервые услышал о 24-часовом телеканале, вещающем только о погоде. Это было в начале 80-х, когда кабельное телевидение только начало входить в дома. Затея показалась мне тогда дурацкой: канал, сутки напролет вещающий о погоде!

Сегодня я беру свои слова обратно. Закладка weather.com установлена в моем обозревателе, я синхронизирую данные на моем карманном PC. используя службу AvantGo. и частенько включаю Weather Channel (362 канал на DirectTV), устраивая вечеринку, уезжая за город или просто устав от остальных 499 каналов, в очередной раз показывающих «Пианино».

Weather Channel, в частности, предлагает информацию, поставляемую weather.com. Этот сайт обеспечивает прогнозами погоды 25 000 пунктов на территории Соединенных Штатов. На weather.com (рис. 1-7) я могу получить прогноз погоды для моего города, указав свой почтовый индекс.

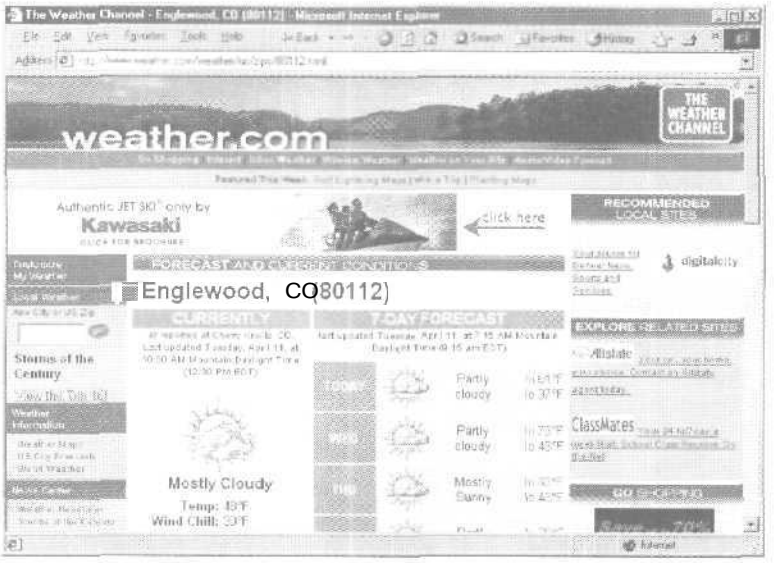


Рис. 1-7. *Компания* ***Weather Channel предоставляет*** *текущую информацию о погоде* ***и многодневный прогноз на сайте weather.com.***

Я хотел бы предложить эти данные посетителям моего сайта. Но я не хочу использовать ссылку па weather.com, поскольку это приведет к тому, что посетитель покинет мой сайт. Кроме того, мне не очень нравится оформление этого сайта — картинки с облаками и солнцем занимают слишком много места. Еще я хотел бы иногда предлагать 3-

или 7-дневный прогноз погоды. И, наконец, мне нужно показывать температуру в градусах Цельсия для тех, кто предпочитает эту шкалу.

Таким образом, вместо использования информации в виде, предлагаемом weather.com, я решаю купить данные в формате, который позволит мне предложить посетителям моего сайта дополнительные услуги. Я иду в офис Weather Channel, стучу в дверь и интересуюсь у персонала, не хотят ли они сотрудничать со мной?

Я формулирую свои требования так: каждый раз, когда мне потребуется информация о погоде, я посылаю им XML-запрос:

```
<weather_request days="5" temp="Фаренгейт" wind="мили в час">
   <zipcode>80112</zipcode>
</weather_request>
```

В ответ я хотел бы получить следующие данные:

```
<weather_response zipcode='80012' Updated='2000-01-07 21:49:06'>
   <location>Аврора, шт. Колорадо</location>
      <current>
      <condition name='temp' >28</condition>
      <condition name='wind chill'>12</condition>
      <condition name='wind' >юго-восточный, 8 миль в час</condition>
      <condition name='dewpoint'>18</condition>
      <condition name=' relative humidity'>66%</condition>
      <condition name='visibility'>25 миль</condition>
      <conditionname='barometer'>30.02flra/iMa</condition>
      <condition name='Sunrise '>7:20</condition>
      <conditionname='Sunset '>16:50</condition>
      </current>
      <forecast>
      <day date='20000108 21:49:06'
      High='49' Low='20' Sky='облачно'/>
      <day date='20000109 21:49:06'
      High='46' Low='21' Sky='облачно'/>
      <day date='20000110 21:49:06'
      High='51' Low='25' Sky='облачно'/>
      <day date='2000-01-11 21:49:06'
      High='51' Low='19' Sky='облачно'/>
      <day date='2000-01-12 21:49:06'
      High='53' Low='21' Sky='облачно'/>
      </forecast>
</weather_response>
```

Прежде чем мы вернемся к моей истории, посмотрим внимательно на приведенный код. Эта двухсторонняя транзакция содержит запрос и ответ на запрос. Архитектура «запрос — ответ» — основа безошибочных транзакций. А что произойдет в случае ошибки?

Допустим, я отправил такой запрос:

```
<weather_request days="5" temp="Фаренгейт" wind="мили в час">
   <zipcode>80000</zipcode>
</weather_request>
```

Код ошибочен, и служба погоды не сможет обработать запрос. Она может ответить мне, скажем, так:

```
<weather_response zipcode="80000"/>
```

Это просто подтверждение, что я ввел почтовый индекс. Получив подобный ответ, я должен как-то сообщить посетителю, что он ввел неверный индекс, и предложить ввести его заново.

Теперь, допустим, я послал совсем ошибочный запрос:

```
<weather_request days="365" temperature="F">
   <zipcode>полная белиберда</zipcode>
</weather_request>
```

В этом случае принимающая сторона вернет ответ:

```
<weather_response/>
```

На переговорах с Weather Channel мне приходится иметь дело с передачей данных и другими потенциальными проблемами. Это важнейшая часть B2B-транзакции. Помните Экклезиаста: «Нет ничего нового под солнцем»? Даже в дни «Пони-экспресс» деловые отношения требовали предусматривать решения любых потенциальных проблем. Правильно составленное соглашение — единственный путь к решению.

В конце концов персонал Weather Channel соглашается, что предоставление прогнозов погоды — дело реальное, но разработка и поддержка этого стоит денег, поэтому они хотят, чтобы я оплатил эти данные.

«Нет проблем! — говорю я. — Я буду платить вам полцента за каждый прогноз, помесячно». «По рукам», — говорят они. Weather Channel разрабатывает систему, предоставляющую мне данные, а я оплачиваю их на основании соглашения.

Итак, что же я буду делать с информацией, когда получу ее? XML-документы, присылаемые Weather Channel, содержат только нужные

мне сведения. Я могу применять различные методы обработки этих данных в зависимости от требуемых результатов: преобразовывать прогнозы в вид, в котором они публикуются на weather.com, или выводить текущие погодные условия в отдельном окне и установить ссылку на страницу, на которой находятся прогнозы на пять дней вперед и т. д. Иначе говоря, я могу обрабатывать и выводить информацию в том виде, в каком считаю нужным потому, что имею доступ к необработанным данным.

Простой протокол доступа к объектам

Как данные из внешних источников попадают на наш сайт? Как только наш партнер — владелец внешнего источника данных — сделает свои данные доступными, я должен быть готов к установлению соединений. Данные могут быть доставлены по протоколу FTP, защищенному протоколу и даже по электронной почте. Однако снова и снова протокол HTTP оказывается наиболее предпочтительным. Дело в том, что HTTP — самый простой для обработки и, что самое важное, не отфильтровывается почти никакими типами брандмауэров. Эти факторы значительно облегчают построение систем, работающих в слабосвязанной среде и использующих данные как связующий уровень.

Как запросить данные и получить ответ? При помощи *простого протокола доступа к объектам* (Simple Object Access Protocol, SOAP). SOAP — проект стандарта, предложенный комитетом по инженерным вопросам Интернета (Internet Engineering Task Force, IETF), — позволяет вызывать процедуры на удаленных системах, применяя стандартный набор тэгов XML. SOAP использует HTTP как транспорт, т. е. как протокол обмена данными между вашим приложением и внешними источниками информации. Подробнее об этом мы поговорим в главе 8.

XML для поставщиков информации

Ряд сайтов предлагает информацию на синдицированной основе. Примеры синдицированных партнеров — газеты, традиционные синдикаты типа King Features или UPI и даже ловкие владельцы Web-сайтов, перепродающие данные, собранные в онлайне.

Я потратил немало времени на переговоры с Weather Channel. Но Weather Channel — не единственный поставщик прогнозов погоды, Этим же занимаются Accuweather, Национальный центр атмосферных и океанских исследований и легионы продавцов их информации. Подобные поставщики информации есть и в других странах. А если бы меня не устроило качество данных от Weather Channel или они не смогли бы предложить мне необходимый набор услуг? Или, допустим, я захотел бы разместить у себя на сайте прогноз погоды по дру-

гим странам? Мне пришлось бы вести переговоры с другим поставщиком нужной информации.

Мой сайт — не единственный, покупающий сведения о погоде для привлечения посетителей. Многие порталы предлагают подобную услугу в обмен на некоторую маркетинговую информацию о посетителе. Если следовать нашему сценарию, представитель каждого портала должен посетить поставщика прогнозов и договориться о формате поставляемых сведений.

Согласитесь, было бы неплохо собрать поставщиков и покупателей информации о погоде и сообща придумать единый формат обмена данными. В информационном бизнесе подобное уже произошло. Давайте рассмотрим возникшие проблемы и меры и решения (рис. 1-8).

Рис.** 1-8. Большое число **партнеров обеспечивают нас информацией,** используя **единые стандарты, что позволяет объединить различные источники данных.

Компании, предлагающие новости, например члены Ассоциации американских газет (Newspaper Association of America, NAA), распо-

лагали информацией для продажи, но им помешали различия в форматах ее представления. Так, издательство «New York Times» использует собственную систему редакционной подготовки и набора для создания своей знаменитой продукции. По сути, существует даже несколько несовместимых систем, каждая для конкретного продукта: одна для ежедневного издания, другая для воскресного и т. д.

Если я покупаю информацию у «New York Times», я должен быть готов либо принимать файлы в формате, используемом ими, либо убедить их представителей преобразовывать их данные в формат, совместимый с моей системой Я могу получить от «New York Times» информацию о событиях в стране и в мире, но ведь моим посетителям могут понадобиться и местные новости.

Если бы все мои посетители были из Нью-Йорка, «New York Times» было бы более, чем достаточно. Но. чтобы удовлетворить всех посетителей, мне может понадобиться информация разных газет из разных уголков мира.

Будучи владельцем небольшого сайта, я могу потратить массу времени, убеждая могучую «New York Times» конвертировать свои данные в удобный мне формат. К тому же я должен найти подход к каждой из сотен газет из разных уголков страны и попытаться убедить их предоставлять мне данные в удобном для меня формате. Вряд ли я буду этим заниматься. Если уж я захочу покупать новости во всех этих местах, проще было бы написать соответствующие конверторы, преобразующие данные к нужному мне виду.

Поскольку каждый из моих поставщиков реализует процесс подготовки издания по-своему, для каждого мне придется изучить формат данных и написать соответствующий конвертор. Этот длительный процесс чреват к тому же массой ошибок при синхронизации данных. На рис. 1-9 показаны традиционные поставщики информации (New York Times и Denver Post), представителя «новой волны» — компания MSNBC — и поставщик информации в формате ежемесячных журналов (Reason).

На самом деле проблем будет гораздо больше. Допустим, другие владельцы сайтов тоже хотели бы приобрести те же данные, что и я, Им также придется писать собственные конверторы, поскольку в данном случае мы являемся конкурентами и я вовсе не собираюсь делиться с ними своим ПО. Учитывая все это, реальная картина может выглядеть так (рис. 1-10):

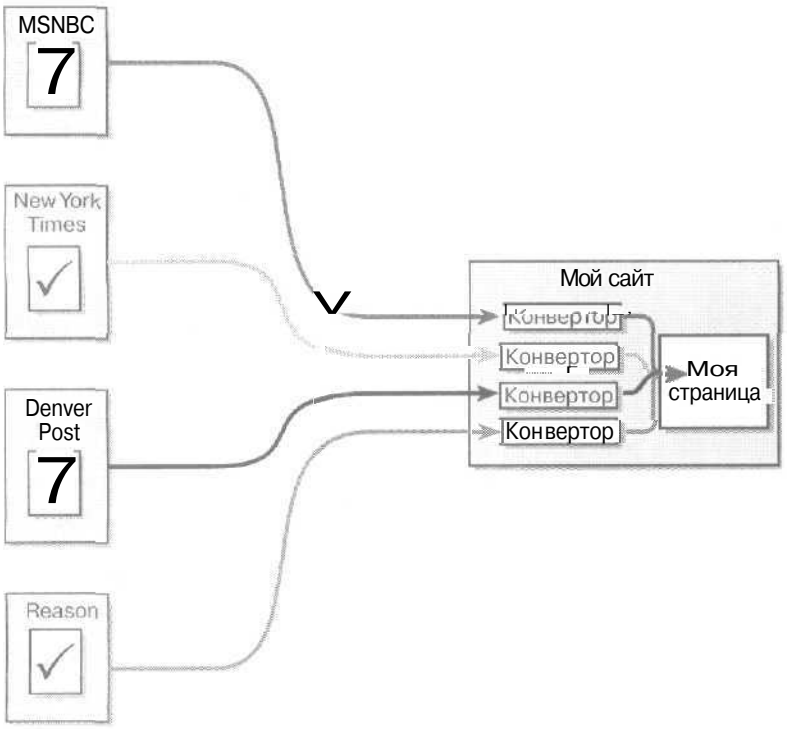


Рис. 1-9. Информация от каждого партнера должна быть преобразована в форму, которую можно использовать на сайте.

И чем больше сайтов-поставщиков и потребителей информации, тем хуже. Но это еще не самое плохое. Каждый источник создает информационное наполнение для своей газеты или Web-сайта. Журналисты иногда предлагают больше информации, чем может быть напечатано в газете. Рассмотрим сообщение;

Фестиваль «Горящий человек» пройдет в июне следующего года. Регистрационный взнос 150 долларов.

Это может заинтересовать читателя, но наш источник новостей располагает большей информацией, чем требуется его целевой аудитории. Например, фестиваль происходит в пустыне Невада каждый год. А для нашей аудитории было бы интересно узнать о будущем шоу все подробности. Безусловно, то, что фестиваль пройдет в июне, важно, но хорошо бы знать точную дату проведения.

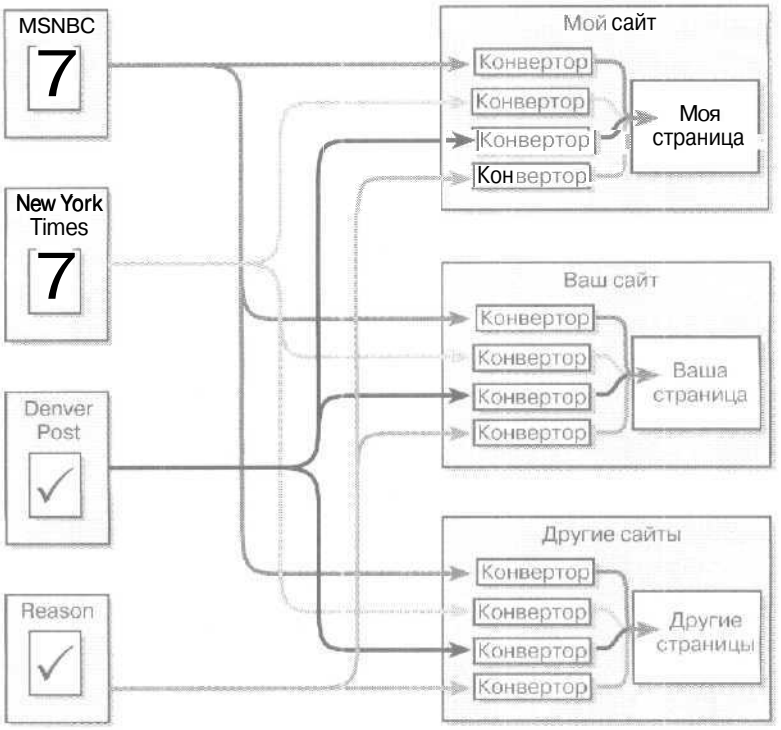


Рис. 1-10. С ростом числа сайтов, использующих информацию, количество проблем для каждого из них также растет.

Исходя из соображений краткости и журналистского стиля, та же информация, помещенная в общенациональной газете, была бы избыточной:

Фестиваль «Горящий человек» в пустыне Невада пройдет с 12 по 19 июня 2000 года. Цена входного билета — 150 долларов США.

Вывод: информация должна существовать в такой форме, которая позволила бы грамотному редактору извлечь из нее именно то, что нужно конкретной аудитории. Иначе говоря, было бы превосходно уметь адаптировать данные для разных аудиторий. Тем самым мы сможем придать нашим новостям большую ценность и получить больше шансов привлечь внимание тех, кто ее купит. Словом, чем разнообразнее потребители нашей информации, тем ценнее она для нас и наших инвесторов.

Стандарты

Когда в середине 90-х годов количество Web-узлов достигло критической массы, традиционные поставщики новостей поняли, что многие владельцы небольших сайтов, подобно мне, готовы покупать информацию, но не могут себе позволить оплачивать ее преобразование. Если бы поставщики смогли предоставлять данные в форме, удобной для дальнейшего использования, это открыло бы новые источники доходов.

Многие газеты состоят в газетной ассоциации. В 70-х годах комитет по кабельной связи (Wire Service Committee) этой ассоциации — тогда она называлась Ассоциацией американских издателей газет — создал спецификацию ANPA 1312 для передачи статей в газеты. Спецификация должна была также гарантировать, что при передаче разметка статей (начертание символов, трекинг, типографская разметка и т. д.) останется неизменной и будет безошибочно воспроизведена в месте получения.

Практически все службы кабельных сетей Северной Америки и поставщики ПО для кабельных сетей приняли спецификацию ANPA 1312 в качестве стандарта, хотя и были сообщения о том, что некоторые сети поддерживают ее не в полном объеме. Вскоре ANPA 1312 получила название «текстовый формат новостей» (News Industry Text Format, N1TF), а Комитет по кабельной связи был преобразован в Оперативную группу по новостной информации (News Information Task Force, NITF).

В 1998 г. текстовый формат новостей был пересмотрен и преобразован в XML-схему. Эта схема очень много добавила к оригинальной спецификации: были определены тэги, которые описывают данные. а не типографскую разметку

Сравните рис. 1-11 с рис. 1-10. Стандартизированный подход к доставке новостей требует от каждого поставщика и получателя информации создания только одного конвертора, поскольку обе стороны используют один и тот же словарь для описания структуры данных. Процесс получения информации от поставщика с использованием стандартизованного формата показан на рис. 1-11.

Каждому поставщику требуется только один конвертор для преобразования данных из собственного формата в NITF. После этого получатели могут использовать этот формат в качестве основного или преобразовывать данные в формат, принятый в их системах.

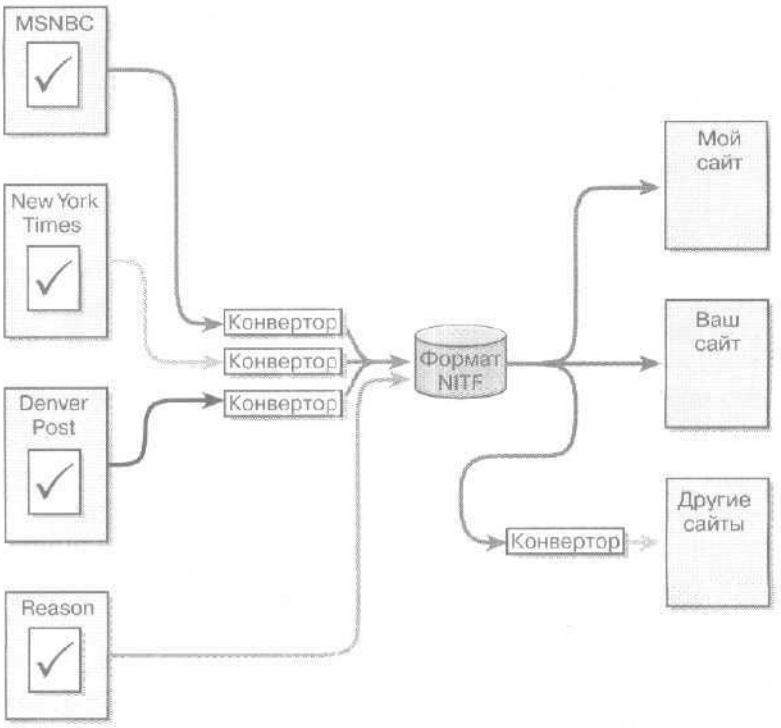


Рис. 1-11. *Единый* ***формат исключает необходимость в преобразовании данных*** *каждого* ***поставщика.***

Но и это еще не все. Рассмотрим пример с «Фестивалем горящего человека». Базовая информация (заголовок сообщения, место и регистрационный взнос) может быть размечена так, чтобы быть более доступной, чем в исходном сообщении. Эта дополнительная разметка дает больше возможностей дизайнеру сайта для генерации информационного наполнения.

```
<event>Фестиваль горящего человека в <location>пустыне Невада</location>
</event>состоится<chron norm="20000612">в июне </chron>.
Регистрационный
взнос <money unit="USD">150</money>
```

Например, щелчок текста внутри тэга <event> выдает сведения о всех событиях в пустыне Невада, а щелчок текста «в июне» — о всех событиях этого месяца. То есть стандартно размеченные данные позволяют дизайнеру сайта предоставлять пользователям прямой дос-

туп к информации. Одни и те же данные можно представить в виде, привычном посетителю из Германии, указав цены в немецких марках, и из Австралии, переведя их в австралийские доллары.

NITF предоставляет богатые возможности для разметки содержания статей и дополнительные возможности, весьма полезные для покупателей, желающих представить информацию иным образом.

Рассмотрим статью от MSNBC (рис. 1-12).

Рис. 1-12. Обычная газетная статья, представленная как HTML-страница.

Эта же информация может быть представлена в NITF-формате при помощи следующей разметки:

```
<?xml version="1.0"?>
<nitf>
   <head>
      <title>Technology Tools and Toys</title>
   </head>
      <body>
         <body.head>
            <headline>
               <hl1>Pac-Man chomps its way toward 20</hl1>
               <hl2>Seminal video game rose from humble
```

```
            beginnings</hl2>
        </headline>
        <byline>
            By<person>Steven Kent</person>
            <bytag>MSNBC CONTRIBUTOR</bytag>
        </byline>
        <dateline>
            <location></location>
            <story.date>1999-09-03</story.date>
        </dateline>
    </body.head>
    <body.content>
        <p>He may have been a fixture in bars for
        most of the last two decades, but Pac-Man,
        one of the video game industry's greatest
        living legends, is still a bit shy of legal
        age.</p>
        ...
    </body.content>
  <body.end></body.end>
</body>
</nitf>
```

Формат NITF позволяет поставщикам обеспечивать сайты-подписчики новостями, а потребителям — легко преобразовывать информацию от поставщиков в нужный **им** вид. Теперь мне не нужно вести переговоры отдельно с Denver Post или Reason Foundation о покупке новостей, которые я смогу прочитать. Я просто спрашиваю: «Вы используете NITF?» Если они отвечают «да» — их информация поступит в мою систему, нет — они потеряют меня как клиента и будут вынуждены изучить этот промышленный стандарт.

Наш сайт рассчитан на индивидуальных подписчиков. Стандарты, подобные NITF, позволят нам создавать специализированное информационное наполнение, привлекательное для каждого посетителя. Мы хотим привлечь покупателей, демонстрируя ему нашу продукцию и услуги. Поэтому наш сайт относится к категории сайтов B2C.

Получение данных от поставщиков для предоставления потребителям требует тщательной координации между сайтами и поставщиками онлайновой информации. Этот аспект, очевидно, относится к компетенции сектора **B2B.**

Создание страницы

Вся информация, которую мы обсуждали ло сих пор, связана приложениями промежуточного уровня и страницами, которые они создают для конкретного посетителя. Эти страницы создаются специально по запросу пользователя и форматируются согласно конфигурации и параметрам каждого пользовательского компьютера. Наш Web-сайт сам создает страницы, доставляет их и ждет дальнейших запросов. И все это благодаря XML!

XML и управление содержанием

Предметом этой книги является применение XML для B2B-транзакций в электронном бизнесе. Поскольку XML изначально задуман для *самодокументированных* транзакций, он идеально подходит для электронной коммерции. Мне нравится определять это как «короткоживущую» форму XML: вы «с нуля» создаете XML-документ (или, точнее, поток) и отправляете его получателю для прочтения, интерпретации и, наконец, удаления.

Технология XML применима и для объектов-долгожителей — таких, например, как традиционные документы. Поставщиками информации являются компании, публикующие данные, потребляемые их клиентами. Издательства вроде Microsoft Press, Prentice Hall или Macmillan также поставляют информацию, однако уже как издатели газет и журналов.

Конкуренция заставляет их изучать новые методы хранения информационных объектов. В прошлом это достигалось путем физической передачи данных потребителям в виде бумажных документов. Эта модель работала столетиями. Большое достоинство бумаги в том, что вы можете добиваться идентичных графических эффектов на странице разными путями. Любой из многих текстовых процессоров, наборных машин, программ верстки — даже свинцовая форма — может быть использован для получения одного и того же оттиска. Раз набранную страницу можно затем напечатать хоть тысячу раз. По мере развития техники появилась возможность скопировать страницу и легко передать ее. Бумага стала универсальным носителем для обмена информацией.

Но бумага, а позже и ее заменители — файлы текстовых процессоров, файлы формата Adobe Acrobat PDF и т. п. — имеют и недостатки. Ведь на самом деле напечатанный документ *не есть* информация. Это лишь одно из ее представлений, а именно в формате для бумажного носителя — ведь бумага много лет была единственным способом распространенияинформации.

С появлением в начале 90-х годов компакт-дисков у поставщиков информации возникли проблемы с бумажным представлением документов: они не хотели просто помещать бумажные документы (или их электронные эквиваленты) на компакт-диск, потому что люди ожидали большего от электронного носителя информации — например, возможности поиска по ключевому слову или строке, или ограничения области поиска определенным контекстом. Издатели юридической литературы обнаружили, что покупатели хотели бы иметь возможность искать определенные работы, но только если они были в научных трудах или только в судебных решениях. Если эти издатели просто помещали электронные эквиваленты своих бумажных публикаций на компакт-диск, система поиска не могла выяснить, относится найденная информация к научному труду, судебному решению или другому тексту.

Извлечение информации с бумажного носителя и ее преобразование в вид, удобный для электронной доставки потребителям вид — процесс сложный, но осуществимый, и многие компании, занимающиеся этим, многократно окупают свои инвестиции.

XML и сбор информации

Интернет повысил нашу требовательность к информации. Никогда раньше мы не хотели столько знать о продукции, чтобы принять решение о покупке.

Рассмотрим торговлю автомобилями в до-Web'овскую эпоху. Чтобы купить машину, вы должны были потратить массу времени, перебираясь из автосалона в автосалон и сравнивания цены, условия обслуживания и прочий набор дополнительных услуг. Все это требовало больших затрат как со стороны покупателя, так и со стороны продавца.

Сейчас я могу зайти на *cars.com*, *carpoint.msn.com* или другой высококлассный сайт и найти все, что мне нужно. Кроме того, я могу получить массу ценных сведений на сайтах производителей авто: от технических характеристик до местонахождения ближайшего дилера. Я могу даже виртуально осмотреть заинтересовавшую меня машину. Интернет создает идеальную среду для покупок, которая избавляет меня от суеты, связанной с покупками в реальном мире. Разумеется, я не могу прокатиться на выбранной машине прямо в Интернете (пока), но информация, которую я получаю, здорово экономит мне время и силы.

«И причем тут XML?» — спросите вы. Рассмотрим сайт, например carpoint.com (рис. 1-13), который собирает информацию из разных

источников и делает ее доступной посетителю. Пройдемся по сценарию покупки и выясним, когда может понадобиться XML.

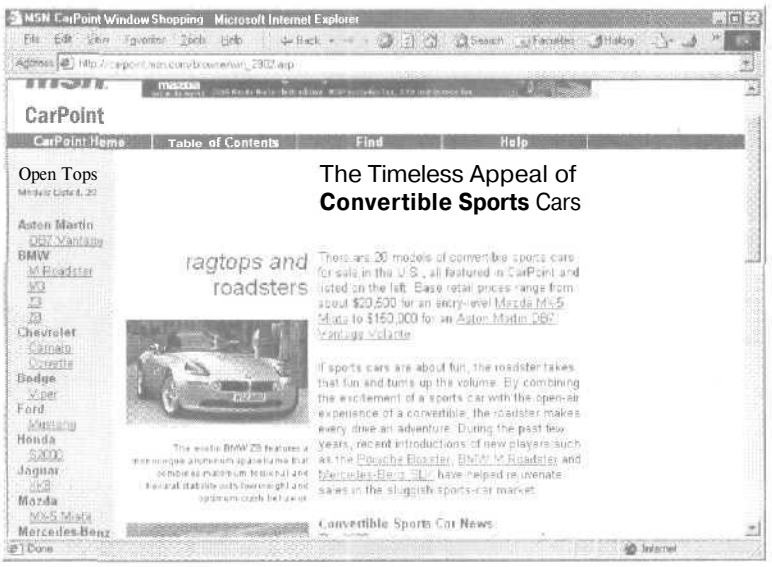


Рис. 1-13. B2C-сайты, такие как MSN CarPoint, обязательно используют компоненты B2B, поскольку для принятия решения о покупке потребителю необходима информация от многих компаний.

Допустим, я хочу приобрести кабриолет и заглянул сюда, чтобы выбрать модель. На сайте MSN CarPoint слева предусмотрено оглавление, а справа — собственно детальная информация.

Итак, я выбрал новый Jaguar XK8 (рис. 1-14). Отличная машина.

Я вижу краткое описание автомобиля и ссылки, указывающие на более подробные сведения (рис. 1-15). Я выберу информацию о моторе. Поскольку я живу в Колорадо, мне нужен достаточно мощный двигатель, способный работать в высокогорных условиях с разреженным воздухом.

Такая конфигурация есть: XKR, 370 лошадиных сил, турбонаддув — то, что нужно. Осталось выбрать дополнительное оборудование и посмотреть, сколько все это будет стоить (рис. 1-16). Пожалуй, я еще куплю навигационную систему, чтобы не заблудиться в дальних поездках за покупками. А сколько это стоит? Хм, остается надеяться, что гонорар заплатят пораньше!

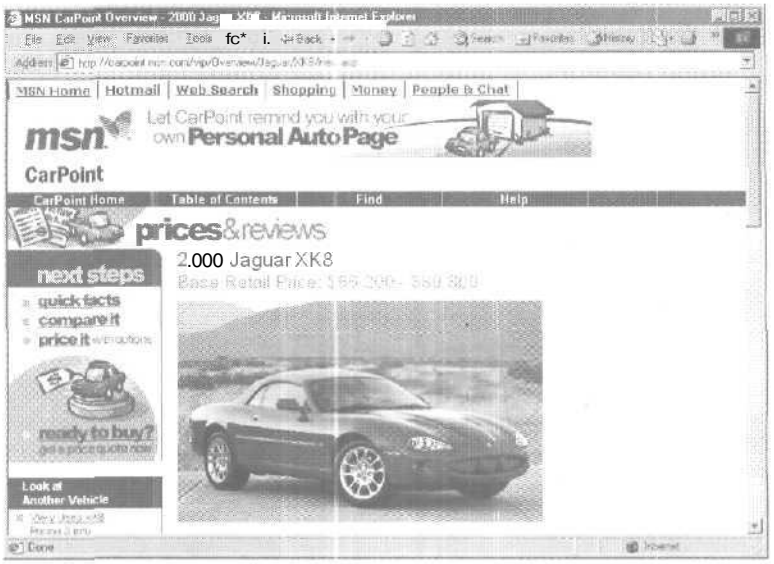


Рис. 1-14. ***Наглядные, информационно-насыщенные*** *сайты содержат разнообразные* ***данные.*** *MSN Car* ***Point собирает их из разных*** *источников.*

Это только один вариант из сотен автомобилей и тысяч конфигураций, доступных покупателю на этом сайте. Как же данные попадают сюда? В основном с сайтов, содержащих необходимую информацию. Данные динамически обновляются в процессе навигации пользователя по сайту.

Допустим, мы хотим создать подобный сайт. Мы, конечно, можем обратиться в «Ягуар», «Хонду» и к другим изготовителям. Это отняло бы у нас массу времени и не исключено, что доходы от нашего сайта не смогли бы окупить наши затраты.

Но мы не единственный сайт, использующий такую информацию. Как и в примете с NITF, существует много поставщиков сведений об автомобилях:

- изготовители автомобилей информируют о технических характеристиках и дополнительном оборудовании;
- специализированные издания публикуют обзоры для своих подписчиков;
- энтузиасты — например, местные и национальные автоклубы, общества коллекционеров автомобилей и т. д. — предлагают информацию о предмете своего увлечения;

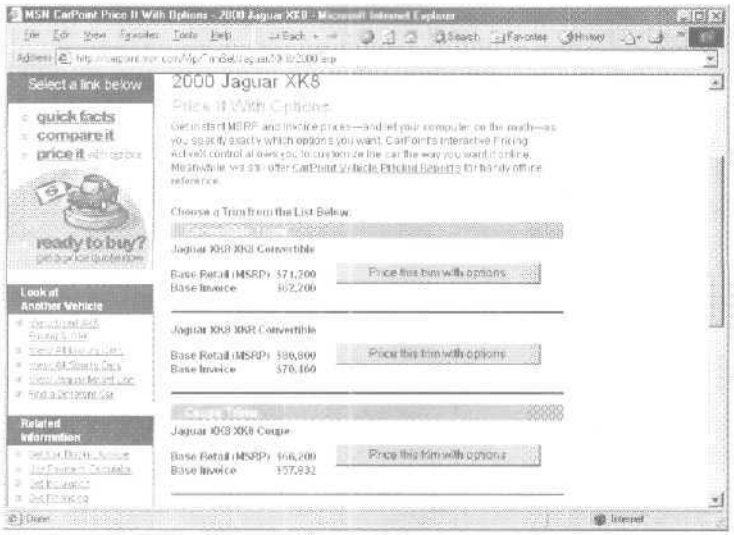


Рис. 1-15. *Доступные конфигурации основываются* ***на данных,*** *полученных* ***от производителей.***

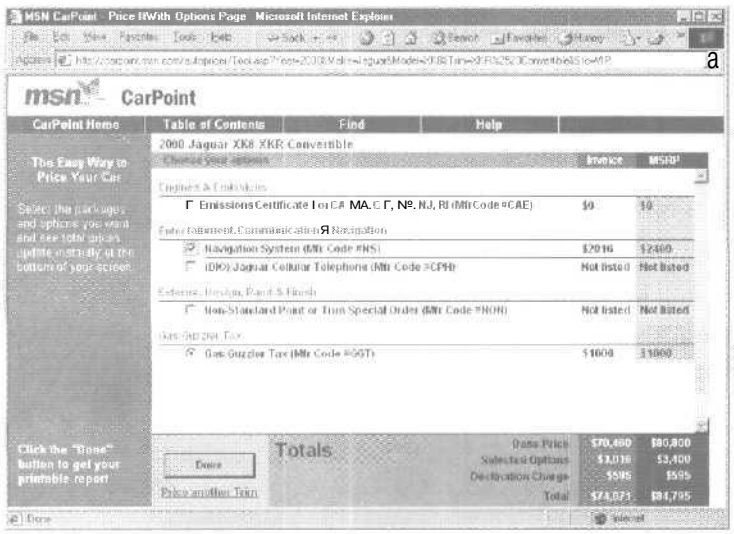


Рис. 1-16. Дополнительное ***оборудование, предназначенное специально*** *для выбранной* ***модели.***

- страховые компании располагают статистическими данными, и вы можете узнать, во что обойдется страховка;
- Институт страхования дорожного движения (Insurance Institute for Highway Safety) и правительственные организации распространяют сведения о результатах тестовых испытаний по безопасности автомобилей:
- дилеры сообщают о своих ценах и услугах.

Чтобы достичь максимальной посещаемости и дохода от рекламы. Web-сайт должен обеспечить потенциальных посетителей всей информацией, которая может быть ими востребована. Такой сайт ориентирован на потребителя, что подразумевает его принадлежность к категории B2C. Однако возможность формирования такого сайта во многом определяется способностью получать данные от поставщиков в удобном для использования формате. С этой точки зрения сайт относится к категории B2B. Эта сфера B2B-транзакций естественна для применения BizTalk.

Электронный документооборот

Бабушкой электронного документооборота была спецификация *электронного обмена данными* (Electronic Data Interchange, **EDI).** В начале 70-х пионерами электронных бизнес-коммуникаций стали такие крупные компании, как Sears и Kmart. На организацию работы тысяч торговых точек и фирм-поставщиков они изводили горы бумаги, что явно снижало производительность. Это и обусловило появление электронных коммуникаций. Для перехода на электронный документооборот компании разработали и внедрили специализированных средства коммуникаций для каждого из своих бизнес-партнеров.

В конце 70-х комитет, составленный из представителей транспортных компаний, правительства и фирм-производителей компьютерной техники, приступил к целевой разработке методов улучшения и стандартизации электронных бизнес-коммуникаций. В 1979 г. Американский национальный институт стандартов (American National Standards Institute, ANSI) принял пакет стандартов X12, ставший первым комплектом стандартов для электронного бизнеса.

Разумеется, весь электронный бизнес существовал исключительно в границах частных сетей. Явление, ставшее потом известным как Интернет, в то время уже разрабатывалось, но было ограничено рамками правительственных и академических учреждений.

Различные комитеты X12 работали над стандартами для отдельных документов — главным образом счетов и заказов. Каждая спецификация должна была обеспечивать требования всех сторон в любых

ситуациях, поэтому стандарты были тяжеловесными и сложными для реализации.

Именно из-за сложности стандартов EDI некоторые группы отделились и занялись созданием стандартов, учитывающих отраслевую специфику. Так, Национальная ассоциация розничной торговли (National Retail Merchants Association) начала разработку набора стандартов сообщений заказа товара в рамках EDI. Но в связи с тем, что эти стандарты были недостаточно четко определены, а сообщения отличались неоднозначностью, предприятия розничной торговли и компании-поставщики ими не воспользовались. Впоследствии Ассоциация присоединилась к стандартам X12.

Этот краткий экскурс в историю демонстрирует двойственную природу стандартов: они должны быть достаточно общими, чтобы быть принятыми, и достаточно конкретными, чтобы быть полезными.

XML и EDI

XML — стандарт, созданный консорциумом W3C (World Wide Web Consortium). W3C представляет собой объединение компаний, занимающихся разработкой и поддержкой основных Интернет-технологий. XML предлагает мощный синтаксис, позволяющий компаниям создавать структуры для эффективного обмена информацией. О том, как выглядит XML, мы поговорим в главе 4.

Проблемы, встававшие перед EDI за последние четверть века (длительное время разработки, взаимоисключающие требования со стороны компаний и сложности реализации довольно общих спецификаций, подходящих каждому) должны послужить уроком разработчикам стандартов обмена с использованием XML. Группа XML/EDI работает над более открытой версией EDI, надеясь, что компании, традиционно использующие EDI, будут продвигаться к применению XML как основного синтаксиса обмена данными. Эта группа не одинока. Многие сегодня работают в направлении создания универсального способа ведения бизнеса с использованием XML как синтаксиса описания данных.

Еще одна точка приложения сил — ebXML-инициатива. Сокращение «eb» здесь, конечно же, от «электронного бизнеса». Над ней работают Организация объединенных наций (подразделение CEFACT) и Организация за продвижение структурированных информационных стандартов (Organization for the Advancement of Structured Information Standards, OASIS) — промышленная группа по XML. Целью ebXML является использование стандартного набора XML-схем дня содействия международной торговле. Задача же ebXML-инициативы — раз-

работка инфраструктуры, позволяющей использовать XML для электронного обмена любыми данными. Список группы участников выглядит как справочник «Кто есть кто в международной торговле»: Ariba, Amazon.com, Bank of America, Boeing, Commerce Net. Dunn and Bradstreet, IBM, Microsoft, Oracle, Sun и множество других компаний со всего мира. Многие спецификации были разработаны отдельными участниками этого списка — например, cXML компании Ariba, CBL от CommerceOne, RosettaNet Framework от RosettaNet, eCo Framework компании Commerce Net и т. д.

Другие направления включают формат обмена финансовой информацией (Financial Information Exchange, FIX), HL7 — для обмена медицинской, ACORD — для обмена страховой информацией о собственности и несчастных случаях, J2008 — для автопромышленности и многие, многие другие. Все эти спецификации применяют XML в качестве базового синтаксиса.

Все эти направления, возможно, будут реализованы в виде комплекта спецификаций — схем (я упоминал о них в начале главы), которые будут описывать некоторые виды документов — например, счета, заказы товаров, истории болезней, сделки с ценными бумагами. Схема является важнейшей составляющей решений для электронной коммерции (подробнее о схемах мы поговорим в главе 4).

XML и электронный бизнес

Так где же именно применяется XML? Всюду. XML — это просто набор правил для описания данных. Это небольшой по объему, стандартизованный компактный синтаксис, и в этом его главное достоинство. Для участия в любой деловой операции вы должны четко понимать своих бизнес-партнеров. XM L предоставляет синтаксис для связи, ясно и точно идентифицируя каждый элемент информации, требующей обязательного завершения транзакции — будь то процедура перечисления средств с кредитной карточки через Интернет, получение HTML-страницы от Web-сервера или загрузка записей в формате MP3.

Еще одна причина интереса электронного бизнеса к XML, как ни странно это звучит, — доверие. Если вы не доверяете кому-либо, вы не будете иметь с ним дело ни лично, на городском рынке, ни анонимно — поверх невидимых государственных границ — через Web. XM L обеспечивает необходимый уровень доверия тем, что информация о сделке передается вместе с ней.

Общий язык

Когда пилот авиакомпании Japan Airlines (JAL) сажает реактивный лайнер в аэропорте «Шарль де Голль» в Париже, он должен иметь

возможность общаться с диспетчером на башне управления полетами на одном языке. Родной язык пилота — японский, а диспетчера — французский. Если они выберут в качестве языка общения японский, ответственность за взаимопонимание ляжет на плечи диспетчера. Правда, тогда ему придется заодно выучить и немецкий, итальянский, английский и все остальные языки, на которых разговаривают пилоты, сажающие свои самолеты в аэропорту.

Если же они предпочтут говорить по-французски, то проблемы перевода будут заботой пилота. При этом ему еще предстоит освоить и немецкий, итальянский, английский и другие языки, родные для диспетчеров аэропортов, где ему предстоит сажать свой самолет.

Любой из этих сценариев дорог, отнимает массу времени и потенциально опасен. Исходя из соображений безопасности, всемирная система управления воздушным движением использует в качестве стандарта для связи между воздушными судами и наземными службами управления английский язык.

Когда пилот JAL связывается с башней, он должен мысленно переводить с японского на английский. Составив фразу на английском, он передает ее диспетчеру по микрофону. Радио доносит английскую фразу до башни, где ее слышит диспетчер. Диспетчер мысленно переводит английскую фразу на родной французский, что дает ему возможность действовать наиболее простым путем. (Разумеется, профессионалы такого класса без труда не только говорят, но и думают на английском, но, я думаю, вы поняли ход моих мыслей.)

ПРИМЕЧАНИЕ Интересно, что тот же пилот, возвращаясь в Японию, по-прежнему разговаривает с диспетчером по-английски, поскольку в воздухе находятся другие самолеты, пилоты которых не владеют японским, но слушают диспетчера.

Точно так же XML-документ обычно создается «на лету» каким-либо приложением. Этот документ содержит всю информацию, необходимую для проведения операции. Этот «свежеиспеченный» документ отправляется нашему бизнес-партнеру, причем синтаксис документа является внешним по отношению к создавшему его приложению. На другой стороне приложение нашего партнера получает XML-поток. Этот поток также является внешним для данного приложения, поэтому оно «переводит» его на свой внутренний язык для дальнейшей обработки. По завершению обработки это уже не XML-документ.

Итак, XMK — это *lingua franca*, или общий язык, электронного бизнеса. «Толковый словарь английского языка» так определяет этот термин:

1. Средство общения между людьми, разговаривающими на различных языках.

2. Смесь итальянского с провансальским, французским, испанским, арабским, греческим и турецким языками, использовавшаяся в качестве разговорного языка на восточном побережье Средиземного моря.

Забудем на время о второй части определения и заменим слова «людьми, разговаривающими на разных языках» на «компьютерами разных конфигураций».

Таким образом, XML — это общий язык электронного бизнеса, Это недостаточно строгое определение: XML не является словарем — этим занимаются схемы, а XML лишь предоставляет синтаксис для описания данных, делающий возможным обмен данными. (Мы подробнее рассмотрим это в главе 3, когда будем обсуждать разработку схем для отраслевых групп.)

XML и языки обработки

XML — это открытый стандарт, не являющийся собственностью какой-либо одной компании. Он принадлежит консорциуму в высшей степени конкурентоспособных компаний, осознавших все преимущества соглашения о простом синтаксисе обмена информацией.

В силу своего открытого статуса XML работает на любых платформах и с любыми языками программирования. По сути XML не исполняет никаких функций — это просто синтаксис, обеспечивающий представление ваших данных. Ключевое достоинство XML — разделение данных и процессов, этими данными оперирующих.

Возможно, вы много слышали о сочетании XML и Java: XML - для представления данных и Java — для их обработки. В сочетании XML и Java действительно много смысла, если учесть их открытую кросс-платформенную природу. Разработчики могут создавать кросс-платформенные приложения на Java и использовать XML для обмена данными между ними.

Но Java не единственный язык программирования, успешно работающий с XML. Для создания и чтения XML-данных подойдут и Microsoft Visual Basic, и Microsoft Visual C++, Perl и даже COBOL. Список языков, работающих с XML, так же велик, как сам перечень языков программирования. Поскольку XML просто определяет правила для представления данных, любой язык программирования может использовать преимущества нейтрального описания данных при помощи XML.

В отличие от HTML, имеющего фиксированный набор тэгов, XML позволяет создавать собственные тэги. Вы можете объявлять все, что вам угодно. Например, определить тэги «Название», «Заголовок», «Номер счета» и т. д., но в отличие от HTML они будут описывать не форматирование данных, а их структуру.

Если вы обмениваетесь данными в виде текстовых файлов, вам приходится создавать собственный синтаксис и собственные правила обработки файлов для извлечения информации. **XML** избавляет вас от такой необходимости. Вы просто создаете синтаксически и семантически корректный XML-документ и при помощи анализатора читаете его и извлекаете из него данные. Синтаксических анализаторов XML — великое множество, включая встроенный в Microsoft Internet Explorer, и они обычно распространяются бесплатно.

Глава 2

Предыстория XML

У XML очень простой синтаксис. Вся спецификация занимает около 30 страниц текста. На своих лекциях мне удается свести описание синтаксиса всего к двум слайдам. Разумеется, это не совсем корректно, однако большинство реальных задач чаще всего сводится к нескольким основным правилам.

Несколько лет назад я обучал своего сына Купера играть в шахматы. Ему было семь лет, и он схватывал все очень быстро. Куперу даже удалось сделать изобретение. В эндшпиле, будучи запертым в угол, он поднял короля над доской и тем самым избежал мата слоном. Безусловно, он понимал, что это выходит за рамки правил, которые в то время уже выучил.

Со временем Купер поймет, что выучить правила не главное. Главное суметь играть по этим правилам так, как настоящий гроссмейстер. Нужно не просто знать, как можно передвинуть фигуру. Важно знать, *зачем* делается этот ход и быть уверенным в том, что это *лучший* ход в данной ситуации.

То же самое и с XML. В этой книге правила синтаксиса XML занимают всего несколько страниц, однако лишь опыт поможет вам в каждой конкретной ситуации решить, воспользоваться ли элементом, атрибутом или чем-то еше. Лично мне на то, чтобы досконально понять, как работает XML, понадобилось несколько лет. За это время я понял, что работает, а что нет. Я постараюсь поделиться с вами своими знаниями, когда мы перейдем к вопросам применения XML, пока же давайте поближе познакомимся с XML, сравнив его с другими технологиями.

HTML: достоинства

Язык гипертекстовой разметки (Hypertext Markup Language, HTML) применяется для создания Web-страниц. Это самый распространенный язык разметки в мире. По некоторым оценкам, существует око-

ло 100 миллиардов HTML-страниц, однако их точное число узнать невозможно, поскольку одна программа на Perl способна создавать тысячи разных страниц в зависимости от параметров запроса. Достаточно сказать, что HTML — бесспорно, одна из технологий, изменивших мир.

HTML достаточно компактен, прост для понимания и поддерживается миллионами устройств. Существует множество книг с прекрасным описанием HTML, подробную информацию можно найти и в сети Интернет. Для создания страниц существуют удобные средства, распространяемые бесплатно или почти бесплатно.

Для форматирования и отображения текста на двумерной странице HTML использует *тэги,* описывающие форматирование. Эти тэги затем интерпретируются обозревателем. Обозреватель компании Microsoft и некоторые другие поддерживают *динамический HTML* (Dynamic HTML, DHTML) — программную модель с развитым набором интерфейсов, которая по сути дела превращает HTML в среду для доставки приложений пользователю. Однако из-за существенных различий между обозревателями эта возможность используется редко. Основное назначение HTML заключается в отображении информации, размещенной на сайте.

Наверное, главный урок HTML заключается в том, что гипертекст реально работает. Любой, кто имеет выход в Интернет, может щелкнуть ссылку в своем обозревателе и тотчас получить страницу с сервера, находящегося за тысячи километров. В последние годы мною говорят об универсальном доступе к информации, а группы по стандартизации предлагают все новые способы получения информации. Однако именно простота создания HTML-страниц и способность их связывания сделала возможным Web, каким мы его знаем. Тем не менее, HTML присущи некоторые проблемы,

HTML: недостатки

Основной недостаток HTML заключается в невозможности выполнения многих функций форматирования документа — например, нельзя заставить текст перетекать из колонки в колонку, нельзя точно управлять выравниванием (без использования сложных табличных структур), точно задать ширину пробелов и т. д., без чего серьезный полиграфический дизайн просто невозможен. Для преодоления некоторых ограничений форматирования были разработаны *иерархические таблицы стилей* (Cascading Style Sheets, CSS), однако это уже расширение HTML, и поэтому разные обозреватели обрабатывают атрибуты стилей по-разному.

Рекомендации HTML немногочисленны и не слишком требовательны, но большинство обозревателей не соблюдает даже эти правила. Например, закрывающий тэг </P> для элемента параграфа <P> не обязателен, чем и пользуются многие создатели страниц, не указывая, где кончается параграф. Привыкнув к этому, они перестают пользоваться и обязательными закрывающими тэгами. Однажды на своей странице я применил такую конструкцию:

```
<P>Дополнительная информация <A HREF="more.html">находится здесь<A>.
```

Тэг <A> определяет элемент, служащий для создания гиперссылки (от слова «anchor» — якорь, в смысле привязки). В обозревателе слова «находится здесь» по умолчанию будут выделены синим цветом и подчеркнуты. Закрывающий тэг, располагающийся сразу после «здесь», должен иметь вид </A> — я просто ошибся. Итак, этот код неверен; по спецификации обозреватель должен сообщить об ошибке и отказаться загружать документ.

Однако этого не происходит — в Microsoft Internet Explorer 4 эта страница выглядела замечательно. Впрочем, после ее публикации на Web-сайте на адрес Web-мастера стали приходить сообщения о том, что вся страница после слов «находится здесь» отображается синим цветом с подчеркиванием и представляет собой одну большую ссылку. Оказывается, пользователи просматривали страницу с помощью Netscape Navigator. Синтаксический анализатор (анализатор считывает документ и определяет его структуру) обозревателя фирмы Microsoft догадался о моем намерении завершить описание ссылки. Думается, что такое на страницах встречалось и раньше, поэтому разработчики обозревателя решили сделать исключение из правил и исправлять подобные ошибки «на лету». Однако разработчики синтаксического анализатора HTML из Netscape подошли к этому более формально, в результате чего и появилась длинная ссылка.

Зачем же обозреватели выполняют и даже исправляют неверный код? Дело в том, что производители хотят, чтобы их обозреватели были совместимы со всем, что находится в Web. Никто из производителей не хочет, чтобы их обозреватель не мог чего-то отобразить, и поэтому заставляют свои продукты обрабатывать даже самый плохой код. Поскольку некорректный код все же выполняется, этим все чаще пользуются недобросовестные программисты. Плохой код рождает еще худший, и конца этому не видно.

В языке HTML определен фиксированный набор тэгов, оптимизированный для представления электронных документов. Для форматирования служат такие директивы, как *p* (paragraph, параграф), *li*

(list, список) и *table* (таблица). Однако в HTML нет и не может быть элементов типа *номер счета, страховое свидетельство* или *кровяное давление*. Если нужны такие элементы, описывающие структуру информации, а не ее представление, HTML не поможет — он просто не предназначен для решения этих задач.

В ранних версиях спецификации HTML элементов было очень мало, поэтому многие программисты использовали тэги не по прямому назначению, а для того, чтобы достичь определенного эффекта в оформлении страниц. Скажем, я сам использовал элемент *dl* (definition list) не для списка определений, а только затем, чтобы сдвинуть левый крав абзаца. Я видел, как тэг *author* применялся не для вывода информации об авторе документа, а только из-за того, что добавляет разрыв строки и выводит содержимое элемента курсивом,

Группа компаний, заинтересованных в развитии технологий Интернета, образовала Консорциум W3C (Worldwide Web Consortium), работаюший над новым стандартом HTML, который будет обязателен для клиентских обозревателей и при этом расширяемым. Ожидаемая спецификация XHTML позволит решить проблемы неверного кода, поскольку работающие на ее основе обозреватели будут обязаны отвергать документы с ошибками. Для новых документов это замечательно, однако для полного структурирования всех страниц Интернета придется переделать огромное количество существующих Web-документов. Очевидно, что это структурирование невозможно провести повсеместно, поэтому синтаксические анализаторы в обозримом будущем будут по-прежнему выполнять неверный код. Хочется надеяться, что в наступающем тысячелетии эту проблему все же решат.

SGML: достоинства

Международная организация по стандартизации (International Organization for Standardization, ISO) создает и поддерживает стандарты, помогающие работать компаниям всего мира. Так, например, болт, произведенный в Люксембурге, подойдет к гайке из Тайваня, если они были сделаны в соответствии со стандартами ISO.

Именно ISO принадлежит стандарт *обобщенного языка разметки* (Standard Generalized Markup Language, SGML), призванный обеспечить обмен информацией между компаниями, использующими разные системы. В начале 80-х годов, когда разрабатывалась спецификация SGML, на рынке было три больших игрока — IBM, DEC и Налоговая полиция США. Когда стандарт был завершен, то к нему подключились другие крупные компании и федеральные агентства. Одним из них было Министерство обороны США (Department of Defense, DoD).

Министерство обороны хотело снизить расходы на передачу контрактов от одного исполнителя другому. Дело в том, что стоимость преобразования проектной документации из формата, принятого в одной компании, в формат другой зачастую была настолько высока, что давала компании-исполнителю контракта огромное преимущество в борьбе за новые контракты (ей в отличие от конкурентов не приходилось учитывать в общей стоимости контракта средства на преобразование документации). Поэтому Министерство обороны искало технологию стандартизации документов, которая обеспечила бы их переформатирование. Этой технологией стал язык SGML, теперь обязательный для всех компаний, работающих по госзаказам. Из-за того, что первыми SGML стали использовать Министерство обороны и налоговая полиция США, аббревиатуру SGML иногда расшифровывают как Standard Government Markup Language («Язык разметки правительственной документации»). Противники сложности стандарта говорят, что это сокращение от Sounds Good, Maybe Later («Звучит неплохо, но лучше когда-нибудь потом»). Даже сами разработчики шутят, что они имели в виду Someone Get My Lithium («Кого подлечить этим снадобьем?»). Так или иначе, но SGML теперь используется для обмена информацией между партнерами во многих отраслях промышленности.

SGML стал первой технологией, позволившей отделить данные от процессов, которые с ними работают. SGML позволяет проанализировать структуру и состав данных. Результат такого анализа — *определение типа документа* (Document Type Definition, DTD). DTD — это словарь, определяющий составные элементы данного класса информации. С помощью точного синтаксиса, который называется *моделью содержания,* DTD описывает содержимое объектов конкретного класса наборов данных.

Поскольку разные наборы данных отличаются, каждое требует своего DTD. Например, DTD для описания работ по обслуживанию авиалайнера могут включать в себя элементы *крышка бака, заправка* и *горючее. В* DTD никогда не встречаются элементы форматирования документа, как в HTML (например, *center* или *bold).* Здесь нет и информации о том, как располагаются элементы (это ограничило бы область применения DTD. Эти обязанности возлагаются на ПО форматирования и обработки данных, преобразующее документ в выбранное представление.

С помощью SGML компании получили прекрасную возможность использовать информацию в одном и том же представлении в разных процессах. В обиход вошел принцип «Создай один раз и публикуй

сколько и как хочешь». Например, разработчик курсов повышения квалификации может использовать один документ SGML для создания опросного листа для студентов и для преподавателя. Разница лишь в том, что студенты получат только список вопросов, а преподаватель — те же вопросы, но с ответами. Таким образом, созданные однажды данные могут быть опубликованы многократно.

Документ SGML представляет собой обычную последовательность символов ASCII с командами разметки. Синтаксический анализатор читает такой документ и определяет его структуру согласно разметке. Поскольку документ является текстовым, синтаксический анализатор может быть компактен и работать практически на любой платформе. Эти же достоинства присущи и языку XML.

Статус стандарта ISO делает SGML очень стабильным, и в его описание трудно вносить изменения. Как и любой другой стандарт ISO, раз в пять лет комитет пересматривает SGML, решая, нужен ли он и требует ли изменений. В случае стандартизации параметров резьбы этот метод приемлем, но информационные технологии развиваются так быстро, что возникают проблемы.

SGML: недостатки

SGML не смог угнаться за развитием Web. Он был разработан в эру медленных и дорогих (по сравнению с нынешними) компьютеров. Для самых примитивных систем разработчики были вынуждены применить в SGML целый комплекс ухищрений по минимизации объема данных, В итоге эта минимизация привела к сложной и громоздкой реализации, отчего синтаксические анализаторы получались слишком большими и медленными.

Для создания документа SGML пользователь должен иметь DTD-определение с описанием структуры информации. Это приводит к необходимости заниматься глубоким анализом информации, определять ее структуру и предугадывать все сомнительные места еще до создания первого документа. За время обработки документа пользователь вынужден обращаться к DTD много раз. Кроме того, вы скоро узнаете, как трудно проверить правильность структуры документа, если документ или его DTD подверглись изменению.

Другая проблема состоит в том, что SGML создавали ученые, для которых главенствующим фактором является не удобство использования, а соблюдение принципов. Во время двух пятилетних пересмотров стандарта в 1991 и в 1996 гг. у комитета был шанс упростить SGML для работы в Web. Многое говорило о необходимости сократить и упростить синтаксис SGML так, чтобы анализатор SGML мож-

но было встроить в обычный обозреватель. Однако комитет ISO отверг эти предложения и тем самым свел SGML в могилу.

Итак, SGML ушел, а необходимость в компактном, дешевом, быстром и простом средстве, аналогичном HTML, которое могло бы еше и расширяться, как SGML, — осталась. Кроме того, это средство должно быть совместимо с прежними стандартами и работать с существующими технологиями и средствами.

Нужен XML

Поскольку комитет по SGML не хотел модернизировать стандарт, группа профессионалов Интернета обратились в консорциум W3C с предложением создать упрошенную версию SGML, которая будет работать в Web и при этом останется совместимой с HTML и SGML. В 1996 и 1997 годах группа разработала спецификацию, окрестив ее *расширяемый язык разметки* (extensible Markup Language, XML). При этом разработчики поставили перед собой десять целей.

1. XML должен быть готов к использованию в Интернете.
2. XML должен поддерживаться многими приложениями.
3. XML должен быть совместим с SGML.
4. Создание программ, обрабатывающих документы XML, должно быть простым.
5. Число необязательных возможностей языка должно быть сведено к минимуму (а лучше — к нулю).
6. Документы XML должны быть достаточно просты и понятны человеку.
7. XML следует разработать в сжатые сроки.
S. Спецификация XML должна быть четкой и краткой.
9. Создание XML-документов должны быть максимально простым.
10. Краткость команд разметки XML не имеет принципиального значения.

XML — это язык, позволяющий создавать языки разметки. Языки, средствами которых создаются языки разметки, называют *метаязыками*.

Язык XML — это техническая рекомендация Консорциума W3C. Он не принадлежит какой либо конкретной фирме-производителю, не связан с конкретным обработчиком и не привязан к конкретной платформе.

Этот язык легко изучить и использовать. Он компактен, лаконичен и оптимизирован для использования в Интернете. Как и в случае с SGML, документы XML представляют собой текстовые файлы, ко-

торые свободно проходят сквозь прокси-серверы и передаются по любым существующим сетям.

Кроме того, XML бесплатен. Консорциум W3C запатентовал только само название XML, но его спецификация опубликована на сайте *http://www.w3c.org*. XML — просто синтаксис, определенный спецификацией W3C, и ничего более.

Мифы об XML

XML часто подвергается критике, порой просто нелепой. Чтобы объяснить, что такое XML, полезно будет рассказать, чем он не является (заодно пояснив и накопившиеся за несколько лет типичные заблуждения).

XML - это язык разметки

На самом деле XML не является языком разметки, даже несмотря на то, что слово «разметка» присутствует в его названии. Это всего лишь стандарт, который определяет синтаксис, позволяющий создавать собственные языки разметки. Созданный вами язык разметки будет специфичен для ваших задач.

Для сравнения представим себе, что мы собираемся в путь, но у нас есть только кусок металла. Металл сам по себе никуда нас не повезет, оно просто лежит. Однако, поработав с ним, можно сделать автомобиль. При этом марка автомобиля будет зависеть от того, как именно мы будем работать с металлом.

К этому же куску металла можно применить другие виды обработки и вместо автомобиля получить велосипед. Технология будет той же (металл один и тот же), изменится лишь комбинация процессов. Таким образом, из одного металла мы можем получить уже два вида изделий, на которых можно отправиться в путь. Не остановившись на этом, можно сделать что-нибудь еще — например, тостер. Правда, на тостере далеко не уедешь, даже несмотря на то, что они прекрасно умеют летать в заставке на экране вашего компьютера.

В нашем примере кусок металла — это XML. Он позволяет применять технологию, но сам ничего не делает. Велосипед — это HTML. Язык разметки руководства для обслуживания авиалайнеров — это автомашина, а язык разметки для курсов повышения квалификации — тостер. Все это приложения технологии XML к конкретному виду деятельности.

XML — это только для Web

XML задумывался и разрабатывался как синтаксис, используемый для доставки содержимого через Web более эффективно, нежели с помощью HTML и SGML. На ранних стадиях развития XML многие счи-

тали, что большие компании в своем документообороте будут по-прежнему использовать SGML, а XML понадобится только для публикации некоторых документов в Интернете.

Фактически же, как сказал один из его разработчиков Тим Брэй (Tim Bray), XML— это «въезд на шоссе SGML». XML намного проще, чем перегруженный SGML, и с его помощью гораздо проще войти в мир иерархического структурирования информации- Переход же от XML к SGML обычно не вызывает труда.

Создавая XML, разработчики начали с SGML, удалив все дополнительные возможности. Поначалу было решено сохранить обратную совместимость с SGML, чтобы пользователи XML могли задействовать средства, поддерживающие SGML. И в самом деле, XML поначалу был точным подмножеством языка SGML. Однако потом к XML добавились некоторые расширения — такие, как схемы и пространства имен, что сделало его несовместимым с SGML.

Кроме того, с появлением XML многие компании, которые использовали или собирались использовать SGML, убедились в том, что функционально XML способен делать практически то же, что и SGML. Большинству таких пользователей были не нужны специфические возможности SGML. Приложения, применявшие SGML, могли быть реализованы на основе XML.

HTML — это подмножество XML

Является ли машина подмножеством металла? Нет, она просто сделана из металла. Также и HTML не является подмножеством XML, хотя этот язык можно описать средствами XML (правда, лишь в том случае, когда конкретный HTML-код строго соответствует всем требованиям спецификации HTML).

Недавно кто-то прислал мне письмо с просьбой перечислить все тэги XML. Разумеется, это невозможно, поскольку XML — не язык, а синтаксис, позволяющий создать свой собственный язык разметки для конкретного класса информации. В этом контексте HTML — просто один из таких языков.

XML - это рекламная уловка

Глава Microsoft Билл Гейтс в сентябре 1997 г. назвал XML супертехнологией. С тех пор в прессе утвердилось довольно предвзятое мнение о XML.

Теперь, чтобы привлечь внимание к своей статье, достаточно вставить аббревиатуру «XML» в заголовок. Мне однажды встретился подобный пресс-релиз для «языка разметки для вопросов и ответов», в котором от XML осталось всего два элемента: *вопрос* и *ответ*.

Еще одно недоразумение — якобы существующая связь XML с налоговой полицией США. Дело в том, что римское число XML у многих ассоциируется с формой номер 1040, в которой американцы указывают свой годовой доход и налоги. Вряд ли подобную связь следует рассматривать серьезно, но я просто рассказываю о том, что слышал.

Ну и конечно, XML не имеет никакого отношения к размеру вашей футболки.

Глава 3

Стандарты

Стандарты — основа процесса информационного обмена. Без них бизнес попросту невозможен. Возьмем для примера национальную валюту — если за ней не стоит общепринятый стандарт, ее значение будет плавать и вряд ли сможет выполнять возложенные на нее функции.

С другой стороны, множество стандартов ничем не лучше их отсутствия. На недавней конференции кто-то сказал мне: «Стандартов так много — просто выберите какой-нибудь из них!» Вряд ли говорящий подумал о том, что если я выберу этот стандарт, а он — другой, вместо единодушия воцарится хаос.

Организации, о которых я расскажу в этой главе, занимаются разработкой единых общепринятых стандартов. Как выяснилось, самый действенный метол их создания — объединить заинтересованные стороны с целью создания единых непротиворечивых методов обмена информацией,

Международная организация стандартов

Международная организация стандартов (International Organization for Standardization, ISO), базирующаяся в Женеве — мировой центр дипломатии в области стандартов. В ее состав входят национальные органы многих стран, отвечающие за поддержание сотен стандартов в разных областях. Попросту говоря, ISO — это своего рода ООН в области стандартизации.

Американский национальный институт стандартов (American National Standards Institute, ANSI) представляет США в ISO и устанавливает стандарты для американского рынка. Каждый член ISO обладает правом голоса во всех комитетах. Страны, гле нет стандартизирующих органов, представлены в ISO с правом совещательного голо-

[1] В России эту роль играет Государственный комитет Российской Федерации по стандартизации и метрологии. — *Прим. реп,*

са. Малые страны представлены в ISO как подписчики — они оплачивают подписку на материалы ISO, что позволяет им постоянно быть в курсе работы организации. Возвращаясь к нашей теме, отметим, что ISO — создатель и *владелец* стандарта SGML, предка XML.

OASIS

Организация за продвижение стандартов структурирования информации (Organization for the Advancement of Structured Information Standards, OASIS) возникла в 1993 году. Сначала она называлась «SGML Open», но в 1998 была переименована. Новое название лучше отражает главное направление деятельности этой организации — поддержку XML и стандарта представления графической информации (Computer Graphics Metafile, CGM).

Организация преследует две главных цели. Первая — всемерное содействие развитию независимых средств и систем описания структуры информации. Работа OASIS в этом направлении в основном связана со сбором и представлением информации о сушествуюших решениях в области структурирования информации. OASIS активно участвует в различных конференциях по структурированию информации и документообороту, включая SGML/XML, Documation и Seybold.

Второе важнейшее направление работы этой организации — оказание содействия компаниям, создающим средства и системы поддержки структурированною информационного обмена. В частности, по предложению своих членов OASIS создает профильные технические комитеты для работы над необходимыми стандартами — так, например, был разработан стандарт регистрации схем и их хранения в общедоступном депозитарии. Все стандарты, разработанные OASIS, общедоступны; информацию о них вы найдете на Web-сайте OASIS по адресу *http://www.oasis-open.org/*.

W3C

Консорциум W3C (World Wide Web Consortium, W3C), базирующийся в Кембридже, шт. Массачусетс — это международная ассоциация компаний, заинтересованных в развитии стандартов для Всемирной паутины.

Стандарты W3C — результат серьезной работы и длительного обсуждения всеми заинтересованными сторонами. Поскольку статус стандартов зависит от того, на какой стадии обсуждения они находятся, давайте подробнее рассмотрим принятый в W3C процесс одобрения стандарта.

1. Как правило, процедура начинается с подачи заявки (submission) одним из членов консорциума. Консорциум рассматривает заявку и решает, соответствует ли она сфере деятельности консорциума и заслуживает ли рассмотрения. Ответ не всегда положителен, поскольку некоторые компании подают заявки только для того, чтобы иметь возможность объявить о работе над «заявкой в W3C». Кстати, подать заявку может любой член консорциума. С другой стороны, факт подачи заявки не означает, что она превратится в стандарт.
2. Заявка, признанная заслуживающей рассмотрения, иногда получает статус ноты (Note). Этот статус — не более чем регистрация идеи, мнения или документа. После рассмотрения ноты ей присваивается статус принятой заявки (Acknowledged Submission). Отметим, что сам по себе факт публикации ноты по заявке не предполагает, что консорциум планирует какую-либо деятельность по этому поводу — это лишь формализованный способ публикации предложений членами консорциума,
3. После принятия заявки консорциум формирует рабочую группу (Working Group). Обычно в ее состав входят представители группы, подавшей заявку, и представители всех заинтересованных сторон. Если работа группы проходит нормально, публикуется проект стандарта (Working Draft), описывающий тему работы и ее состояние, а также свидетельствующий о намерении консорциума продолжать работу в этой области. Публикация проекта не свидетельствует о достижении согласия в рамках рабочей группы или консорциума. Обычно проекты публикуются на сайте W3C вместе с приглашением к обсуждению. Проекты стандартов иногда используются разработчиками ПО для создания прототипов, проверяющих реализуемость стандарта. Рабочая группа, как правило, приветствует подобную деятельность компаний-разработчиков ПО, однако последние должны отдавать себе отчет, что спецификация может претерпеть значительные изменения, обессмыслив проделанную компанией работу.
4. Как только, по мнению рабочей группы, проект входит в завершающую стадию, публикуется предварительная рекомендация (Proposed Recommendation). Одновременно рабочая группа подает рекомендацию на рассмотрение в Консультативный комитет (Advisory Committee).
5. Статус рекомендации свидетельствует о единодушии членов консорциума и о том, что заявка получила одобрение директора консорциума. На этой стадии консорциум рассматривает заявку с точ-

ки зрения соответствия ее идей направлению деятельности консорциума и анализирует, заслуживают ли описанные в ней технологии широкого использования.

6. Сложные стандарты часто проходят дополнительные стадии. Обычно в этом случае выпускается предварительная рекомендация, цель которой — уведомить все заинтересованные стороны, что рабочая группа считает работу над проектом практически завершенной, а спецификацию — достаточно стабильной для начала программной реализации. Тем не менее, обсуждение предварительной рекомендации может привести к небольшим изменениям спецификации.

Стандарты W3C

За время существования консорциума W3C создано немало рекомендаций. Возможно, две самых известных — HTML и CSS. В 1996 году группа связанных с Web специалистов по SGML предложила консорциуму W3C идею создания упрощенной версии SGML, которую можно будет реализовать в Web- обозревателях и серверных приложениях. Именно эта инициатива и получила название XML.

Первая версия спецификации XML 1.0 получила статус рекомендации W3C в феврале 1998 года. Рабочая группа по XML консорциума W3C начала с того, что изъяла из SGML все дополнительные возможности и упростила синтаксис. Результатом стал простой краткий синтаксис, с которым эффективно справляются небольшие анализаторы. В предыдущих главах вы узнали, что XML предлагает синтаксис для описания данных; к обработке данных он не имеет отношения. Однако заинтересованность пользователей в реализации наиболее типичных сценариев обработки данных средствами XML оказалась столь велика, что в консорциум W3C инициировал разработку соответствующих стандартов.

В этом разделе мы рассмотрим разработанные консорциумом W3C стандарты, связанные с XML (рис. 3-1).

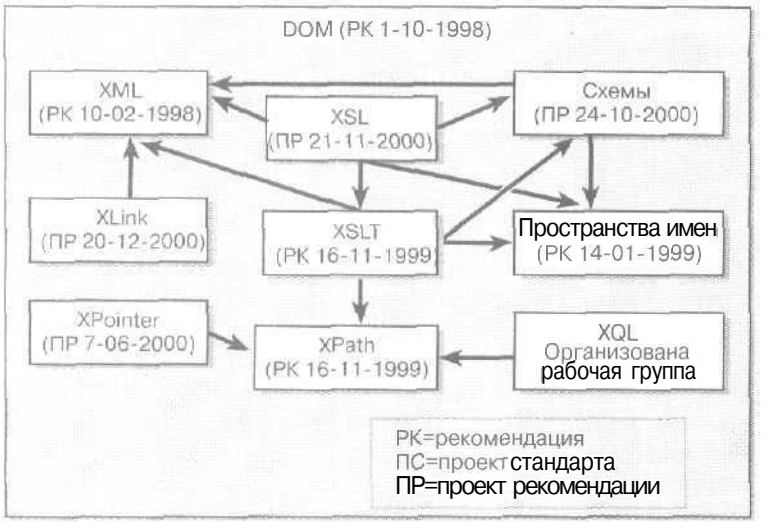


Рис. 3-1. *Консорциум W3C опубликовал несколько стандартов, связанных с XML. Подключая их к базовой спецификации объектной модели документа, разработчики могут создавать решения для конкретных областей.*

XLink и XPointer

В качестве примера стандарта, отвечающего на упомянутый в предыдущем разделе вопрос о встроенных средствах обработки, рассмотрим вопрос связывания. XML описывает структуру данных в терминах иерархических связей. У любого элемента могут быть элементы-потомки, которые, в свою очередь, могут иметь потомков. Структура документа XML описывается именно такой (и только такой) моделью, однако часто встречаются ситуации, когда объект должен быть связан с другим объектом, причем эта связь не является иерархической,

Методы связывания описаны в двух проектах стандартов W3C: XLink и XPointer. Я проиллюстрирую их на примере. Рассмотрим тэг HTML <A>:

```
<A HREF="http://architag.com/solutions/senior.html#btravis">Brian
Travis</A>
```

Этот тэг состоит из двух разных элементов.

- Тэг <A> описывает семантику ссылки или, другими словами, метод ее обработки. HTML-анализатор, встретив элемент <A>, вы-

деляет строку, заданную в атрибуте *HREF*. Кроме того, текст между открывающим тэгом <A> и закрывающим тэгом </A> в обозревателе обычно подчеркнут, а сама ссылка не отображается.

- Атрибут *HREF* задает местонахождение связанного ресурса; строка в атрибуте *HREF* состоит из доменного имени, пути и имени файла (обычно это HTML-файл). В нашем примере атрибут *HREF* указывает обозревателю, что нужно открыть файл senior.html в папке solutions сайта http://architag.com методом *HTTPget,* после чего открыть в окне часть этого файла, указанную закладкой *btravis.*

Что если наши ссылки требуют другой семантики? Например, если я хочу создать ссылку, позволяющую посетителю выбрать ресурс из списка? Понятно, что ссылка на Бетховена может на самом деле указывать на несколько ресурсов в зависимости от того, чем интересуется посетитель: на биографию композитора, на перечень его симфоний или на одноименный фильм, не имеющий никакого отношения к Людвигу ван Бетховену.

Чтобы создать на своей странице такую ссылку существующими средствами, мне придется написать сценарий, который открывает новое окно, позволяет посетителю выбрать ссылку из списка и затем открывает выбранный ресурс в окне обозревателя. XLink позволяет сделать то же самое стандартными средствами без написания сценариев — нужно лишь, чтобы обозреватель поддерживал XLink.

Кроме того, XLink поддерживает встроенные ссылки. Речь идет о включении в документ фрагментов других ресурсов — например, если вам нужно не просто сослаться на другие документы, а включить их (целиком или частично) в свою страницу.

XPointer — вторая часть нового стандарта связывания — относится (в терминах нашего примера) к атрибуту *HREF.* В нашем примере указатель ресурса задавал документ и закладку в этом документе; в результате обозреватель открывает в новом окне указанный документ, начиная с закладки.

Однако мне не всегда нужна вся страница. Что если я хочу сослаться только на один абзац этого документа, а все остальное мне не нужно? Синтаксис XPointer позволяет ссылаться на фрагменты документов с гораздо большей степенью детализации, чем атрибут *HREF.* Например, средствами XPointer можно описать ссылку XLink, указывающую на абзацы с третьего по шестой раздела с идентификатором chapter2.

Схемы

Схема описывает структуру набора данных. В спецификации XML 1.0 эти обязанности выполняет определение типа документа (DTD).

DTD — наследие SGML и поэтому не располагает встроенными средствами обработки данных. С помощью определения типа удобно описывать текстовые документы типа технических руководств, бизнес-планов и справочников пользователя, однако его возможности не соответствуют типичным для XML задачам обмена результатами выполнения запросов к БД и другими бизнес-документами.

Вскоре после одобрения рекомендации XML 1.0 несколько групп подали консорциуму W3C заявки на стандарт синтаксиса схем, который должен был значительно расширить возможности DTD: XML Data, Document Definition Markup Language (DDML), Schema for Object-oriented XML (SOX) и Document Content Description for XML (DCD). Мы обсудим XML-схемы во всех подробностях в главе 4.

Пространства имен

Этот механизм позволяет указать XML-анализатору, где он должен искать правила (схемы) для синтаксического анализа документа. Иными словами, пространство имен указывает схему, в которой описан синтаксис документа,

Посмотрим на примере, как работают пространства имен. В предыдущем разделе я сказал, что XLink и XPointer позволяют более детально описывать ссылки на фрагменты документов. Допустим, я собрал на одну страницу фрагменты нескольких XML-документов. Понятно, что в них могут встретиться элементы с одинаковыми именами, но разные по смыслу. Пусть, например, мне нужно собрать на странице три абзаца из документа Джорджа и четыре — из документа Марты. Оформлением страницы будет заниматься таблица стилей, которую я написал.

В документе Джорджа есть элемент *date,* описывающий день недели:

```
<date>13 февраля 2000 г.</date>
```

У Марты тоже есть элемент *date,* однако у нее это свидание с Джорджем:

```
<date>Обед, потом кино</date>
```

Собирая документы с этими элементами вместе, я столкнусь с *конфликтом имен.*

Пространства имен как раз и предназначены для устранения подобных ситуаций. Чтобы избежать конфликта имен, я объявляю два пространства имен, указывающие на схемы с описанием структуры соответствующих документов. Принадлежность элемента конкретному пространству имен указывается с помощью префикса:

```
<document
    xmlns:george=http://nolpreg.com/schemas/schema1.xdr
    xmlns:martha="http://schemas.martha.com/funstuff.xdr">
    ...
    <george:date>13 февраля 2000 г.</george:date>
    ...
    <martha:date>Обед, потом кино</martha:date>
    ...
</document>
```

В результате элементов с совпадающими именами больше нет. Как только элементы описаны полностью, с указанием пространства имен, их имена больше не совпадают, и я могу обрабатывать их по разному. Мы еше поговорим о пространствах имен в главе 5,

XSL, XSLT и XPath

Рекомендация XML 1.0 не предусматривает встроенных средств форматирования содержания документов XML. По этой причине появил*ся расширяемый язык стилей* (Extensible Stylesheet Language. XSL). Рабочая группа по XSL создана в основном теми, кто занимался разработкой *языка описания стиля документа* (Document Style Semantic Specification Language, DSSSL), стандарта ISO, который должен был реализовать синтаксис форматирования документов SGML. Однако сложность проблемы описания форматирования в общей постановке привела к тому, что DSSSL оказался чрезвычайно сложным. Рабочая группа по XSL хотела воспользоваться DSSSL как основой для создания нового языка, однако быстро отказалась от этой идеи (кстати, сработавшей в случае SGML-XML) — оказалось, что DSSL требует столь масштабных изменений, что проще создать новый язык «с нуля».

XSL фактически представляет собой две спецификации: одна относится к преобразованию документов XML, а вторая — к интерпретации форматирования. Основная цель XSL — преобразовать документы XML в форматированное представление, а интерпретация форматированного документа необходима лишь при создании «печатной» версии (будь то на бумаге, па Web-странице или на компакт-диске).

По мере работы над стандартом XSL рабочая группа осознала, что преобразование документов важно, даже когда документ XML не предназначен для человеческих глаз. Как следствие часть спецификации, описывающая преобразование документов, была выделена в отдельный стандарт *языка преобразования* (XSL Transformations, XSLT). Сейчас XSLT имеет статус рекомендации W3C, и уже существует не-

сколько XSLT-процессоров, основанных на этой спецификации. Фактически XSLT — это механизм преобразования структурированной информации общего назначения, поэтому если вы собираетесь всерьез использовать XML, вам придется изучить XSLT.

Кроме преобразований, в отдельную спецификацию XPath был выделен синтаксис описания шаблонов поиска. XSLT и XPath подробно обсуждаются в главе 6.

XQL

Многие члены сообщества XML выражали заинтересованность в стандартизации методов выполнения запросов к набору XML-документов. Стандарт этого рода будет эквивалентом языка структурированных запросов (Structured Query Language, SQL), стандарта описания запросов к реляционным базам данных.

В настоящий момент этот стандарт — *язык XML-запросов* (XML Query Language, XQL) — находится в стадии разработки. Он должен предоставить стандартный синтаксис описания запросов к базе данных документов XML и результатов выполнения таких запросов.

DOM

Все описанные в этой главе спецификации могут использоваться приложениями, обрабатывающими документы XML. С другой стороны, необходима модель, обеспечивающая стандартные интерфейсы доступа к свойствам документа. Этой цели служит *спецификация объектной модели документа* (Document Object Model, DOM).

Модель DOM описывает набор стандартных интерфейсов, открывающих приложениям свойства и методы документов XML. В принципе этот стандарт согласован, однако детали реализации — т. е. самое сложное — оставлены тем, кто будет реализовывать DOM. Мы подробнее обсудим этот вопрос в главе 5.

IETF

Инженерная группа Интернета (Internet Engineering Task Force, IETF) — открытое международное сообщество специалистов, компаний и исследователей в области сетей, заинтересованных в развитии архитектуры Интернета. В частности, под эгидой IETF разработаны протоколы HTTP, FTP, NNTP и другие транспортные протоколы Интернета.

Работа IETF организована примерно так же, как и у других групп по разработке стандартов. Собственно работу по различным темам — например, маршрутизации, транспортным протоколам, защите — ведут тематические рабочие группы IETF. Кроме того, трижды в год IETF проводит встречи по актуальным темам.

Чтобы стать стандартом, спецификация должна пройти через обсуждение сообществом Интернет. Это обсуждение обычно проводится в несколько итераций; по результатам обсуждения спецификация уточняется и дополняется. По окончании обсуждения соответствующий комитет IETF присваивает спецификации статус стандарта и публикует ее. Один из стандартов IETF — SOAP, который мы подробно обсудим в главе 8.

Стандарты де-факто

Помимо стандартов, которые создаются и принимаются в результате заинтересованности участников какого-либо сегмента рынка, существуют и стандарты де-факто, ставшие таковыми просто потому, что они прекрасно работают и удовлетворяют всех. Отсутствие факта их признания стандартом каким-либо стандартизирующим органом — скажем, ISO — отнюдь не значит, что ими не следует пользоваться.

Отличный пример стандарта де-факто — язык PostScript компании Adobe. Он был создан для описания разметки бумажных публикаций, причем при разработке языка учитывалась необходимость передачи материалов электронным образом. PostScript — в высшей степени полезный «стандарт», выдержавший проверку временем. Компания Adobe сделала описание языка общедоступным, что позволило множеству других компаний создать PostScript-интерпретаторы для чтения PostScript-документов, созданных на разных платформах.

Соперники Adobe настаивали на том, что для открытости рынка систем подготовки печатной продукции необходимо разработать общепринятый стандарт *языка описания страницы* (Page Description Language, PDL). В начале 90-х ISO действительно предприняло попытку создания такого независимого стандарта, полагая, что его независимости от конкретной платформы и производителя окажется достаточно для одобрения всеми участниками рынка.

После нескольких лет экспериментов рабочая группа создала стандарт языка описания страницы (Standard Page Description Language, SPDL), фактически ничем не отличающийся от языка PostScript. Чем это кончилось? Сотни компаний используют PostScript в качестве языка описания страниц, миллионы людей каждый день создают PostScript-документы, а вот о реализациях SPDL я не слышал.

Еще один стандарт, разработанный заинтересованными сторонами вне общепринятых стандартизирующих организаций, — открытая спецификация электронного документооборота (BizTalk Open Document Specification). Спецификация доступна на Web-сайте BizTalk по адресу *http://www.biztalk.org*ей посвящена вторая часть книги.

Разработка отраслевых схем

Чтобы успешно обмениваться информацией с партнерами, вам нужен общий словарь. В электронном бизнесе, основанном на XML, словари — это XML-схемы.

Кто создает схемы? На одной недавней конференции, где я рассказывал о схемах и о важности создания общих словарей для обмена информацией между партнерами по бизнесу, ко мне подошел представительный господин и спросил: «Брайан, кто разрабатывает схемы для залоговой сферы?» «Зачем ему словарь?» спросите вы. Чтобы обмениваться информацией со своими партнерами — банками и финансовыми компаниями. Он надеялся, что существует централизованный орган — скажем, какой-нибудь комитет, который создаст специализированный словарь для его отрасли. Прежде чем я скажу, как я ответил, давайте я поясню логику, приведшую к ответу.

Большинство отраслей промышленности прошли стадию самоорганизации, создав ассоциации, представляющие их общие интересы в судах, стандартизирующих органах и, что также немаловажно, перед обществом. Например, большинство производителей самолетов и авиаперевозчиков входят в Ассоциацию авиатранспорта (Air Transport Association, ATA). Несмотря на жесткую конкуренцию среди компаний-производителей и среди коммерческих авиалиний общие интересы перевешивают, заставляя их объединяться для решения общих проблем.

Много лет назад ATA опубликовала Приложение №100 (ATA-100) спецификации, описывающей обслуживание авиалайнеров. Большинство самолетов, кем бы они ни были сделаны, все равно состоят из одних и тех же компонентов, поэтому члены ATA (как производители, так и авиаперевозчики) пришли к необходимости выработки единой *спецификации обслуживания авиалайнеров* (Airline Maintenance Manual, AMM). AMM описывает стандартные процедуры обслуживания различных компонентов авиалайнера, включая электронику, двигатели, гидравлику и т. д. При покупке самолета авиалинии достаточно потребовать соответствия лайнера спецификации ATA-100, чтобы быть уверенной в возможности его обслуживания стандартными методами.

В 80-х гг. ATA предприняла проект преобразования этой спецификации в электронный вид. В результате появилась спецификация ATA-2100 (кстати, использующая синтаксис DTD), описывающая процедуры обслуживания авиалайнеров. Боинг-777 был первым самолетом, который поставлялся без единого бумажного документа — начиная с первого самолета (кстати, покупателем была компания

United Airlines) вся документация поставлялась в электронном виде по спецификации ATA-21000.

ПРИМЕЧАНИЕ Спецификация ATA-2100 написана на SGML, однако в ней нет никаких концептуальных отличий от XML.

Компания United Airlines покупает не только «Боинги» и требует от всех компаний-производителей представления документации в электронном виде по стандартной спецификации. Огромный плюс такого подхода — возможность интеграции всей документации (независимо от поставщика) в единую систему обслуживания. Этой практике следуют и другие авиаперевозчики, и поэтому компании-производители вынуждены представлять документацию в едином формате.

Эта история неоднократно повторялась в разных отраслях. Например, Совет по технологиям (Agents Council for Technology, ACT) Ассоциации независимых страховщиков США (Independent Insurance Agents of America) разработал схемы для электронного обмена страховыми требованиями, описаниями страховых случаев и ущерба, и другими документами, традиционно существовавшими только в бумажной форме.

Автомобильная промышленность и Федеральная комиссия по защите окружающей среды США совместными усилиями разработали схему J2008 для обмена информацией между производителями автомобилей и регулирующими государственными органами. Кроме того, компании-производители автомобилей пользуются этой схемой для обмена информацией со своими поставщиками. Компании, специализирующиеся в сфере автоперевозок, на базе спецификации J2008 разработали свою схему T2008.

Итак, вы уже наверное догадались, что я ответил тому презентабельному господину, специализирующемуся на залогах? Я сказал, что схемы придется разрабатывать ему самому. Нет, конечно, ему не нужно изучать XML — лучше нанять специалиста в этой области. Однако только те, кто работает в этой отрасли, могут описать предметную область, понять, какими данными они обмениваются и что должен включать их общий словарь. Только после этого специалист по XML может формализовать эти знания, превратив их в одну или несколько XML-схем.

Глава 4

Синтаксис XML

Синтаксис XML довольно очевиден и основан на нескольких правилах. Одни правила очень просты, другие потруднее, но понять их намного проще, чем большинство других Интернет-технологий.

Документ XML может быть *правильным* (well-formed) и *действительным* (valid). Правильный документ отвечает нескольким ограничениям, накладываемых спецификацией XML. Чтобы правильный документ был еще и действительным, «он должен иметь связанное определение типа документа и соответствовать ему». Про определение типа документа и действительные документы XML я расскажу в этой же главе.

Правильный документ XML

Специально для нетерпеливых ниже приведены основные требования к правильным документам XML. Подробное описание — чуть ниже.

- У каждого элемента должны быть открывающий и закрывающий тэги.
- В документе должен быть ровно один корневой элемент.
- Названия элементов и атрибутов чувствительны к регистру.
- Элементы могут быть вложены друг в друга, но не могут пересекаться.
- Некоторые символы должны предваряться управляющими символами или заменяться специальными последовательностями символов.
- Значения атрибутов должны быть взяты в кавычки.
- У пустых элементов — особый формат, которому они должны соответствовать.

Объявление XML

Правильный XML-документ начинается с необязательного *объявления*, предназначенного для синтаксического анализатора. В нем указывается тип документа — XML — и приводится дополнительная информация о нем. Объявление XML выглядит так:

```
<?xml version="1.0"?>
```

Версию указывать необходимо, даже несмотря на то, что пока существует только одна версия XML — 1.0. Как и значение любого атрибута, строка *1.0* заключается в одинарные или двойные кавычки.

Кодировка указывается после версии, например, так:

```
encoding="SHIFT_JIS"
```

Объявление кодировки указывает набор символов документа. По умолчанию считается, что документ XML имеет кодировку UTF-8 (8-разрядный Unicode) или в UTF-16 (16-разрядный Unicode). Если у документа одна из названных кодировок, этот атрибут можно не задавать.

За кодировкой следует атрибут standalone:

```
standalone="yes"
```

Он принимает значения *yes* или *no*. *Yes* информирует синтаксический анализатор, что все необходимое для обработки данного документа содержится внутри него. Иначе говоря, документ не содержит ссылок на внешние элементы, а значения по умолчанию заданы внутри документа. *No* предупреждает, что для полной обработки документа, возможно, придется обращаться к внешним источникам.

Атрибуты encoding и standalone должны располагаться внутри объявления XML. Если собрать все вместе, получится что-нибудь вроде:

```
<?xml version="1.0" encoding="EBCDIC" standalone="no"?>
```

Объявление XML должно находиться в самом начале правильного документа. Как и весь документ, объявление чувствительно к регистру символов.

Открывающий и закрывающий тэги

Документ состоит из текста и разметки. Разметка указывает приложению, обрабатывающему документ, его структуру. Разметка состоит из охватывающих тэгов, которые описывают данные документа. Тэг представляет собой инструкцию, отличающуюся от остального содержимо-

го документа тем, что она взята в угловые скобки (<>). Пара тэгов — открывающий и закрывающий — ограничивает элемент данных.

Элемент — это объект данных, являющийся составной частью структуры документа. У элемента есть свойства, важнейшее из которых — *название*. Это информативное имя объекта или типа данных. Тэги задают начало и конец элемента. Каждый элемент начинается *открывающим тэгом* и заканчивается *закрывающим*. Последний совпадает с открывающим, но начинается с символа «/» перед названием элемента. Данные между открывающим и закрывающим тэгами называются *содержимым* элемента. Содержимое может состоять из текста или даже из вложенных других элементов с их собственными тэгами. Вот пример элемента:

```
<prologue>Мы люди</prologue>
```

Открывающий тэг в этом примере — <prologue>. содержание — строка «Мы люди», а закрывающий тэг – </prologue>. Эти три части образуют элемент.

Названия элементов должны начинаться с латинской буквы или с символа подчеркивания. Остальные символы, кроме первого, могут отсутствовать или быть цифрами, латинскими буквами, символами подчеркивания или дефисами.

ПРИМЕЧАНИЕ Многие синтаксические анализаторы XML 1.0 допускают использование в названиях элементов двоеточия, однако от этой практики лучше воздержаться. Двоеточие зарезервировано для ссылки на пространства имен в рекомендации консорциума W3C «Пространства имен в XML», вышедшей после рекомендации XML 1.0. Пространства имен мы рассмотрим чуть ниже.

Корневой элемент

Документ XML должен иметь один и только один корневой элемент — элемент, открывающий тэг которого находится в начале документа, а закрывающий — в конце. За корневой тэг принимается первый открывающий тэг документа. Синтаксический разбор XML-документа заканчивается после разбора закрывающего тэга корневого элемента. Любой открывающий тэг после него считается ошибкой, поскольку рассматривается как попытка объявления нескольких корневых элементов.

Чувствительность к регистру

Названия элементов и атрибутов в HTML можно писать и прописными, и строчными буквами, поэтому открывающий тэг таблицы может

выглядеть как <TABLE>, <table> и даже <TaB1E>. Синтаксический анализатор HTML безразличен к регистру.

А вот названия элементов и атрибутов XML к регистру чувствительны. Если элемент открывается тэгом <List-item>, то закрывающий тэг должен иметь вид </List-item>, а не </LIST-ITEM>, </list-item> и т. д.

Вложенность

Элементы правильного документа должны образовывать дерево. Закрывающий тэг элемента, заключенного в другой элемент, должен находиться до закрывающего тэга родительского элемента. Большинство обозревателей часто пренебрегают этим правилом, допуская. например, такой код:

```
<P>Права <B>одного <I>человека</B> важнее</I> праа коллектива.
```

Здесь сначала начинается элемент *В,* а затем элемент *I*, однако *Б* заканчивается раньше, чем *I*. Обозреватель отобразит этот код следующим образом;

Права **одного** ***человека*** *важнее* прав коллектива.

С точки зрения XML этот код неверен, поэтому синтаксический анализатор должен отвергнуть его. Для достижения такого же эффекта по правилам XML следовало бы создать такую конструкцию:

```
<P>Права <B>одного <I>человека</I></B> <I>важнее</I> прав коллектива.</
P>
```

Особые символы

Синтаксический анализатор производит разбор документа XML, ориентируясь на несколько специальных символов, позволяющих решить, где содержимое элемента, а где разметка. Я уже упоминал три таких символа: угловые скобки (<>) и слеш (/). Если в документе надо разместить один из таких особых символов, следует использовать *подстановочные строки* (entity reference). Встретив такую строку в документе, синтаксический анализатор заменяет ее другой строкой или символом. Например, синтаксический анализатор прекратит разбор и выдаст сообщение об ошибке, если обнаружит в правильном XML-документе символ левой угловой скобки. Дело в том, что левая угловая скобка переключает анализатор на обработку открывающего или закрывающего тэга. Рассмотрим пример:

```
<P>Вася предложил выражение A<B, но мне оно не понравилось.</P>
```

Васе, мне и вам эта формула понятна, поскольку для человеческого разума здесь все просто. Но вот синтаксический анализатор подумает, что с левой угловой скобки перед буквой В начинается тэг. Далее синтаксический анализатор найдет символ В, г, е. нормальный символ для названия элемента, но потом обнаружит запятую и сочтет это ошибкой, так как запятой в названии быть не может.

Если в документе понадобилась левая угловая скобка — например. в качестве знака меньше — замените ее подстановочной строкой *<*:

```
<P>Вася предложил выражение A&lt;B, но мне оно не понравилось.</P>
```

Еще один специальный символ — амперсант (&). Подстановочная строка всегда начинается с этого символа, поэтому если вам понадобился он сам, придется использовать подстановочную строку:

```
<P>Мой мобильник фирмы AT&amp;T теперь работает гораздо лучше -
недалеко от нас установили сотовую антенну.</P>
```

Синтаксический анализатор, проверяющий документы на правильность, распознает пять подстановочных строк:

Подстановочная строка	Значение
<	< (меньше)
>	> **(больше)**
&	& (амперсант)
'	' (апостроф)
"	« (двойная кавычка)

Атрибуты

Иногда просто дать элементу название недостаточно. Для дополнительного описания элементов XML и их структуры служат атрибуты. Атрибуты содержат уточняющую информацию об элементе. Атрибуты размещаются в открывающем тэге элемента и состоят из пар название — значение:

```
<insuranceClaim dateFiled="2000-06-24">
```

Здесь название атрибута — *dateFiled*, а значение — *2000-06-24*. Название и значение разделяет знак равенства (=). При желании слева и справа от него можно добавить пробелы.

Значение должно быть в кавычках. В HTML значения элементов можно было не заключать *в* кавычки, если значение состояло из ол-

ного слова, но в XML значения всегда обрамляются кавычками. Кавычки могут быть простыми или двойными, важно лишь, чтобы открывающая и закрывающая кавычки были одинаковыми.

В значениях атрибутов могут встретиться специальные символы, которые придется заменить соответствующими подстановочными строками. Однако если в значении понадобится апостроф, его проще всего взять в двойные кавычки, не заменяя его подстановочной строкой. Тот же метод работает и для значения, содержащего двойные кавычки:

```
<driveway length="350'">
<gangster name='Волл Гамбино, кличка "Волшебные Руки"'>
```

Если же в значении атрибута встречаются и двойные кавычки. и апострофы, используются подстановочные строки:

```
<irish-gangster name='Шон O&apos;Дулл, кличка "Левша"'>
```

Разумеется, необходимо заменять левую угловую скобку:

```
<math calculation="A&lt;B">
```

Закрывающий тэг не может содержать атрибуты.

Довольно долго XML-программисты (как и их коллеги, использующие SGML) обсуждали вопрос о том, где все-таки правильно размешать информацию: в содержании элемента или в значениях его атрибута. Вот несколько соображений по этому поводу.

- В то время как элемент задает некоторый объект (например, заказ товара, номер строки, код изделия и т. д.), атрибуты указывают свойства объекта — например, дату последнего внесения изменений, автора, тип валюты и пр. Другими словами, атрибут уточняет информацию об элементе наподобие прилагательного по отношению к существительному.
- Атрибуты эффективнее. Длина самого короткого элемента - 7 байт, а самого короткого атрибута — только 5. Это различие тем заметнее, чем длиннее название элемента.
- При использовании объектной модели документа получить доступ к атрибутам легче, чем к элементам. В XSL доступ одинаково прост.
- Атрибуты не могут иметь вложенные структуры, а элемент может, Если у объекта есть дочерние объекты, то атрибут не подойдет.

Однако это всего лишь соображения, и на деле всегда приходится решать самостоятельно. Простой совет: если вы не можете решить, где разместить информацию, включите ее в содержание элемента и двигайтесь дальше.

Пустые элементы

Как я уже говорил, элемент состоит из открывающего и закрывающего тэгов и содержания. Иногда нужно создать элемент без содержания. Такие элементы тоже бывают полезны. В качестве примера рассмотрим тэг <HR> языка HTML, рисующий в окне обозревателя горизонтальную линию. У линии не может быть содержания, поэтому в дополнительном закрывающем тэге нет смысла. В HTML элемент <HR> выглядит просто:

```
<HR>
```

В правильном документе XML такая конструкция не работает, так как синтаксический анализатор XMX ожидает закрывающего тэга. Чтобы закрывающий тэг нашелся, пришлось бы написать:

```
<HR></HR>
```

Конструкция довольно глупа — ведь она ничего не говорит об истинном назначении элемента. Для решения этой проблемы разработчики XML позаимствовали из SGML полузабытый трюк — объединение двух тэгов в один тэг пустого элемента. Такой тэг начинается, как обычный открывающий тэг, и заканчивается символом «/». В нашем примере элемент будет выглядеть так:

```
<HR/>
```

Обнаружив такой тэг, синтаксический анализатор не пытается отыскать парный закрывающий тэг.

Пустой элемент может содержать атрибуты. Рассмотрим элемент языка HTML, размещающий на странице рисунок. Элемент *IMG* является пустым элементом с атрибутом *SRC,* который указывает на исходный файл изображения. Чтобы быть правильным в смысле XML, он должен выглядеть так:

```
<IMG SRC="/images/Кальян.gif"/>
```

Пустой тэг заканчивается комбинацией символов />. Слеш и правая угловая скобка должны идти подряд, без пробелов.

Комментарии

Комментарии допустимы в любом месте документа XML. Комментарий выглядит так:

```
<!- До отправки файла обязательно проверь рисунки ->
```

Синтаксический анализатор пропускает комментарии, и они не передаются приложению для обработки. Это самый лучший способ скрыть в исходном коде дополнительную информацию.

Примеры правильных документов

В этом разделе я привел несколько примеров правильных документов. В первом примере описываются компоненты конфигурации компьютера. Поскольку названия элементов хорошо известны, все здесь должно быть понятно. В этом проявляется мощь разметки, основанной на описании элементов.

```
<configuration type="printer">
 <parm name="port">/usr/lpr</parm>
 <parm name="driver">/usr/drivers/HP5SIPS</parm>
 <parm name="option">Лоток для бумаги</parm>
 <configured/>
 <online/>
</configuration>
```

В примере нет объявления XML. Как уже говорилось, объявление можно не использовать.

Вот еще один пример:

```
<?xml version="1.0" encoding="windows-1251"?>
<Joke author="Маркс Гручо">
 <Setup>Если не брать в расчет собаку,
 то книга лучший друг.</Setup>
 <Punchline>Если все же брать в расчет собаку,
 то книгу лучше держать от нее подальше.</Punchline>
</Joke>
```

Здесь в начале документа присутствует объявление. Отметим, что все открывающие и закрывающие тэги набраны в одном регистре.

Действительные документы XML

Допустим, я попросил вас оформить ваши любимые анекдоты в виде правильных XML-документов. Вы создали два документа, один с коротким анекдотом, другой -- с длинным.

```
<?xml version="1.0" encoding="windows-1251"?>
<favorite-joke author="Народ">
 <one-liner>Приходит гусь в бар и говорит бармену:
 "Налей мне стаканчик виски и запиши на мой счет".</one-liner>
</favorite-joke>
```

Формат длинного анекдота посложнее:

```
<?xml version="1.0" encoding="windows-1251"?>
<duck_joke>
 <scene number="1">
 Пришел гусь в бар. Подходит к бармену и говорит:
 "У вас есть виноград?" Бармен отвечает: "Это
 ведь бар, здесь нет винограда".
 </scene>
 <scene number="2">
 На следующий день гусь снова пришел в бар, Опять
 подходит к бармену и говорит: "У вас есть виноград?"
 Бармен отвечает: "Я уже вчера сказал тебе, что винограда
 здесь нсг. Если ты еще раз придешь сюда и спросишь
 про виноград, то я приколочу твои перепончатые лапы
 прямо к стойке бара гвоздями!"
 </scene>
 <scene number="3">
 На третий день гусь опять пришел в бар. подошел к
 Бармену и спрашивает: "У вас есть гвозди?" "Нет, конечно", -
 отвечает Бармен. Гусь ухмыльнулся и спрашивает: "А виноград?"
 </scene>
</duck_joke>
```

Оба документа содержат достаточно информации, чтобы человек догадался о назначении каждого элемента. Первый — обычная шутка (кстати, это мой любимый анекдот про гуся и бармена), второй — довольно длинный анекдот, состоящий из трех сцен. Оба документы правильны, и синтаксический анализатор XML воспримет их без ошибок.

Можно ли создать программу, которая, скажем, записывала бы эти и другие подобные анекдоты в базу данных? Оказывается, что это чрезвычайно трудно, так как документы приходят из разных источников и сформированы по-разному. У них нет обшей структуры. Нельзя предугадать, как тот или иной создатель подобного документа разместит данные и даже как он отформатирует тэги. Поэтому написать программу, которая обработала бы все анекдоты в том виде, в каком они приходят, нельзя.

Компьютеры пока не так умны, как люди, и не могут разобраться в документах, которые понятны обычному человеку. Им нужны дополнительные средства, помогающие определить структуру данных и

обработать их. От обычного текстового файла XML-документ отличается формализованным описанием структуры.

Для облегчения дальнейшего разбора документов можно разработать и разослать всем авторам документов предопределенную структуру. Тогда новый документ может быть создан в соответствии с этим образцом. Получившийся документ XML сможет автоматически обработать программа, размещающая данные в БД. Такое определение называется *описанием структуры.*

Технология описаний структуры не нова. Так. любой новый журналист получает в редакции образцы статей на свою тему для того, чтобы писать собственные материалы с учетом «местных» правил, определяющих стиль и общий тон. а также желательную структуру публикаций. Образец, например, может указывать на необходимость озаглавить материал, затем сделать отступ, поставить дату, привести план и лишь потом поместить саму статью, текст которой должен состоять из абзацев.

Имея такой образец, журналист вставляет пустой лист в пишущую машинку и начинает работу: печатает название, затем делает отступ, печатает план и собственно текст, вынимает готовую статью и относит редактору. Редактор указывает, что журналист забыл поставить дату, и возвращает материал. В результате из-за одной ошибки уже два человека теряют время. Если бы у журналиста был ангел-хранитель. который следил бы за соблюдением стиля, ошибки бы не случилось. Для тех, кто использует XML, ангелом-хранителем становятся действительность XML-документа, а набором правил — схема.

Определение типа документа (DTD)

Как я уже говорил, по спецификации XML «Документ называется действительным, когда имеет связанное с ним объявление типа документа и отвечает ему»,

Так что же такое *объявление типа документа*?

Объявление типа документа сообщает синтаксическому анализатору XML о том, где размешаются правила, по которым данный документ можно проверить. Эти правила создаются кем-то, кто определяет структуру класса документов и делает эти правила доступными синтаксическому анализатору. Анализатор сначала считывает описание структуры, а затем разбирает документ и проверяет его на соответствие грамматике из этого описания. В XML 1.0 набор правил, задаваемый объявлением типа документа, называется *определением типа документа* (Document Type Definition. DTD).

К действительному документу гораздо больше требований, чем к правильному: он не только должен соответствовать правилам оформ-

ления, о которых мы уже говорили, но и обязан быть структурирован согласно определению,

Это структурированное описание называется *схемой* (schema). В XML 1.0 только один тип схем — DTD. Определение типа документа основано на SGML, однако при этом оно было значительно сокращено, дабы избавиться от лишних дополнительных возможностей и упростить реализацию.

Для документа Joke — первого из наших анекдотов — DTD выглядит так:

```
<!ELEMENT Joke (Setup, Punchline)>
<!ATTLIST Joke author CDATA #REQUIRED
 firstTold CDATA #IMPLIED >
<!ELEMENT Setup (#PCDATA) >
<!ELEMENT Punchline (#PCDATA) >
```

Чтобы XML-документ был действительным, в начало его нужно поместить объявление типа документа, указывающее, что сначала должен быть прочитан файл схемы Joke.DTD, а затем проверена структура документа Joke и ее соответствие определению.

```
<?xml version="1.0" encoding="windows-1251"?>
<!DOCTYPE Joke SYSTEM "Joke.dtd">
<Joke author="Маркс Гручо">
 <Setup>Если не брать в расчет собаку,
 то книга лучший друг,
 </Setup>
 <Punchline>Если собаку все же брать в расчет,
 то книгу лучше держать от нее подальше.
 </Punchline>
</Joke>
```

Видимо эта единственная книга про XML, в которой мы не будем вдаваться в подробности DTD. Если читателю все же нужна подробная информация о синтаксисе или назначении отдельных фрагментов DTD, он сможет обратиться к соответствующей литературе — ее просто пруд пруди.

Целью разработки DTD было описание информации о накопленных ранее документах. Пятнадцатилетний опыт применения SGML и XML подтвердил работоспособность DTD, но некоторые факторы препятствуют использованию DTD в электронном бизнесе.

Во-первых, синтаксис DTD отличается от синтаксиса XML. Как видно из приведенного примера, у DTD весьма сжатый синтаксис со

странными словечками типа #PCDATA и ATTL1ST. Такой синтаксис несовместим даже с шестым принципом XML — он неудобен для восприятия человеком.

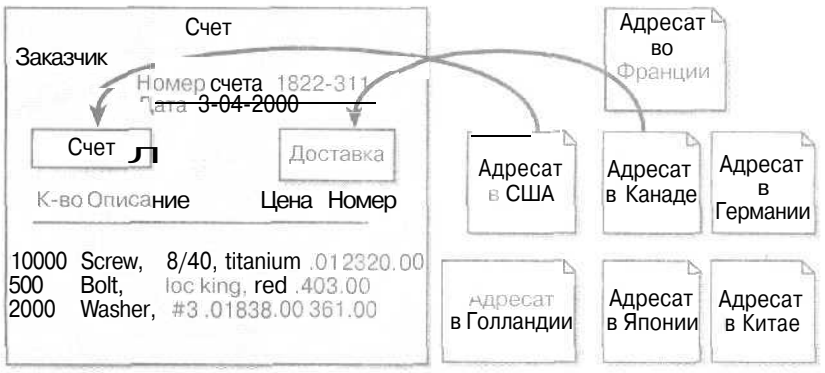
Во-вторых, в DTD используется только текстовый тип данных TEXT. При этом элемент может содержать другие элементы или текст. Узнать, например, чем является элемент содержимого документа: числом или датой, — можно только с привлечением внешнего приложения. Синтаксический анализатор может проверить лишь, является ли содержимое элемента последовательностью символов.

Одно из преимуществ XML состоит в разделении данных и процессов, которые эти данные обрабатывают. Таким образом, один и тот де документ могут обрабатывать несколько разных приложений. Каждое из них вынуждено проверять, соответствует ли содержание обрабатываемых элементов их типу. Было бы намного проще возложить эту проверку на синтаксический анализатор, который делал бы это еще до передачи документа приложениям.

Самая серьезная проблема DTD в том, что у документа может быть только одно определение типа. DTD был придуман во времена иерархической бюрократизации документооборота. Он хорошо подходит для документов, которые со временем практически не меняются. Скажем, руководства по обслуживанию авиалайнера должны строиться по традиционной схеме, поскольку ставка здесь — безопасность пассажиров. Над формированием этой структуры многие годы работали инженеры по безопасности полетов, обслуживающий персонал и контролирующие организации. И отход от правил в данном случае может привести к катастрофе.

К бизнес-документам, например, к коммерческим предложениям, счетам или медицинским заключениям, нужен иной подход. Рассмотрим счет (рис. 4-1). Обычно у счета есть заголовок, содержащий номер счета, дата и код покупателя. Далее следует адрес поставщика, а затем список заказанных товаров.

Такой счет можно выставлять покупателям со всего мира. Часть счета всегда будет оформляться одинаково независимо от местопребывания покупателя: это код покупателя, номер счета и список товаров. А вот адрес — другое дело. Если покупатель находится в США, адрес должен включать город, штат и индекс. Канадский же адрес состоит из города, названия провинции и индекса. Японский, немецкий и французский способ написания адреса также различаются между собой. Если попытаться определить наш счет с помощью иерархического подхода DTD, структура будет очень сложной.



Рис, 4-1. *Бизнес-документ может содержать другие документы. Этот счет может быть выслан заказчикам из разных стран, и при этом адрес будет указываться по-разному в зависимости от страны.*

Намного удобнее было бы описать счет с помощью одной схемы. а потом подключить к ней другую, где описывались бы только структуры, связанные с адресом. Это достигается применением синтаксиса схем и пространств имен XML.

Пространства имен

Прежде чем перейти к схемам, поговорим о рекомендации «Пространства имен в XML» консорциума W3C (*http://www.w3.org/TR/REC-xml-names*).

Пространства имен преследуют две цели: они позволяют схемам задействовать структуры и типы данных других схем и гарантируют однозначность указания элемента при наличии одноименных элементов в разных схемах.

Пространства имен указывают схемы с информацией, необходимой документу. Это достигается объявлением пространства имен с указанием местонахождения схемы, что позволяет синтаксическому анализатору обратиться к ресурсу.

Рассмотрим счет (листинг 4-1). Поскольку это счет за книги, один из его элементов называется *title* — название книги. Однако в схеме, описывающей структуру адреса, тоже есть элемент *title* — обращение к покупателю (Господин, Госпожа, Доктор и т. п.). У этих элементов *title* совершенно разное назначение, но одинаковое название. Они должны обрабатываться по-разному, однако есть риск спутать один элемент с другим.

Листинг 4-1. Конфликт имен - это неоднозначность, когда у элементов с разными свойствами одинаковые названия. В данном случае это элемент title как обращение к человеку и как название книги.

Invoice.xml

```
<invoice>
 <number>A-99-1443</number>
 <customer>RA-81122</customer>
 <billto>
 <company></company>
 <contact>
 <title>Мистер</title>
 <name>Барнс</name>
 </contact>
 <street1></street1>
 <street2></street2>
 <cityx/city>
 <state></state>
 <zipx/zip>
 </billto>
 <body>
 <line>
 <title>Святой Источник</title>
 <quantity>600</quantity>
 <price>4.12</price>
 </line>
 <line>
 <title>XML-программирование для серверов BizTalk</title>
 <quantity>1000</quantity>
 <price>49.99</price>
 </line>
 <line>
 <title>1001 шутка про гуся в баре</title>
 <quantity>1000</quantity>
 <price>6.34</price>
 </line>
 </body>
</invoice>
```

Для устранения неоднозначности надо использовать объявление пространства имен. Пространство имен должно быть объявлено либо перед, либо внутри первого элемента, входящего в это пространство имен. Внешне объявление похоже на обычный атрибут в открывающем тэге. Все объявления пространства имен начинаются со служебной строки *xmlns*, за которой указывается название локального пространства имен (о названиях пространств имен чуть позже),

Значение атрибута в объявлении пространства имен представляет собой указатель ресурса схемы, используемой в документе.

ПРИМЕЧАНИЕ В спецификации пространства имен консорциума W3C говорится; «Чтобы пространство имен выполняло свою роль, его название должно быть уникальным и неизменным. Название пространства имен не обязано служить для извлечения схемы (если она существует). Синтаксис, отвечающий этим целям, описан в спецификации универсального имени ресурса (Uniform Resource Name, RFC2141). Однако тех же целей можно достичь, применяя обычные указатели URL».

В нашем примере со счетом пространство имен задается внутри открывающего тэга:

```
<invoice xmlns="http://architag.com/schemas/invoice.xdr">
```

Приложение, считывающее документ-счет, может обратиться к указанному ресурсу, получить схему и проверить структуру документа на соответствие правилам схемы. Поскольку объявление пространства имен находится в элементе корневого уровня документа, оно распространяется на все содержимое документа до тех пор, пока не встретится другое пространство имен.

Пространство имен относится к элементу, в котором оно объявлено, точно так же, как к нему относятся дочерние элементы. В элементе можно задействовать столько пространств имен, сколько нужно, но при этом текущим всегда является пространство, объявленное последним.

Что означает «объявленное последним»? Вспомним, что у XML иерархический синтаксис. Все элементы документа являются потомками корневого элемента- Эти элементы соотносятся друг с другом согласно иерархии. Пространство имен распространяется на все элементы, происходящие от данного родительского элемента, до тех пор. пока не будет объявлено другое пространство имен. Новое пространство имен действует в пределах элемента, в котором оно объявлено.

Вернемся к нашему счету. Создавая его, мы предусмотрели место для подключения блока адреса. Такой тип структуры называется моделью с открытым содержанием, так как содержимое документа не заполнено. Приведенный ниже документ иллюстрирует область действия пространства имен (листинг 4-2).

Листинг 4-2. Пространства имен применяются в счете для встраивания в отведенное место блока с адресом.

Nsinvoice.xml

```
<?xml version="1.0"?>
<invoice xmlns="http://architag.com/schemas/invoice.xdr">
 <number>21153</number>
 <customer>SY221-00</customer>
 <date>2000-01-24</date>
 <address xmlns="http://xml.org/schemas/us-address.xdr">
 <street>750 Davis Street</street>
 <city>San Francisco</city>
 <state>CA</state>
 <zipcode>94107</zipcode>
 </address>
 <body>
 <item num="YI-2289" qty="100" price="23.11"/>
 <item num="WD-7198" qty="35000" price=".137"/>
 <item num="ER-3211" qty="120" price="112.00"/>
 </body>
</invoice>
```

В этом листинге корневой элемент *invoice* содержит указание пространства имен, а его дочерний элемент *address* использует другое пространство имен. Открыв документ, указанный в атрибуте *xmlns*, вы увидите, что в пространстве имен invoice (счет) определены элементы *number* (номер счета), *customer* (покупатель) и *date* (дата выставления счета), в то время как пространство имен address (адрес) определяет элементы *street* (адрес), *city* (город), *state* (штат) и *zipcode* (индекс),

Всякий раз при объявлении пространства имен все находящиеся в зоне его действия дочерние элементы автоматически относятся к этому пространству имен. Это так называемое пространство имен по умолчанию. Возникает естественный вопрос: как смешивать пространства имен внутри одного элемента? Что будет, например, если

находящийся в элементе *invoice* элемент *date* должен иметь тип данных, описанный в другом месте?

Б этом случае внутри открывающего тэга элемента *invoice* можно определить несколько пространств имен. Поскольку в любой момент может действовать только одно пространство имен по умолчанию, при использовании нескольких пространств имен придется явно указывать их с помощью префикса в имени элемента. Если элемент является членом другого адресного пространства, его префикс надо добавлять перед названием элемента в его открывающем и закрывающем тэгах.

Счет с несколькими пространствами имен представлен в листинге 4-3. Два пространства имен указывают схемы, в которых описаны элементы счета.

Листинг **4-3. Счет с** несколькими пространствами имен.

Mnsinvoice.xml

```
<?xml version="1.0"?>
<invoice xmlns="http://architag.com/schemas/invoice.xdr"
 xmlns:dt="http://schemas.com/schemas/datatypes.xdr">
 <number>21153</number>
 <customer>SY221-00</customer>
 <date dt:type="date">2000-01-24</date>
 odd ress xmlns="http://xml.org/schemas/us-address.xdr">
 <street>750 Davis Street</street>
 <city>San Francisco</city>
 <state>CA</state>
 <zipcode>94107</zipcode>
 </address>
 <body>
 <item num="YI-2289" dt:qty="100" dt:price="23.11"/>
 <item num="WD-7198" dt:qty="35000" dt:price=".137"/>
 <item num="ER-3211" dt:qty="120" dt:price="112.00"/>
 </body>
</invoice>
```

В третьей строке объявляется пространство имен, которое можно использовать где угодно внутри элемента *invoice,* но при этом нужно явно указывать в виде префикса название этого пространства имен в открывающих и закрывающих тэгах элементов и в названиях атрибутов. В шестой строке атрибут элемента *dare* использует пространство

имен *dt,* а с помощью атрибута *type* выбирается тип данных *date.* Отметим, что пространство имен *dt* используется и атрибутами в элементе *body* (строки с 14 по 16).

Еще один способ указать пространство имен — просто использовать соответствующий префикс для всех элементов. Такой подход вообще освобождает от необходимости использовать какое бы то ни было пространство имен по умолчанию (листинг 4-4).

Листинг 4-4. В начале каждого элемента можно указывать префикс пространства имен. При этом сразу видно, к какому пространству имен принадлежит каждый из элементов, но создавать такой код не очень удобно.

Prefixns.xml

```
<?xml version="1.0"?>
<inv:invoice xmlns:inv="http://architag.com/schemas/invoice.xdr"
 xmlns:dt="http://schemas.com/schemas/datatypes.xdr">
 <inv:number>21153</inv:number>
 <inv:customer>SY221-00</inv:customer>
 <inv:date dt:type="date">2000-01-24</inv:date>
 <addr:address
 xmlns:addr="http://xml.org/schemas/us-address.xdr">
 <addr:street>750 Davis Street</addr:street>
 <addr:city>San Francisco</addr:city>
 <addr:state>CA</addr:state>
 <addr:zipcode>94107</addr:zipcode>
 </addr:address>
 <inv:body>
 <inv:item inv:num="YI-2289"
 dt:qty="100" dt:price="23.11"/>
 <inv:item inv:num="WD-7198"
 dt:qty="35000" dt:price=".137"/>
 <inv:item inv:num="ER-3211"
 dt:qty="120" dt:price="112.00"/>
 </inv:body>
</inv:invoice>
```

Схемы

После выхода в феврале 1998 г. рекомендованной спецификации XML 1.0 консорциум W3C начал работу над альтернативной спецификацией, которая бы заменила DTD в части описания схем. Работа над спецификацией XML-схем должна быть закончена в конце 2000 г.[1]

Группа разработки спецификации схем придерживалась трех основных направлений.

- Схемы должны следовать синтаксису XML, что позволит использовать для их обработки те же средства, что и для документов XML.
- Стандарт схемы должен поддерживать распространенные типы данных (число, дата и т. п.) и типы данных, определяемые пользователем. Это позволит возложить заботы по проверке корректности данных на синтаксический анализатор XML.
- Схема XML должна иметь возможность строиться по принципу модели с открытым содержанием. Скажем, в примере со счетом нам нужна была гибкость обработки адреса. XML-схемы позволяют описать схему информации для счета, оставив место для подключения схемы с описанием структуры адреса.

ПРИМЕЧАНИЕ W3C разрабатывает спецификацию типов данных для DTD (Datatypes for DTDs, DT4DTD), которая добавит поддержку некоторых типов данных в DTD. Это расширение синтаксиса DTD должно позволить приверженцам DTD получить доступ ко многим типам данных. Естественно, чтобы это заработало, новую спецификацию должны будуг поддерживать синтаксические анализаторы XML.

Предлагаемая консорциумом XML-схема должна преодолеть и другие недостатки DTD.

XDR — сокращенная версия XML Data

Microsoft довольно быстро поддержала XML и подключилась к работе консорциума W3C над стандартами XML. Компания участвовала в техническом анализе спецификаций, а также во многих комитетах, связанных с XML. В январе 1998 г. компании Microsoft, DataChannel, ArborText и Inso выдвинули альтернативный синтаксис для схем – XML Data. Этот синтаксис позволил решить не только перечисленные выше задачи, но и некоторые другие.

[1] На начало январи 2001 г. работа нал схемами не завершена, однако опубликованы предварительные рекомендации по структурам и типам данных XML-схем.

В декабре 1998 г. появился обозреватель Microsoft Internet Explorer 5 со встроенным синтаксическим анализатором MSXML (он реализован как COM-сервер). В этом синтаксическом анализаторе используется сокращенная версия XML-схем XDR (XML Data Reduced),

Поскольку рабочая группа консорциума W3C еще не завершила свою работу над официальной версией схемы XML, Microsoft рекомендовала XDR как синтаксис схем для BizTalk. Microsoft предполагает перейти на стандарт схем W3C (он носит название XSD), как только он будет завершен.

Стоит ли использовать XML Data сегодня, если завтра ожидается новый стандарт схем? Не понадобится ли завтра переписывать приложение, созданное сегодня с применением BizTalk и XML Data? Microsoft и другие производители обещают предоставить средства, позволяющие конвертировать XDR в XSD. А поскольку возможности обоих синтаксисов одинаковы, автоматическое преобразование одного синтаксиса в другой должно быть тривиально. Поскольку оба синтаксиса построены на XML, для перевода одного синтаксиса схемы в другой можно использовать шаблоны XSLT.

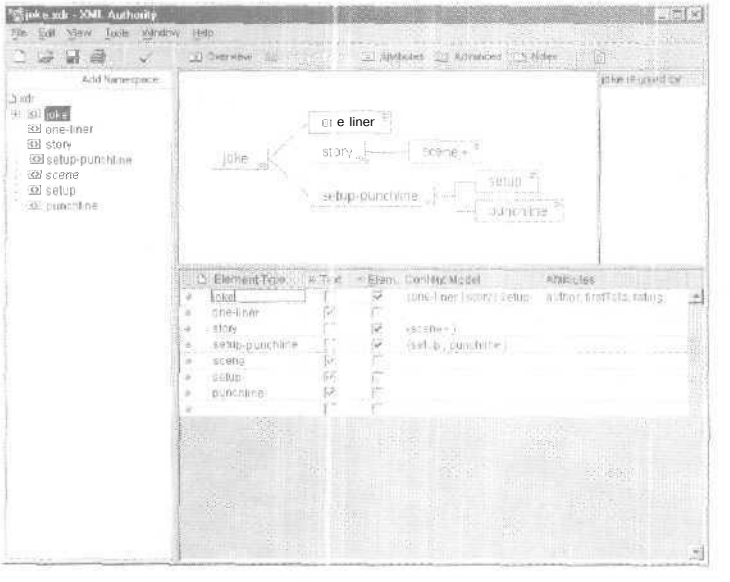


Рис. 4-2. Редактор схем XML Authority компании Extensibility Inc.

Уже сейчас компания Extensibility (*http://www.extensibility.com*) поставляет редактор XML Authority (рис. 4-2), позволяющий создавать

и поддерживать схемы в графическом виде, не заботясь о том, как именно выглядит синтаксис. Этот продукт позволяет отыскать наиболее важные места схемы, легко разобраться, что к чему, и представить иерархию данных в графическом виде. XML Authority может поддерживать самые разные синтаксисы схем. Можно открыть любую схему и сохранить ее в другом синтаксисе.

Можно загрузить правильный документ XML, и XML Authority построит на основании этого документа схему. XML Authority позволяет создавать схемы на основе объектов COM, Java-классов и источников данных ODBC.

Сокращенный синтаксис XML Data документирован на сайте Microsoft XML Developer Center по адресу *http://msdn.microsoft.com/xml*. Вот пример схемы XDR (листинг 4-5):

Листинг 4-5. Пример схемы XDR.

Joke.xdr

```
1 <?xml version ="1.0" encoding="windows-1251"?>
2 <Schema name="Joke.xdr"
3 xmlns="urn:schemas-microsoft-com:xml-data"
4 xmlns:dt="urn:schemas-microsoft-com:datatypes">
5 <ElementType name="joke" content="eltOnly" order="one">
3 <AttributeType name="author" dt:type="string"
7 required="yes"/>
8 <AttributeType name="firstTold" dt:type="dateTime.tz"/>
9 <AttributeType name="rating" dt:type="enumeration"
10 dt:values="Хор Отл Плох Неизв" required="yes"/>
11 <attribute type="author"/>
12 <attribute type="firstTold"/>
13 <attribute type="rating"/>
14 <element type="one-liner"/>
15 <element type="story"/>
16 <element type="setup-punchline"/>
17 </ElementType>
18
19 <ElementType name="one-liner" content="textOnly"/>
20 <ElementType name="story" content="eltOnly" order="seq">
21 <element type="scene" minOccurs="1" maxOccurs="*"/>
22 </ElementType>
23
```

```
24 <ElementType name="setup-punchline" content="eltOnly"
25 order="seq">
26 <element type="setup"/>
27 <element type="punchline"/>
28 </ElementType>
29
30 <ElementType name="scene" content="textOnly"/>
31 <ElementType name="setup" content="textOnly"/>
32 <ElementType name="punchline" content="textOnly"/>
33 </Schema>
```

Схема из листинга 4-5 описана в табл. 4-1.

Табл. 4-1. Описание схемы из листинга 4-5.

Строка	Описание
1	Схема представляет собой документ XML.
2	Корневой элемент начинается с открывающего тэга *Schema*. Название схемы указано в атрибуте *name*.
3–4	Спецификация схемы XDR включает два пространства имен: для структур и для типов данных, используемых в схеме.
5–17	Элемент *ElementType* объявляет элемент и его содержимое. В данном случае объявляется элемент joke. Его атрибуты указывают, что он содержит только элементы (content=«eltOnly»), причем только один (order=«one»).
6–7	Элемент *AttributeType* объявляет атрибуты. Здесь объявлен атрибут author со строковым типом данных (dt:type=«string»). Это обязательный атрибут элемента *joke* (required=«yes»).
8	Объявлен атрибут *firstTold* с типом даты. Тип данных *dateTime.tz* определен спецификацией XDR как подмножество стандарта ISO 8601 для указания даты, времени и часового пояса. Значения будут выглядеть примерно так: 2000-04-25T09:00-08:00, что означает 9 часов 25 апреля 2000 г. в часовом поясе с 8-часовой разностью с Гринвичем. Атрибут *firstTold* не является обязательным.
9–10	Атрибут *rating* перечислимого типа (enumerated), т. е. этот атрибут может принимать только значения из списка, заданного в атрибуте *dt:values*. В данном случае это оценки анекдота: «Хор», «Отл», «Плох» и «Неизв». Атрибут *rating* — обязательный.

(продолжение)

11–13	Атрибуты элементов должны быть размешены с помощью элемента *attribute*.
14-16	Элемент *element* указывает дочерние элементы элемента *joke*. Атрибут *content* элемента *ElementType* указывает, что внутри родительского элемента может содержаться только один из трех элементов. Поэтому каждый из этих элементов должен быть объявлен отдельным элементом *ElementType*.
24–28	Здесь объявляется элемент *setup-punchline*. Атрибут order=«seq» указывает, что перечисленные элементы *(setup* и *punchline]* должны идти в порядке, указанном в объявлении. Поэтому и в документе элемент *setup-punchline* должен состоять из одного элемента *setup* и следующего за ним элемента *punchline*.
30-32	Весь документ состоит из этих трех элементов.

Рабочая группа консорциума W3C продолжает работу над спецификацией схем XML. В апреле 2000 г. вышли первые три части предварительной спецификации схемы XML. Эта схема расширит возможности DTD для XML 1.0. Однако в BizTalk для схем документов пока что используется синтаксис XDR.

Главное различие XSD и XDR состоит в деталях синтаксиса схем; никаких принципиальных различий между ними нет. Компания Microsoft решила временно реализовать XSD, поскольку эта спецификация уже существует, а после выхода стандарта перейти сразу на XDR.

Глава 5

Работа с XML из приложений

Итак, вы познакомились с синтаксисом XML-документов и схемами, описывающими их структуру. Попробуем теперь создать приложение для работы с этими данными. В этой главе вы узнаете о двух основных методах работы с XML-документом: как с деревом, находящимся в памяти, и как с объектом, генерирующим события по мере его обработки анализатором или другим приложением.

Объектная модель документа

Объектная модель документа (Document Object Model, DOM) открывает доступ к XML-документу как к древовидной структуре в памяти. DOM представляет XML-документ в виде объекта, доступного как любой другой объект в современных языках программирования.

DOM описывает стандартный набор объектов и интерфейсов, обеспечивающих работу с XML-документом, его элементами и их атрибутами. DOM позволяет представить XML-документ в объектном виде, с которым можно работать как с любым другим объектом посредством хорошо документированного интерфейса прикладного программирования, описывающего свойства и методы объекта.

Как вы помните, DOM имеет статус рекомендации консорциума W3C. Поскольку создание спецификации DOM — большой и сложный проект, рабочая группа W3C по объектной модели документа все еще продолжает свою деятельность. По естественным соображениям группа разделила проект на несколько частей. Первая часть спецификации была принята в октябре 1998 г. Вторая, надеюсь, будет завершена в первой половине 2000 г.[1]

[1] О текущем статусе спецификации DOM можно узнать на странице *http://www.w3.org/DOM/DOMTR*. - *Прим. ред.*

Рекомендация W3C — описание общей объектной модели, однако она не определяет конкретную реализацию DOM. Вследствие этого разные реализации DOM скорее всего будут несколько различаться в части свойств и методов объекта-документа. Например, в версии DOM, описанной в рекомендации W3C, нет нескольких важных интерфейсов, которые Microsoft включила в свою реализацию. Кстати, в приведенных ниже примерах я буду часто использовать эти методы, *selectNodes* и *selectSingleNode*, отсутствующие в спецификации W3C. Другие анализаторы предлагают свою реализацию DOM, поэтому, выбирая анализатор, посмотрите, реализует ли он нужные вам свойства и методы.

Я сосредоточусь на реализации DOM, выполненной Microsoft, не в последнюю очередь благодаря ее подробной документации и отличной поддержке. Microsoft DOM — это составная часть анализатора Microsoft XML и его объектной модели. Объектная модель Microsoft XML DOM включена в Microsoft Internet Explorer 5, Microsoft Office 2000 и Microsoft Windows 2000. Кроме того, вы можете использовать возможности XML DOM в своем приложении. DOM реализован в виде библиотеки msxml2.dll (подробнее о ее версиях — во врезке) и регистрируется как COM-объект под именем MSXML2.DOMDocument. Поскольку DOM — это COM-сервер, его можно использовать в любом COM-совместимом приложении. Если вы работаете со сценарными языками, он доступен как объект ActiveX Microsoft.XMLDOM.

Версии XMLDOM

Разработка XML DOM продолжается, поэтому на вашей машине может оказаться сразу несколько версий библиотеки xmldom.dll. Работая с объектом *MSXML.DOMDocument* вы имеете дело с версией 2.5; имя *MSXML2.DOMDocument* соответствует версии 2.6, а *MSXML.DOMDocument30* — версии 3. Утилита xmlinst.exe позволяет заменить все ссылки в реестре на последнюю (третью) версию библиотеки MSXML. В примерах этой главы я использую версию 2.6.

Как работает DOM

DOM можно представлять себе как динамический иерархический объект с набором интерфейсов, свойств и методов. Важно помнить, что с точки зрения компьютера объект, который мы называем XML-документом, — это просто последовательность байтов. Кстати, поскольку эти документы являются текстовыми, их можно без особых проблем пересылать по сети и Интернету,

Однако для работы XML-документ надо считать в память[2] и превратить в объект в смысле современных объектных моделей программирования. Это делается с помощью объекта DOM, который вызывает XML-анализатор, считывающий документ и превращающий его в древовидный объект (рис. 5-1).

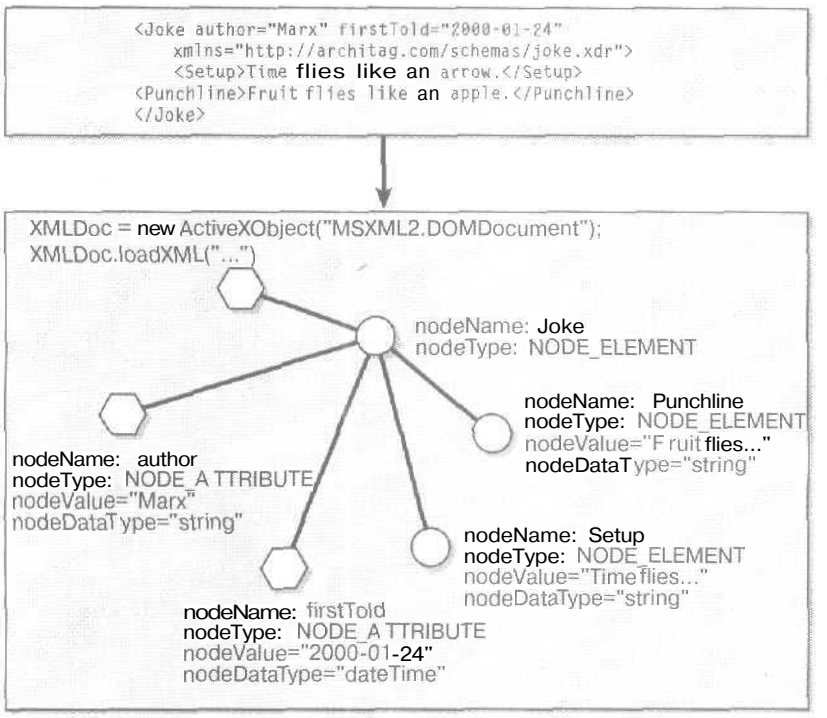


Рис. 5-1. DOM предлагает набор стандартных интерфейсов, обеспечивающих программный доступ к иерархическому представлению объектов XML-потока.

Небольшой XML-документ в верхней части рисунка представляет шутку (вы уже, наверное, поняли, что это моя любимая тема). В документе есть элементы *Joke, Setup* и *Punchline* с атрибутами *author* и *firstTold*. XML-анализатор посимвольно считывает документ, отделяя символы разметки от содержания. Если документ не содержит ссыл-

[2] Считывание XML-документа в память до начала обработки является необходимым условием работы DOM. Это часто неудобно и, как правило, неэффективно, чем и вызвано появление событийно-ориентированного интерфейса SAX. — *Прим. ред.*

ки на схему, анализатор применяет правила разбора правильных XML-документов. Если же схема есть, анализатор считывает схему и проверяет документ на соответствие структуре, заданной в схеме.

Считав и проверив документ, анализатор создает набор связанных узлов с некоторыми свойствами (мы обсуждали узлы в главе 4). Вот типы узлов, допустимые в реализации Microsoft DOM (табл. 5-1):

Табл. 5-1. Типы узлов **XML-документа в реализации Microsoft DOM.**

Значение	Имя	Описание
1	NODE_ELEMENT	Эти узлы представляют элементы. У элементного узла могут быть потомки следующих типов: Element. Text, Comment, ProcessingInstruction, CDATA-Section и EntityReference. В то же время он сам может быть потомком узлов типа Document, Document-Fragment, EntityReference и Element.
2	NODE_ATTRIBUTE	Атрибут элемента. У такого узла могут быть потомки типа Text и EntityRefe rence. Атрибутный узел не может быть потомком узлов другого типа; в частности, он не является потомком узла соответствующего элемента.
3	NODE_TEXT	Текстовое содержание тэга. У текстовых узлов не бывает потомков. Сам текстовый узел может быть потомком узла типа Attribute, DocumentFragment, Element и EntityReference.
4	NODE_CDATA_ SECTION	Раздел CDATA исходного XML-документа. Разделы типа CDATA служат для маскировки текстового содержания, которое в противном случае интерпретировалось бы как разметка, У узла типа CDATA не бывает потомков, а сам он может быть потомком узлов типа DocumentFragment, Entity-Reference и Element.

(*продолжение*)

5	NODE_ENTITY_ REFERENCE	Подстановочная строка. Узел этого типа может содержать подстановочные строки любого типа, включая односимвольные. У такого узла могут быть потомки типа Element, ProcessingInstruction, Comment, Text, CDATASection и EntityReference. Он может быть потомком узлов типа Attribute, DocumentFragment, Element и EntityReference.
6	NODE_ENTITY	Подстановочная строка в развернутом виде. Узлы этого типа могут иметь потомков, представляющих подстановочную строку (например, узлы типа Text и EntityReference). Единственный узел, потомком которого может быть узел такого типа, — DocumentType.
7	NODE_PROCESSING_INSTRUCTION	Инструкции по обработке (Processing Instruction, PI) XML-документа. У узлов этого типа не бывает потомков; сами они могут быть потомками узлов типа Document, DocumentFragment, Element и EntityReference.
8	NODE_COMMENT	Комментарий. У узлов этого типа не бывает потомков; сами они могут быть потомками узлов типа Document. DocumentFragment, Element и EntityReference.
9	NODE_DOCUMENT	Документ. Это корневой узел, обеспечивающий доступ к объекту, представляющему XML-документ. Создается с помощью ProglD «MSXML2.DOMDocument» или в сценарном разделе посредством объявлений <SCRIPT LANGUAGE=XML> или <XML>. Узел этого типа может иметь потомков типа Element (максимум одного), Processing Instruction, Comment и DocumentType. Как корневой узел он. естественно, не может быть потомком другого узла.

(продолжение)

Значение	Имя	Описание
10	NODE_DOCUMENT_TYPE	Определение типа документа, указанное в тэге <!DOCTYPE>. Узел этого типа может иметь потомков типа Notation и Entity, а сам быть потомком узла типа Document.
11	NODE_DOCUMENT_FRAGMENT	Фрагмент документа. Узел этого типа позволяет связать узел или часть дерева с фрагментом, не обязательно являющимся частью документа (это, например, полезно при копировании или перемещении фрагментов). Узел этого типа может иметь потомков типа Element, ProcessingInstruction, Comment, Text. CDATASection и EntityReference, а сам не может быть потомком другого узла.
12	NODE_NOTATION	Обозначение, использованное в определении типа документа. У узлов этого типа не бывает потомков, а сами они могут быть потомками узла типа DocumentType.

Свойства и методы узла определяются его типом. Например, можно загрузить в узел типа «Документ» XML-документ как последовательность байтов, однако загрузить отдельно узел типа «Элемент» или «Атрибут» нельзя. Чтобы появилась возможность обращения к элементам и атрибутам, нужно сначала загрузить документ в корневой узел.

Документ Joke (рис. 5-1) состоит из 4 узлов. Первые два — элементные узлы *Setup* и *Punchline,* другие два — атрибутные узлы *author* и *firstTold.* В открывающем тэге <Joke> присутствует объявление пространства имен со ссылкой на внешнюю схему. Схема следует синтаксису XML Data Reduced **(XDR).** Обратите внимание на свойство *nodeDataType* (оно есть у всех узлов), указывающее тип данных узла. Это свойство имеет значение *string для* всех узлов, кроме атрибутного *узла firstTold,* тип данных которого *dateTime.* Заметьте: если пространство имен не задано, все узлы будут иметь строковый тип данных (*nodeDataType=«string»*).Свойство *typedValue* позволяет получить форматированное значение узла — например, дату в случае узла типа *dateTime*; значение возвращается в переменной типа variant, так что

вам не придется ни проверять его корректность, ни преобразовывать его к нужному типу.

Подробное описание интерфейса Microsoft DOM см. по адресу *http://msdn.microsoft.com*. Как и большинство прикладных интерфейсов, DOM предлагает объектную модель с богатыми возможностями. Она позволяет выполнять разнообразные операции с XML-документами, включая загрузку и сохранение документа, создание узлов и элементов и, конечно же, синтаксический анализ документа. Как и у большинства интерфейсов, в этой модели есть несколько методов и свойств, особо популярных у разработчиков. Ими-то мы и займемся.

Создание объекта DOM

Первое, что нужно сделать для работы с DOM, — создать экземпляр объекта XML-анализатор/DOM. В сценарии на JavaScript этим занимается метод *ActiveXObject*:

```
var objDocument = new ActiveXObject("MSXML2.DOMDocument");
objDocument.async = false;
```

В первой строке примера создается экземпляр объекта; он хранится в переменной *objDocument* типа variant. После загрузки документа этот объект будет содержать корневой узел. Если хотите проверить, посмотрите значение свойства *objDocument.nodeType* — должно получиться *9*, что соответствует узлу типа NODE_DOCUMENT (табл. 5-1).

Свойство *async* указывает, должен ли анализатор загрузить документ целиком прежде, чем сделать его доступным. Значение *false* гарантирует от выполнения операций с не полностью загруженным документом. Это самый простой и надежный, но, естественно, и самый медленный способ. Если свойству *async* присвоено значение *true* (это значение по умолчанию), управление возвращается вызывающей программе до завершения загрузки документа. При этом придется проверять статус загрузки документа с помощью свойства *readyState*. Есть и другой метод — создать обработчик для события *onreadystatechange* или просто отслеживать появление этого события.

Один из случаев, когда свойству *async* лучше присвоить значение *true*, — загрузка больших документов. Это позволяет приложению продолжить работу, пока документ загружается. Фактически при этом загрузка выполняется отдельным потоком. Применяя асинхронную загрузку, обязательно проверьте значение свойства *readyState* перед обращением к документу. Возможные значения этого свойства перечислены в табл. 5-2. Два метода загрузки XML-документа описаны в табл. 5-3.

Табл. 5-2. Возможные значения свойства readyState.

Значение	Состояние	Описание
1	LOADING	Происходит загрузка документа; анализ еще не начался.
2	LOADED	Считывание документа закончено; начат анализ.
3	INTER-ACTIVE	Анализ документа частично завершен; часть дерева доступна.
4	COMPLETED	Документ загружен (удачно иди нет),

Табл. 5-3. Методы загрузки XML-документа в объект DOM.

Метод	Описание
load(указательресурса)	Загружает документ, находящийся по адресу. заданному указателем ресурса. Если указатель некорректен или не указывает на XML-документ, свойству *documentElement* присваивается значение *null* и возвращается сообщение об ошибке.
loadXML(xml-строка)	Загружает XML-документ, представленный в виде строки. В качестве аргумента может выступать правильный или действительный документ. Если при анализе документа обнаружены ошибки, свойству *documentElement* присваивается значение *null* и возвращается сообщение об ошибке.

Вот пример кода, загружающего XML-строку в объект DOM:

```
objDocument.loadXML("<fact verified='2000-01-24'>Фильмы " +
 "лучше книг, потому что на них" +
 " нельзя вылить кофе.</fact>");
```

После загрузки объекта надо проверить свойство *errorCode* объекта *objDocument.parseError*. Если анализ закончился нормально, оно имеет значение 0:

```
if (objDocument.parseError.errorCode != 0)
 {
 alert("Ошибка: " + objDocument.parseError.reason +
```

```
" в строке " + objDocument.parseError.line);
}
```

Свойства объекта *parseError* (он доступен только для чтения) перечислены в табл. 5-4.

Табл. 5-4. Свойства объекта parseError.

Свойство	Описание
errorCode	Десятичный код ошибки.
url	Указатель ресурса XML-файла, при анализе которого произошла ошибка.
reason	Описание ошибки.
srcText	Строка, в которой произошла ошибка.
line	Номер строки, в которой произошла ошибка. Имейте в виду: нумерация ведется от начала документа, однако если документ содержит определение типа (DTD), нумерация начинается со строки, следующей за закрывающим тэгом DTD, а не с первой строки документа.
linepos	Номер символа, вызвавшего ошибку, отсчитываемый от начала строки.
filepos	Номер символа, вызвавшего ошибку, отсчитываемый от начала файла.

Работа с документом

Убедившись, что документ загружен, с ним можно начать работать. Это можно сделать по-разному. Самое простое — начать с корневого узла; доступ к нему обеспечивает свойство *documentElement*. Следующий фрагмент кода добавляет к строке *result* свойства *documentElement.nodeName* и *documentElement.text*:

```
result += "objDocument.parseError.errorCode: "
 + objDocument.parseError.errorCode + "\n";

result += "objDocument.documentElement.nodeName: "
 + objDocument.documentElement.nodeName + "\n";

result += "objDocument.documentElement.text: "
 + objDocument.documentElement.text + "\n";
alert (result);
```

Свойства *nodeName* и *text* есть у всех элементных узлов. Объект *objDocument* представляет корневой узел документа, свойство *documentElement* которого обеспечивает доступ к хранящемуся в нем элементу. Поскольку этот элемент используется очень часто, упростим код, создав для него специальную переменную:

```
var rootElem = objDocument.documentElement;
```

Атрибуты элемента хранятся в объекте *XMLDOMElement* в виде именованного набора. По сути набор *attributes* является ассоциативным массивом, элементы которого индексированы по строковому ключу. Например, чтобы получить значение атрибута *verified,* применим метод *getNamedItem*:

```
result += "rootElem.attributes.getNamedItem('verified').nodeValue: "
 + rootElem.attributes.getNamedItem("verified").nodeValue + "\n";
```

Итак, доступ к свойствам не проблема. А как с другими элементами документа, спросите вы? Это чуть сложнее — сейчас я опишу несколько способов прямого доступа к элементам,

Доступ к элементам документа

Сначала загрузим документ посложнее. Воспользуемся документом с текстом анекдота из главы 4.

```
<?xml version="1.0"? encoding="windows-1251"?>
<joke type="story" keywords="гусь бар виноград гвозди">
 <scene number="1">
 Пришел гусь з бар. Подходит к бармену и говорит:
 "У вас есть виноград?" Бармен отвечает: "Это
 бар, здесь нет винограда".
 </scene>
 <scene number="2">
 На следующий день гусь снова пришел в бар. Опять
 Подходит к бармену и говорит: "У вас есть виноград?"
 Бармен отвечает: "Я уже вчера сказал тебе, что винограда
 здесь нет. Если ты еще раз придешь и спросишь
 про виноград, то я приколочу твои перепончатые ноги
 гвоздями прямо к стойке бара!"
 </scene>
 <scene number="3">
 На третий день гусь опять пришел в бар, подошел к
```

```
 бармену и спрашивает: "У вас есть гвозди?" "Нет, конечно", -
 отвечает бармен. Гусь ухмыльнулся и спрашивает: "А виноград?"
 </scene>
</joke>
```

Допустим, что этот документ успешно загружен. Как вы узнали выше, мы можем без труда обратиться к любому элементу и атрибуту любого элементного узла, но как найти элемент — скажем, элементы *scene?* Для этого мы используем свойство *childNodes,* а затем выберем из элемента *scene* нужную нам информацию:

```
result += " rootElem.childNodes.item(1).text: "
 + rootElem.childNodes.item(1).text + "\n"
```

Свойство *Item* возвращает дочерний узел. Помните, что номера элементов набора *childNodes* начинаются с 0, так что в предыдущем примере выводится текст не первой, а второй сцены — той, где бармен, как сказал бы юрист, угрожает нашему водоплавающему герою физической расправой.

Набор дочерних узлов доступен через интерфейс *XMLDOMNodeList* (табл. 5-5).

Табл. 5-5. Свойства и методы интерфейса XMLDOMNodeList.

Свойство/метод	Описание
length	Возвращает число узлов в списке. Длина списка изменяется при добавлении дочерних узлов или атрибутов.
item(индекс)	Возвращает узел с указанным номером; номера отсчитываются от 0.
nextNode	Возвращает узел, следующий за текущим,
reset	Возвращает итератор в исходное состояние (перед первым узлом в списке), так что следующий вызов метода *nextNode* вернет ссылку на первый узел.

Свойство *length* позволяет просмотреть узлы списка по очереди:

```
for (i = 0; i < rootElem.childNodes.length; i++)
 {
  result += " rootElem.childNodes.item(" + i + ").text: "
         + rootElem.childNodes.item(i).text + "\n";
 }
```

Чтобы упростить код предыдущего примера, создадим специальную переменную для списка узлов:

```
var colScenes = rootElem.childNodes;
for (i = 0; i < colScenes.length; i++)
   {
   result += "colScenes.item(" + i + ").text: "
          + colScenes.item(i).text + "\n";
   }
```

Объект DOM *NodeList* переменной *colScenes* содержит список всех прямых потомков object корневого элемента *joke,* что обеспечивает легкий доступ к элементам DOM.

А если нам нужна сцена, атрибут которой содержит конкретное значение? Вот один из способов решения этой задачи:

```
var colScenes = rootElem.childNodes
for (i = 0; i < colScenes.length; i++)
 {
   if (colScenes.item(i).attributes.getNamedItem("number").nodeValue == "1")
   {
     result += "colScenes.item(" + i + ").text: "
            + colScenes.item(i).text + "\n"
   }
 }
```

Это работает, но выглядит слишком топорно. Мы пойдем другим путем. В реализации Microsoft DOM есть два удобных метода: *selectNodes(запрос)* и *selectSingleNode(запрос),* позволяющие очень просто обратиться к конкретному узлу (табл. 5-6).

Табл. 5-6. К элементу или набору элементов можно обратиться с помощью методов selectNodes и selectSingleNode из реализации Microsoft DOM. Аргумент задает шаблон поиска в формате XPath.

Метод	Описание
selectNodes(запрос)	Возвращает список узлов, удовлетворяющих запросу, заданному в аргументе. Поиск ведется от текущего узла. Если запросу не удовлетворяет ни один узел или строка запроса содержит ошибку, возвращается пустой список.

(продолжение)

selectSingleNode(запрос)	Возвращает первый узел списка узлов, удовлетворяющих запросу, заданному в аргументе. Поиск ведется от текущего узла. Если запросу не удовлетворяет ни один узел, возвращается *null*.

Строка-запрос в аргументе методов, перечисленных в этой таблице, представляет собой шаблон XSL. Мы обсудим их в главе 6, а пока используем метод *selectNodes* в качестве альтернативы обращению к списку *childNodes*:

```
var colScenes = rootElem.selectNodes("scene")
for (i = 0; i < colScenes.length; i++)
 {
 result += "colScenes.item(" + i + ").text: "
 + colScenes.item(i).text + "\n"
 }
```

На первый взгляд, главное преимущество метода *selectNodes* — ясность кода. Настоящие достоинства этого метода станут понятны, лишь когда вы начнете работать с вложенными элементами сложных документов. Например, чтобы извлечь список товаров из счета, достаточно применить такой шаблон:

```
selectNodes("/invoice/body/items/item")
```

Чтобы решить эту задачу методом *childNodes,* понадобится немало кода — ведь придется просмотреть список узлов и выбрать нужные. Метод *selectSingleNode* позволяет сократить объем кода до предела:

```
result += rootElem.selectSingleNode("scene[@number='2']").text;
```

В этом примере XSL-шаблон возвращает первую сцену, атрибут (@)*number* которой имеет значение *2*.

Итак, вы познакомились с основными методами для работы с объектной моделью документа — попробуем применить их на практике.

Упражнение: работа с DOM средствами Visual Basic

DOM предоставляет интерфейс к XML-документу, позволяющий работать с содержанием документа. Созданный Microsoft XML-анали-

затор, реализующий спецификацию W3C DOM, является COM-объектом, поэтому вы можете использовать его из любой среды, поддерживающей COM. В частности, с ним можно работать в программе на Microsoft Visual Basic (что мы и сделаем), в коде на JavaScript, вложенном в Web-страницу, в программе на C++ или в Java-приложении.

Мы создадим простую программу на Visual Basic для работы с XML-документом средствами DOM. Наша программа создает экземпляр объекта DOM, загружает в него XML-документ, создает список элементов и просматривает этот список. Все файлы этого упражнения, включая XML-редактор Architag XRay, вы найдете на приложенном к книге компакт-диске. Прежде чем приступать к упражнению, установите редактор, запустив программу установки XRaySetup.exe.

Выполненная Microsoft реализация W3C DOM распространяется в виде библиотеки msxml2.dll. Она регистрируется как COM-сервер, а в своей программе вы будете создавать объект *MSXML2.DOMDocument*.

Для начала откройте документ mvp.xml в редакторе XRay (рис. 5.2). Файл находится в папке \Samples\Ch05\ компакт-диска.

Документ mvp.xml представляет собой правильный XML-документ. В нем около 150 записей, каждая из которых описывает игрока в бейсбол, получавшего приз самого ценного игрока лиги. Для любителей подробностей отмечу, что на самом деле таких призов было три: приз Чалмерса (Chalmers Award), присуждавшийся с 1911 по 1914 год, приз лиги (League Award), 1922-1929, и приз Ассоциации бейсбольных журналистов (Baseball Writer's Award или MVP Award), впервые присужденный в 1931 г. У каждой записи файла есть уникальный идентификатор и 7 элементов, которые содержат информацию об игроке. Наша задача — сосчитать статистические показатели для каждого игрока и вывести в окне наихудший и наилучший показатели по всем игрокам.

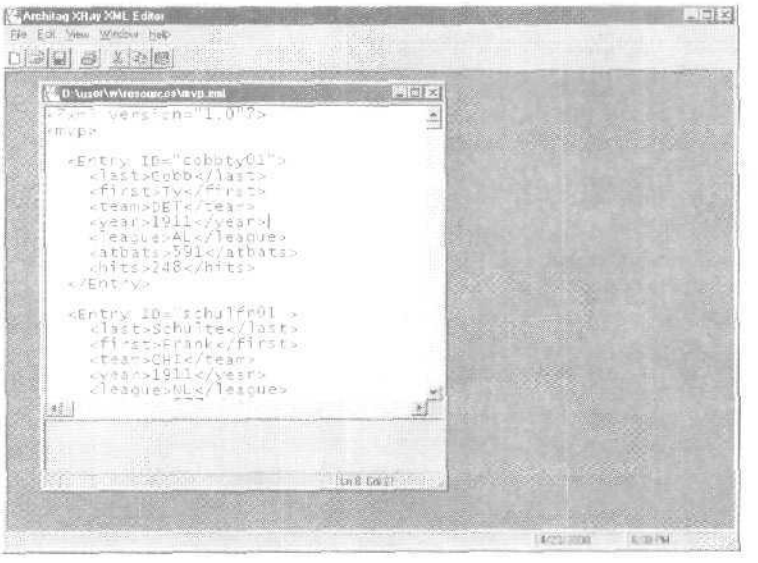


Рис. 5~2, Файл mvp.xml, открытый в редакторе Architag XRay.

1. Запустите Visual Basic (рис. 5-3), выберите проект типа Standard EXE и щелкните кнопку Open.

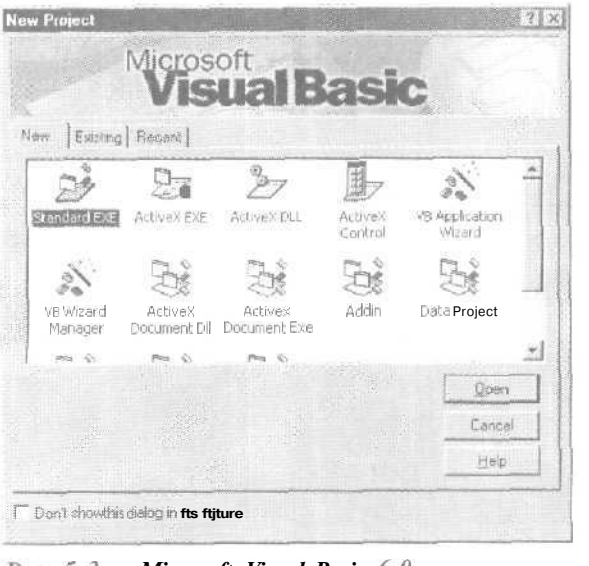


Рис. 5-3. Microsoft Visual Basic 6.0.

2. В окне Project Explorer выберите форму Form1 (рис. 5-4), *а* в меню Project — команду Remove Forml.

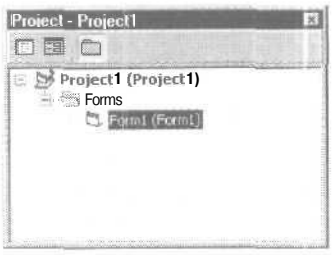


Рис. 5-4. Окно Project Explorer.

3. В меню Project выберите команду Add File и добавьте в проект форму \Samples\Ch05\DOM.frm.
4. Откройте форму, щелкнув значок + в окне Project. Дважды щелкните форму BaseballStats (рис. 5-5).

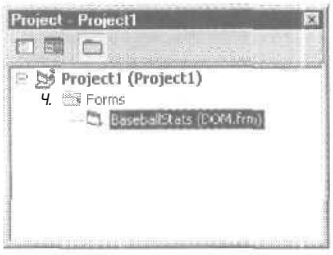


Рис. 5-5. Открытие формы BaseballStats.

5. Сделайте эту форму стартовой, выбрав в меню Project команду Projectl Properties, а затем выбрав форму Choose BaseballStats в меню Startup Object (рис. 5-6). Щелкните кнопку OK.

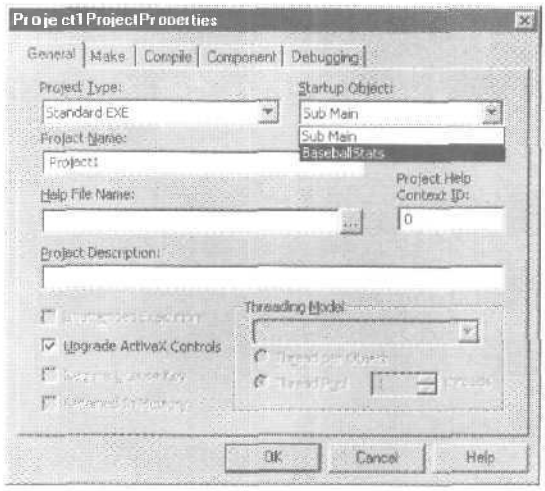


Рис. 5-6. ***Выбор формы BaseballStats в качестве стартовой.***

6. В меню Project выберите команду References. Добавьте в проект библиотеку MSXML, выбрав в списке Available References пункт Microsoft XML, v3.0 (рис. 5-7). Щелкните кнопку OK.

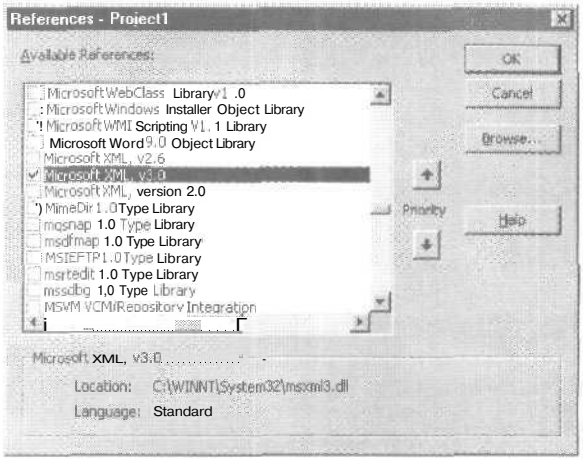
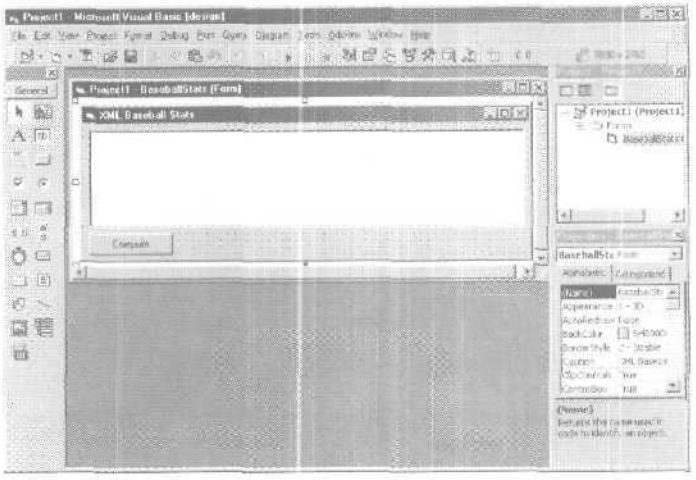


Рис. 5-7. ***Добавление в проект библиотеки анализатора MSXML.***

Теперь окно Visual Basic должна выглядеть как на рис. 5-8.



Рис, *5-8.* ***Форма*** ***BaseballStats.***

7. Дважды щелкните форму справа от кнопки Compute. На экране появится подпрограмма *Form Load.* Замените путь во второй строке на путь к файлу mvp.xml на вашем компьютере:

```
Set oXMLDoc = CreateObject («MSXML2.DOMDocument»)
oXMLDoc.Load («C:\Inetpub\wwwroot\xmlprobz\CH05\mvp.xml»)
```

Обратите внимание на всплывающее окно с доступными свойствами и методами, которое появляется, когда вы набрали текст «oXMLDoc.» (рис. 5-9).

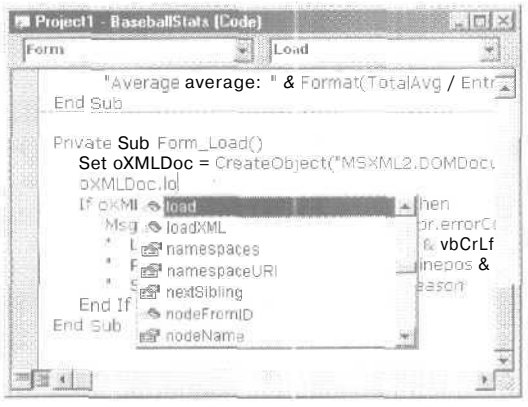


Рис. *5-9.* ***Всплывающее*** *меню со списком свойств и методов объекта.*

Теперь ваша подпрограмма должна выглядеть так (рис. 5-10):

```
Project1 - BaseballStats (Code)
Form                                   Load
    "Average average: " & Format(TotalAvg / Entr
End Sub

Private Sub Form_Load()
    Set oXMLDoc = CreateObject("MSXML2.DOMDoc
    oXMLDoc.load("mvp.xml")
    If oXMLDoc.parseError.errorCode <> 0 Then
        MsgBox "Error: " & oXMLDoc.parseError.errorC
        " Line: " & oXMLDoc.parseError.Line & vbCrLf
        " Position; " & oXMLDoc.parseError.linepos a
        " Source: " & oXMLDoc.parseError.reason
    End If
End Sub
```

Рис. 5-10. Подпрограмма Form_Load.

Первая строка создает экземпляр объекта Microsoft.XMLDOM. Это XML-анализатор, реализованный как COM-сервер, которым вы можете пользоваться в своей программе.

Вторая строка загружает документ в объект.

8. В верхней части окна BaseballStats(Code) есть список форм. Щелкните его и выберите форму Compute. В окне появится подпрограмма (рис. 5-11). Эта подпрограмма вызывается, когда пользователь щелкает кнопку Compute,

```
Project1 - BaseballStats (Code)
Compute                                Click
Private Sub Compute_Click()

    Highest = 0
    Lowest = 1
    POГ i = 0 To Entries.length - 1
        Set Player = Entries.Item(i)
        Hits = Int(Player.selectSingleNode("hits").Text)
        AtBats = Int(Player.selectSingleNode("atbats").Text)
        If AtBats <> 0 Then
            Avg = Hits / AtBats
            TotalAvg = TotalAvg + Avg
            If Avg > Highest Then
                Highest = Avg
                Set HighPlayer = Player
            End If
            If Avg < Lowest Then
                Lowest = Avg
                Set LowPlayer = Player
```

Рис. 5-11. Подпрограмма Compute_Click.

9. Замените строку в начале подпрограммы на:

```
Set Entries = oXMLDoc.selectNodes(«//Entry»)
```

Справочная служба Visual Basic снова приходит вам на помощь (рис. 5-12).

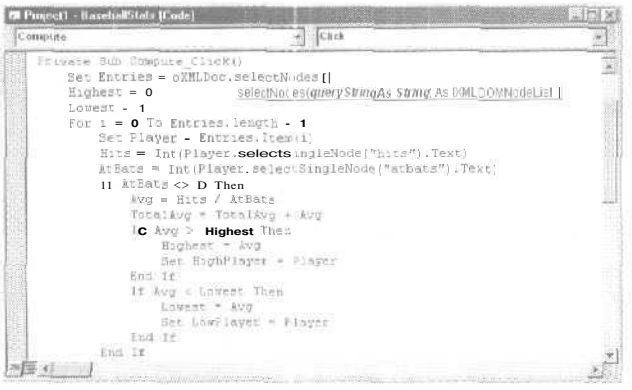


Рис. 5-12. ***Справочная*** *служба* ***VisualBasic подсказывает параметры метода*** *selectNode.*

Подпрограмма *Compute_Click* создает список элементных узлов, который содержит информацию обо всех 154 игроках. Теперь мы можем просмотреть этот список и выяснить, у кого наилучшие и наихудшие результаты.

Когда мы найдем этих игроков, объекты *HighPlayer* и *LowPlayer* будут содержать соответствующие элементы. В конце подпрограммы мы с их помощью выбираем нужную информацию и выводим ее в текстовых полях.

10. Запустите программу, нажав клавишу F5. На экране появится окно (рис. 5-13).

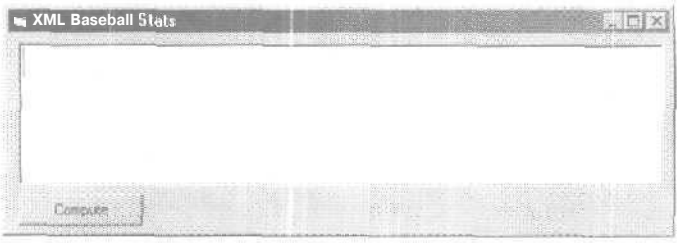


Рис. *5-13. Главное* ***окно*** *программы MVP.*

11. Щелкните копку Compute и вы увидите результаты (рис. 5-14).

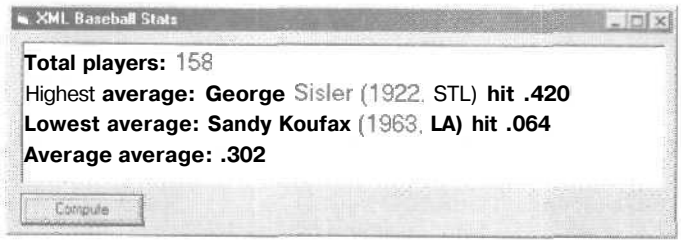


Рис. 5-14. Результаты работы программы MVP.

12. Закройте окно и сохраните проект под именем DOM.vbp.

Другие операции с документом

Не нужно думать, что ваши возможности ограничиваются просмотром элементов документа. Вы можете добавлять узлы, удалять их и выполнять другие операции с документом, загруженным в память. Попробуем создать вот такой XML-документ (листинг 5-1):

Листинг 5-1. Структура XML-документа, который мы будем создавать.

ToyCarPO.xml

```
<?xml version="1.0"?>
<ToyCarPO_1.0
 poNum = "18892"
 poDate="2000-02-21">
 <header>
 <vendorNum>76671</vendorNum>
 <terms>1/10 net 30</terms>
 <shipTo>shipto</shipTo>
 <billTo>billto</billTo>
 </header>
 <body>
 <item needBefore="2000-02-29" needAfter="2000-02-22">
 <partNum>SY120</partNum>
 <qty>10</qty>
 <price>12.43</price>
 </item>
 <item needBefore="2000-03-01" needAfter="2000-02-23">
 <partNum>TQ331</partNum>
```

```
<qty>100</qty>
<price>0.443</price>
</item>
</body>
</ToyCarPO_1.0>
```

Мы создадим каждый элемент, «сцепляя» строки с помощью встроенных функций Visual Basic. Для этого нам, правда, пришлось бы самостоятельно анализировать документ. Это утомительно, так что мы воспользуемся встроенными средствами объекта Microsoft DOM.

Мы создадим пустой документ-счет и с помошью DOM заполним его. Для создания документа воспользуемся Visual Basic. Это простая программа, которая запускается, открывает окно и выводит результаты. Проект см. на компакт-диске в папке \Samples\Ch05\Struct.vbp.

Сначала нам нужно создать экземпляр объекта для хранения XML-документа и загрузить корневой элемент (листинг 5-2, табл. 5-7).

Листинг 5-2. Загрузка кода XML методом loadXML.

buildXML.frm

```
1 Dim oXMLDoc As MSXML2.DOMDocument30
2 Dim rootElement
3
4 Set oXMLDoc = CreateObject( 'MSXML2.DOMDocument")
5 oXMLDoc.loadXML ("<ToyCarPO_1.0></ToyCarPO_1.0>")
6 set rootElement = oXMLDoc.documentElemen
```

Табл. **5-7.** Создание **необходимых объектов.**

Строка	Описание
1–2	Объявление переменных *oXMLDoc* и *rootElementobjects.*
4	Создание экземпляра объекта *oXMLDoc.*
5	Загрузка корневого элемента. В результате создается узел, к которому мы «прицепим» другие узлы.
6	На корневой элемент можно ссылаться, используя конструкцию *oXMLDoc.documentElement,* но проще создать новый объект *rootElement.*

Когда документ будет создан и заполнен, мы сможем добавить элементы и: атрибуты. Сначала мы добавим несколько атрибутов к корневому элементу (листинг 5-3. табл. 5-8).

Листинг **5-3. Добавление атрибутов методом createAttribute.**

```
buildXML.frm
1 Dim namedNodeMap
2 Set namedNodeMap - rootElement.Attributes
3 Dim newAtt
4 Set newAtt = oXMLDoc.createAttribute("poNum")
5 newAtt.Value = "19982"
6 namedNodeMap.setNamedItem newAtt
```

Табл. 5-8. Добавление **атрибутов к корневому** элементу.

Строка	Описание
1—2	**Создание** узла, который будет содержать атрибуты. **Объект** *namedNodeMap* будет содержать объекты-атрибуты, состоящие из пар имя-значение.
3—5	Создание объекта-атрибута и присвоение ему имени. В данном случае создается атрибут *poNum* со значением 19982,
6	Метод *setNamedItem* добавляет новый объект-атрибут в список атрибутов.

Если бы мы сейчас вывели наш XML-документ в текстовом виде, **получилось бы вот что:**

Нам нужно много атрибутов, поэтому **оформим их создание в** виде отдельной **функции:**

```
Public Function addAttribute(oDoc, nodeMap, attrName, attrValue)
 Dim newAtt
 Set newAtt = oDoc.createAttribute(attrName)
 newAtt.Value - attrValue
 nodeMap.setNamedItem newAtt
End Function
```

Функция *addAttribute* **позволяет** создать **атрибут,** передав ей в качестве **параметров имя и значение** атрибута:

```
addAttribute oXMLDoc, namedNodeMap, "poNum", "19982"
addAttribute oXMLDoc, namedNodeMap, "poDate", Now()
```

Теперь наш XML-документ в текстовом виде выглядит так:

Добавим в наш документ элементы. Мы создадим их методом *create Element*, который создает отдельный узел. Затем мы разместим этот узел в дереве документа, применив метод *appendChild*:

```
Dim headerNode
Set headerNode = oXMLDoc.createElement("header")
rootElement.appendChild (headerNode)
```

Наш документ снова изменился:

Созданный элемент нужно заполнить — иначе он останется пустым. Мы добавим к элементу *header* (заголовок) несколько вложенных элементов и заполним их текстовыми значениями.

```
Dim headerNode
Set headerNode = oXMLDoc.createElement("header")
 Dim newChild
 Set newChild = oXMLDoc.createElement("vendorNum")
 newChild.Text = "76671"
 headerNode.appendChild (newChild)
 rootElement.appendChild (headerNode)
```

Объект *newChild* создается точно так же, как элемент *headerNode,* Это элементный узел, которому присваивается имя (*vendorNum*) и значение *(76671).* Метод *appendChild*добавляет этот узел как дочерний к узлу, заданному объектом *headerNode.* Этот узел в свою очередь привязывается к узлу *rootElement*еще одним вызовом метода *appendChild.* Документ теперь выглядит так:

Как и атрибуты, элементы создаются часто, так что эту операцию лучше вынести в отдельную функцию:

```
Public Function addElement(oDoc, node, elementName, elementValue)
 Dim newChild
 Set newChild = oDoc.createElement(elementName)
 newChild.Text = elementValue
 node.appendChild (newChild)
End Function
```

Теперь мы можем вызвать эту функцию для создания всех элементов, вложенных в корневой элемент:

```
addElement oXMLDoc, headerNode, "vendorNum", "76671"
addElement oXMLDoc, headerNode, "terms", "1/10 net 30"
addElement oXMLDoc, headerNode, "shipTo", "Shipto address"
addElement oXMLDoc, headerNode, "billTo", "Billto address"
```

Точно так же мы создаем элемент *body* (тело) и его дочерние элементы (листинг 5-4, табл. 5-9),

Листинг 5-4. Создание тела документа методом createElement.

buildXML.frm

```
 1 Dim bodyNode
 2 Dim itemNode
 3 Set bodyNode = oXMLDoc.createElement( "body")
 4 Set itemNode = oXMLDoc.createElement("item")
 5 ' создание атрибутов
 6 Set namedNodeMap = itemNode.Attributes
 7 addAttribute oXMLDoc, namedNodeMap, "needBefore", Now() + 8
 3 addAttribute oXMLDoc, namedNodeMap, "needAfter", Now() + 14
 9 ' создание вложенных элементов
10 addElement oXMLDoc, itemNode, "partNum", "SY120"
11 addElement oXMLDoc, itemNode, "qty", "10"
12 addElement oXMLDoc, itemNode, "price", "12.43"
13 bodyNode.appendChild (itemNode)
```

```
14
15 Set itemNode = oXMLDoc.createElement("item")
16 ' создание атрибутов
17 Set namedNodeMap = itemNode.Attributes
18 addAttribute oXMLDoc, namedNodeMap, "needBefore", Now() + 7
19 addAttribute oXMLDoc, namedNodeMap, "needAfter", Now() + 21
20 ' создание вложенных элементов
21 addElement oXMLDoc, itemNode, "partNum", "T0331"
22 addElement oXMLDoc, itemNode, "qty", "100"
23 addElement oXMLDoc, itemNode, "price", "0,443"
24 bodyNode.appendChild (itemNode)
25 rootElement.appendChild (bodyNode)
```

Табл. 5-9. Описание кода создания тела документа из листинга 5-4.

Строка	Описание
1—2	Объявление объектов *bodyNode* и *itemNode*. Они будут содержать объекты-элементы.
3—4	Создание узлов и присвоение им имен методом *createElement*.
7—8	Добавление атрибутов. Обратите внимание на создания ассоциативного массива *namedNodeMap* с помощью свойства *Attributes*. Все атрибуты, созданные функцией *addAttnbute*, добавляются в этот массив.
10-12	Функция *addElement* добавляет элементы к элементу *itemNode*.
13	Теперь с помощью метода *appendChild* мы добавляем элемент *item* с его потомками (*partNum*, *qty* и *price*) к узлу *body*.
15—24	Точно так же добавляется еще один элемент *item*.
25	Теперь элемент body добавляется как потомок к корневому элементу методом *appendChild*.

Теперь наш документ готов:

Вот завершенная программа (листинг 5-5):

Листинг 5-5. Программа создания XML-документа на Visual Basic с помощью интерфейса DOM.

buildXML.frm

```
Option Explicit
Public Function addAttribute(oDoc, nodeMap, attrName, attrValue)
 Dim newAtt
 Set newAtt = oDoc.createAttribute(attrName)
 newAtt.Value = attrValue
 nodeMap.setNamedItem newAtt
End Function
Public Function addElement(oDoc, node, elementName, elementValue)
 Dim newChild
 Set newChild = oDoc.createElement(elementName)
 newChild.Text = elementValue
 node.appendChild (newChild)
End Function

Private Sub Form_Load()
 Dim oXMLDoc As MSXML2.DOMDocument30
 Dim rootElement

 Set oXMLDoc = CreateObject("MSXML2.DOMDocument")
 oXMLDoc.loadXML ("<ToyCarPO_1.0></ToyCarPO_1.0>")
 Set rootElement = oXMLDoc.documentElement

 Dim namedNodeMap
 Set namedNodeMap = rootElement.Attributes
 addAttribute oXMLDoc, namedNodeMap, "poNum", "19982"
 addAttribute oXMLDoc, namedNodeMap, "poDate", Now()

 Dim headerNode
 Set headerNode = oXMLDoc.createElement("header")
 addElement oXMLDoc, headerNode, "vendorNum", "76671"
 addElement oXMLDoc, headerNode, "terms", "1/10 net 30"
 addElement oXMLDoc, headerNode, "shipTo", "Shipto address"
 addElement oXMLDoc, headerNode, "billTo", "Billto address"
 rootElement.appendChild (headerNode)
 MsgBox (rootElement.xml)
```

```
 Dim bodyNode
 Dim itemNode
 Set bodyNode = oXMLDoc.createElement("body")
 Set itemNode = oXMLDoc.createElement("item")
 ' создание атрибутов
 Set namedNodeMap = itemNode.Attributes
 addAttribute oXMLDoc, namedNodeMap, "needBefore", Now() + 8
 addAttribute oXMLDoc, namedNodeMap, "needAfter", Now() + 14
 ' создание вложенных элементов
 addElement oXMLDoc. itemNode, "partNum", "SY120"
 addElement oXMLDoc, itemNoce, "qty", "10"
 addElement oXMLDoc, itemNode, "price", "12.43"
 bodyNode.appendChild (itemNode)

 Set itemNode = oXMLDoc.createElement("item")
 ' создание атрибутов
 Set namedNodeMap - itemNode.Attributes
 addAttribute oXMLDoc, namedNodeMap, "needBefore", Now() + 7
 addAttribute oXMLDoc. namedNodeMap, "needAfter", Now() + 21
 ' создание вложенных элементов
 addElement oXMLDoc. itemNode, "partNum", "TQ331"
 addElement oXMLDoc, itemNode, "qty", "100"
 addElement oXMLDoc, itemNode, "price", "0,443"
 bodyNode.appendChild (itemNode)
 rootElement.appendChild (bodyNode)

 MsgBox (rootElement.xml)
 End
End Sub
```

Событийные модели

DOM — компактный простой набор интерфейсов для обработки XML-документов, однако он подходит не для всех задач. Пусть, например, вам нужно обработать большой XML-документ (объемом в десятки мегабайт). Чтобы решить эту задачу средствами DOM, документ придется полностью загружать в память до начала обработки, что неудобно и неэффективно. В таких случаях удобнее событийные интерфейсы, которые в процессе анализа документа передают вызывающему приложению сообщения о возникающих событиях. Для больших документов такой подход гораздо эффективнее.

Другой пример: допустим, вам нужно всего лишь получить несколько первых элементов документа. Загрузка документа целиком будет при этом пустой тратой времени. Событийный интерфейс сообщает приложению о событиях, происшедших при анализе документа — скажем, о том, что анализатор обнаружил нужный вам элемент. Элемент сразу становится доступен независимо от содержимого остатка документа (в частности, от наличия или отсутствия в нем синтаксических ошибок). DOM, напротив, не позволит работать с документом, при анализе которого обнаружена ошибка.

Интерфейсы, реализующие событийный подход, генерируют события, которые перехватываются правилами. Правила обрабатывают элементы — источники события.

Один из широко распространенных интерфейсов этого типа — *упрощенный интерфейс для XML* (Simple API for XML, SAX), разработанный группой XML-DEV при поддержке OASIS. SAX иногда рассматривают как альтернативу объектной модели документа, однако мне так не кажется. В одних случаях оптимальным решением будет DOM, в других — SAX. Недаром Microsoft приступила к реализации SAX в своем XML-анализаторе MSXML[3]. Интерфейс SAX подробно описан на сайте *http://www.megginson.com/SAX/*.

SAX — превосходный интерфейс, однако я предпочитаю сосредоточиться на OmniMark, гораздо более зрелой событийно-ориентированной объектной модели. Среда и язык OmniMark подробно описаны в приложении А.

[3] В третьей версии анализатора MSXML полностью реализована поддержка интерфейса SAX2. *Прим. ред.*

Глава 6

XSL

Расширяемый язык стилей (Extensible Stylesheet Language, XSL) — механизм преобразования и форматирования документов XML. Под преобразованием понимается превращение одного документа XML в другой, часто с изменением структуры. Форматирование — это создание представления документа XML для конкретного устройства отображения (бумага, экран и т. п.). Табл. стилей, или *шаблон* XSL, позволяет, например, отформатировать документ XML в HTML-представление для последующего отображения на экране, а также преобразовывать структуру документов XML. Преобразование чаще всего применяется, когда в организации используют одну схему для документов, а партнер — другую. XSL позволяет преобразовать XML-документ, основанный на одной схеме, в XML-документ, основанный на другой- При этом данные обычно остаются неизменными, но с помощью сценариев можно преобразовать и их.

История XSL

Начнем с небольшого исторического экскурса. Если вы торопитесь, можете без вреда для себя пропустить этот раздел, хотя история развития XSL интересна и поучительна.

Как я уже говорил, Международная организация стандартов в 1986 г. приняла стандарт обобщенного языка разметки SGML. Как и XML, SGML позволяет описывать структуру (но не внешний вид) информации. Вскоре после принятия SGML некоторые члены комитета SGML организовали новый комитет для разработки стандартного языка для описания форматирования SGML-документов. Этот язык был назван *языком описания стиля документа* (Document Style Semantics and Specification Language, DSSSL). Его разработка заняла целых 10 лет, поскольку единый синтаксис для описания всех способов представления документов для разных устройств создать очень трудно.

Спецификация DSSSL была окончательно принята ISO в 1996 г. К несчастью для DSSSL, к этому времени уже появился XML, поэтому большинство программистов решило не торопиться внедрять DSSSL. В результате на сегодняшний день нет ни одной сколь-нибудь широко распространенной реализации DSSSL.

Некоторые члены комитета DSSSL при содействии консорциума W3C сформировали в 1998 г. группу для разработки стандарта форматирования представлений документов XML. В основу стандарта были положены принципы DSSSL для документов SGML. Поскольку группа уже имела десятилетний опыт в этой области и четко понимала задачу, XSL был разработан очень быстро.

Как работает XSL?

У языка XSL два назначения: форматирование объектов и преобразование документов XML. Чтобы отформатировать документ, XSL-шаблон считывает документ XML и, применяя некоторые преобразования, создает другой документ XML — *итоговое дерево* (result tree). Это итоговое дерево наследует пространство имен форматируемого объекта и содержит элементы и атрибуты, описывающие представление XML-документа. Например, итоговое дерево хранит информацию об отображении текстового объекта: начертании (полужирное или курсивное), цвете (красный, синий...), кегле (маленький или большой) и т. д. Кстати, в итоговом дереве абсолютные значения кегля не задаются. Информация этого рода добавляется на следующем этапе — при интерпретации форматируемого объекта.

Итоговое дерево поступает на вход интерпретатора форматируемого объекта, который обрабатывает (интерпретирует) элементы и атрибуты форматируемого объекта и выдает результат в командах используемого устройства вывода (рис. 6-1).

В этом примере дизайнер решил отобразить фрагмент текста зеленым курсивом. Для этого ему достаточно использовать соответствующие тэги. В пространство имен форматируемого объекта в качестве атрибутов шрифта входят начертание и цвет — *italic* (курсив) и *green* (зеленый). Такое описательное форматирование позволяет работать с любым устройством, и разработчик не должен задумываться над кодами для отображения информации на конкретном устройстве.

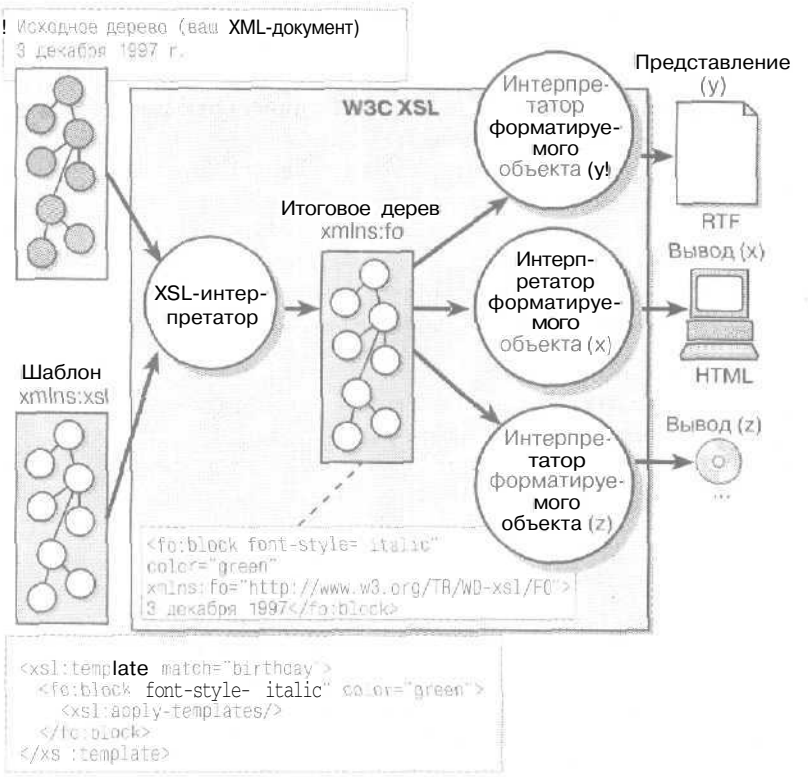


Рис. 6-1. ***Форматирование документа средствами XSL проходит в два этапа: сначала XML-документ преобразуется в итоговое дерево, а затем интерпретатор форматируемого объекта преобразует итоговое дерево в представление для конкретного устройства вывода.***

Посмотрим, как работает XSL на примере создания представления в формате HTML. В HTML начертание и цвет шрифта можно указать прямо в коде с помощью стиля CSS. Интерпретатор форматируемого объекта для HTML преобразует команды отображения текста зеленым курсивом в атрибуты стиля CSS *STYLE=«font-style:italic;color:green»*, смысл которого понятен приложению, отображающему HTML-код (за этим сложным названием скрывается простой Web-обозреватель). Таким образом, окончательный тэг параграфа будет выглядеть так:

```
<P STYLE="font-style:italic;color:green;">3 декабря 1997 г.</P>
```

Теперь допустим, что тот же документ нужно распечатать. Представить документа на бумаге поможет интерпретатор форматируемого объекта, который преобразует его в формат RTF. RTF был разработан компанией Microsoft в середине 80-х гг. для хранения текстовыми редакторами 7-разрядного представления форматированного текста. Поскольку код RTF является текстовым, его можно легко передавать по разным транспортным протоколам — например, по электронной почте. В нашем примере интерпретатор форматируемого объекта преобразует команду *font-style=«italic»* в *\i*, что означает курсив. Команда *color=«green»* преобразуется в *\cб,* что означает выбор 6-го цвета из цветовой таблицы, расположенной в начале документа RTF. В итоге получится следующий фрагмент документа RTF:

```
{\c6\i 3 декабря 1997 г.\par}
```

Создание представлений документов средствами форматирования XSL — мощный инструмент, позволяющий создавать множество представлений одного и того же документа XML с помощью различных таблиц стилей. Безусловно, преимущества этой модели проявляются, лишь когда интерпретатор форматируемого объекта поддерживает необходимое устройство вывода.

XSL в реализации Microsoft

Когда писалась эта книга, спецификацию XSL еще только разрабатывали. Библиотеки форматируемого объекта должны быть достаточно универсальными, поэтому нужно было решить массу сложных вопросов.

В 1998 г. Microsoft решила, что часть XSL, относящаяся к преобразованию документов, уже достаточно стабильна, и реализовала ее согласно проекту спецификации. Впервые эту реализацию XSL использовали в синтаксическом анализаторе MSXML, входящем в Microsoft Internet Explorer 5. Этот шаг предоставил разработчикам доступ к механизму, позволяющему выполнять преобразование XML-XML.

Выделение спецификаций XSLT и XPath

Несмотря на критику реализации XSL компанией Microsoft, которая оказалась частично несовместима со спецификацией консорциума W3C, многие успешно применяли эту реализацию для изучения возможностей языка декларативного преобразования документов. Результатом широкого обсуждения стало, в частности, решение рабочей группы по XSL выделить часть языка, относящуюся к XSL-преобразованию, в отдельную спецификацию XSLT (XSL Transforma-

tions). Теперь преобразование документов возложено на XSLT, а на XSL — все, что связано с форматированием.

XSLT требует синтаксиса, позволяюшего выделять отдельные составляющие элементы XML-документа. Например, для отображения названия главы одним способом, а названия раздела другим, соответствующие правила — их будет два — должны иметь возможность сослаться на объект «название» в текущем контексте. Поскольку XSLT — не единственный стандарт консорциума W3C, которому нужен такой синтаксис, из XSLT была выделена еше одна спецификация — семантики описания путей к элементам документов XML (XML Path Language, XPath). В ноябре 1999 г. консорциум W3C утвердил обе спецификации XPath и XSLT.

Как работает XSLT

XSLT — язык программирования на основе набора правил и обработки событий — отличается от классических процедурных языков. XSLT похож на клерка, в офисе которого множество ящичков. Каждый ящичек для чего-нибудь нужен. На рис. 6-2 в виде отдельного объекта показано одно из правил шаблона, ожидающее наступления события, заданного атрибутом *match*. Получив извещение о событии, такое правило может передать его для обработки другим правилам (название тэга *apply-templates* и означает «применить правила»), а может просто вывести значение с помощью тэга *value-of*.

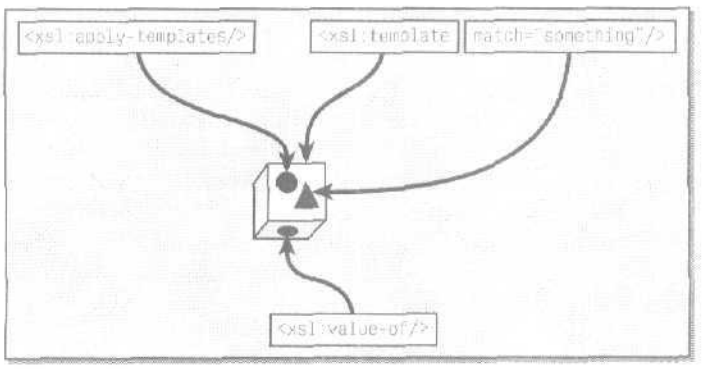


Рис. 6-2. Все детали обработки реализуются правилами шаблона XSLT, что позволяет программисту сосредоточиться на входящем и исходящем потоках.

Когда из входящего потока поступает элемент, клерк по некоторому критерию (обычно это событие, происшедшее во входном XML-документе) помещает результат в нужный ящичек (рис. 6-3).

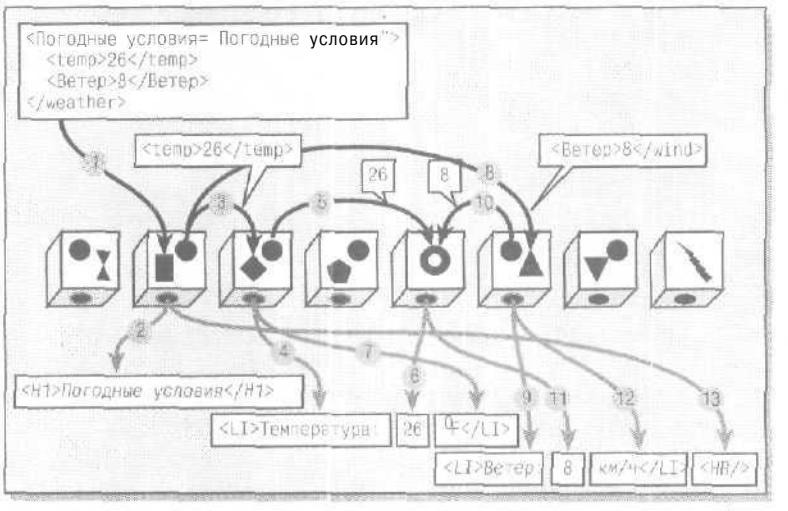


Рис. 6-3. Шаблон XSLT похож на черные ящички, в которых происходит какая-то обработка. В зависимости от типа обработки ящичек может выдать что-нибудь в другой или направить что-нибудь в поток вывода, а может вообще ничего не делать. Подробные комментарии к рисунку см. в табл. 6-1.

Обработка происходит в ящичках, что обычно приводит к выдаче ящичком какого-то результата: статического текста, вычисленного значения или структурного элемента. Результат направляется прямо в поток вывода либо в другие ящички для последующей обработки, Чуть позже я покажу, как это делается средствами преобразований XSL. Сначала мы посмотрим, как процедура, показанная в общих чертах на рис. 6-3, применяется к обработке приведенного ниже простого документа XML Curcon.xml.

```
<?xml version="1.0" encoding="windows-1251"?>
<weather type="Погодные условия">
   <temp>26</temp>
   <wind>8</wind>
</weather>
```

Для преобразования используется Табл. стилей XSL Weathersheet.xsl (листинг 6-1).

Листинг 6-1. Табл. стилей XSL, которая форматирует данные о погоде.

Weathersheet.xsl

```
1  <?xml version="1.0" encoding="windows-1251"?>
2  <xsl:stylesheet
3     version="1.0"
4     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
5
6     <xsl:template match="weather">
7        <H1><xsl:value-of select="@type"/></H1>
8        <xsl:apply-templates/>
9           <HR/>
10    </xsl:template>
11
12    <xsl:template match="temp">
13       <LI>Температура <xsl:apply-templates/>&#x00B0;C</LI>
14    </xsl:template>
15
16    <xsl:template match="wind">
17       <LI>Ветер <xsl:apply-templates/>км/ч</LI>
18    </xsl:template>
19
20    <xsl:template match="text()">
21       <xsl:value-of select="."/>
22    </xsl:template>
23
24 </xsl:stylesheet>
```

Ниже подробно описана процедура обработки (табл. 6-1); шаги пронумерованы, как на рис. 6-3. В столбце «Описание» приведены номера строк из листинга 6-1. Описание поможет понять, как каждый шаг обработки реализуется на языке XSL.

Табл. 6-1. Пошаговое описание преобразования документа.

Шаг	Описание
1	Один из ящичков служит для обработки элемента *weather* документа XML. Поскольку элемент *weather* является корневым, то в яшичек входит он сам и все (это важно!) его потомки. Этот яшичек с прямоугольным отверстием (рис. 6-3) определяется правилами в строках 6–10.
2	Во-первых, наш яшичек выдает данные. С помощью элемента <xsl:value-of/> выбирается текстовое содержимое объекта, заданного атрибутом *select*. В данном случае содержимое атрибута *type* (*@type*) оформляется как заголовок первого уровня (<H1>) кода **HTML.** Обработчик XSL обрабатывает все элементы с префиксом пространства имен *xsl:*. Элементы без префикса *xsl:* передаются в исходящий поток без изменений, Следующая команда <xsl:apply-templates/> передает содержимое элемента *weather* другим ящичкам.
3	Первый дочерний элемент внутри *weather*— <temp>26</temp>. Его должен обработать ящичек с отверстием в виде ромба. Этот ящичек соответствует правилам в строках 12--14.
4	Открывающий тэг элемента HTML <LI> (элемент списка) и текстовая строка «Температура» не содержат префикса пространства имен *xsl:*, поэтому они сразу направляются в поток вывода.
5	Теперь ящичек видит тэг <xsl:apply-templates/> и, как в случае с элементом *weather*, передает содержимое элемента *temp* (т. е. «26») другим правилам для обработки.
6	Ящичек с отверстием в виде кольца просто ожидает данных. Он обрабатывает текст, получаемый от других ящичков. Правила его работы приведены в строках 20-22. Единственный элемент <xsl:value-of/> сразу посылает все входящие текстовые данные (на входе может быть только текстовый элемент) в поток вывода, обозначенный черным кружком.
7	После обработки текста возобновляется выполнение правила «temp», которое выводит закрывающий тэг </LI>. После полной обработки элемента *temperature* управление передается обратно правилу обработки корневого элемента weather, где уже ожидает обработки очередной дочерний элемент.
8–12	Следующий дочерний элемент — *wind* — обрабатывается ящичками аналогично элементу *temperature*.

(продолжение)

13	После обработки всех дочерних элементов элемента *weather* выполнение правила <xsl:apply-templates/> завершается, и выполняются следующие за ним команды. В данном случае выводится горизонтальная черта-разделитель с помощью директивы <HR/>. На этом обработка завершается.

Отдельного внимания заслуживает тэг HTML в строке 9, который рисует горизонтальную разделительную черту. Табл. стилей XSL должна быть правильным XML-документом, т. е. должна следовать правилам оформления документов (см. главу 4). Соответствующий тэг HTML имеет вид <HR>; вспомним, однако, что это пустой элемент. Чтобы быть правильным в смысле XML. он должен сопровождаться закрывающим тэгом; поэтому он и имеет вид <HR/>.

Рекурсия

Замечу, что при наличии тэга <xsI:apply-templates/> после разбора ящичком содержимого очередного элемента результат подается на вход всех имеющихся ящичков-правил, включая ящичек, только что выполнивший разбор. В качестве примера разберем небольшую XSL-программу Generic.xsl (листинг 6-2).

Листинг 6-2. Таблица стилей XSL Generic.xsl для представления любого документа XML в иерархическом виде.

Generic.xsl

```
 1 <?xml version="1.0" encoding="windows-1251"?>
 2 <xsl:stylesheet
 3    version="1.0"
 4    xml:space="default"
 5   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 6
 7    <xsl:template match="node()">
 S       <DIV STYLE="margin-left:12pt;">
 9          [начало <xsl:value-of select="name()"/>]
10          <xsl:apply-templates select="@*"/>
11          <xsl:apply-templates/>
12          [конец <xsl:value-of select="name()"/>]
13       </DIV>
14    </xsl:template>
15
```

```
15     <xsl:template match="@*">
17        [атрибут <xsl: value-of select="name( )"/>:
18        <xsl:value-of select=" ."/>]
19     </xsl:template>
20
21     <xsl:template match="text()">
22        <xsl:value-of select="."/>
23     </xsl:template>
24
25 </xsl:stylesheet>
```

В таблице стилей, приведенной в этом листинге, нет названий элементов и атрибутов. **Все** они обрабатываются анонимно в порядке их появления в документе. Для создания иерархического представления документа нужен механизм, позволяющий выяснить имя и тип каждого узла, что и сделано в таблице стилей. Вот документ **XML** 80112.xml, который мы будем обрабатывать (листинг 6-3):

Листинг 6-3. Документ XML, предназначенный для обработки.

8*0112. ml

```
<?xml version="1.0" encoding="windows-1251"?>
<weather zipcode="80112">
   <location>Иглвуд, штат Колорадо</location>
      <current>
         <condition name="temp">32</condition>
         <condition name="wind chill">20</condition>
         <condition name="wind">южный 7 м/с</condition>
         <condition name="dewpoint">16</condition>
         <condition name="relative humidity">51%</condition>
         <condition name="visibility">10 км</condition>
         <condition name="barometer">630 мм рт.ст.</condition>
         <condition name="sunrise">7:20 MST</condition>
         <condition name="sunset">16:51 MST</condition>
      </current>
   <forecast updated="2000-03-21T15:03:27-07:00">
      <day date="2000-03-22T15:03:28-07:00" high="46" low="21"
         sky="Переменная облачность"/>
      <day date="2000-03-23T15:03:28-07:00" high="51" low="25"
         sky="Переменная облачность"/>
```

```
      <day date="2000-03-24T15:03:28-07:00" high="51" low="19"
         sky="Переменная облачность"/>
      <day date="2000-03-25T15:03:28-07:00" high="53" low="21"
         sky="Переменная облачность"/>
      <day date="2000-03-26T15:03:28-07:00" high="49" low="20"
         sky="Переменная облачность"/>
   </forecast>
</weather>
```

В табл. 6-2 шаг за шагом описана обработка документа из листинга 6-3 с помощью таблицы стилей из листинга 6-2.

Табл. 6-2. Обработка XML-документа 80122.xml с использованием шаблона Generic.xsl.

Строка	Описание
7	Правило *match=«node()»* выберет практически все интересующие нас узлы, и прежде всего узлы-элементы.
8	В первой строке правила *match* (строка 8 таблицы стилей) стоит элемент <D1V>. Этот тэг HTML создает маркированный список. уровень элементов которого описывается отступами; мы сформатируем в этом виде список элементов документа. Поскольку у <DIV> нет объявления пространства имен *xsl:*, то этот элемент с его атрибутами будет просто передан в поток вывода. Атрибут CSS *STYLE* тэга <DIV> задает левое поле элемента списка.
9	У строки символов *[начало* тоже нет префикса пространства имен *xsl:*, поэтому она тоже просто отправляется в поток вывода. Функция *name* в атрибуте *select* элемента <xsl:value-of/> позволяет получить название обрабатываемого узла (точно так же с помощью шаблона *select=«.»* можно получить его значение). После названия элемента выводится символ */*. В результате в потоке вывода окажется строка вида: [начало weather]
10	До сих пор элемент <xsl:apply-templates/> использовался без атрибутов, здесь же появился атрибут *select,* который выбирает и передает все узлы для обработки всем правилам шаблона. В данном случае мы хотим ограничиться обработкой узлов-атрибутов, поэтому использовали атрибут *select=«@*».*
16	Поскольку у первого элемента есть атрибут (*zipcode*), он попадает на вход правила, расположенного в этой строке шаблона,

(продолжение)

Строка	Описание
17-18	Проверка *select=«name()»* возвращает название текущего узла. На вход правила поступил атрибут, поэтому *это* название атрибута. Затем через двоеточие выводится его значение. В результате в поток вывода отправляется строка: [атрибут zipcode: 80112]
11	После обработки атрибутов управление возвращается строке, следующей за отображающим атрибуты элементом *applytemplates*, т. е. строке 11. Здесь находится элемент *apply-templates* без атрибутов, который передает все дочерние элементы на вход всех правил шаблона.
7	Какой сюрприз! Первый дочерний элемент элемента <weather> (погода) — <location> (место) — обрабатывается тем же правилом, что и корневой элемент <weather>.
8	Как и в случае с родительским элементом, вновь используется элемент <DIV> с тем же стилем (отступ 12 пунктов от левого края), Элемент <weather> уже обработан, и уже создан соответствующие ему элемент списка <DIV>. Но предыдущий тэг <DIV> еще не закрыт, поэтому новый тэг <D1V> для элемента <location> вложен в тэг <DIV> для элемента <weather>. Одна из замечательных особенностей иерархических таблиц стилей CSS в том, что они, естественно, иерархические. Поэтому к отступу первого тэга <DIV> в 12 пунктов добавятся 12 пунктов отступа второго тэга <D1V>, так что в результате текст будет сдвинут уже на 24 пункта.
11	Здесь обрабатывается содержимое элемента <location>. Поскольку внутри этого элемента находится обычный текстовый узел, его обрабатывает правило в строке 21, обрабатывающее весь обычный текст.
21	Как и правило, обрабатывавшее узлы, это правило выбирает и обрабатывает текстовое содержимое.
22	Текст передается в поток вывода. После этого управление возвращается строке 11.
12	По завершении обработки содержимого элемента нужно выдать соответствующее сообщение. Эта строка выводит в квадратных скобках название обработанного элемента.
13	Закрывающий тэг </DIV> уменьшает текущий отступ на 12 пунктов.
7-14	Это правило шаблона вызывается рекурсивно для обработки всех элементов файла XML.

В результате получится код HTML, который можно загрузить в любой Web-обозреватель. В окне обозревателя вы увидите файл, созданный методом *transfornNode()* реализации XML DOM компании Microsoft (рис. 6-4). Некоторые другие возможности этого метода будут рассмотрены позже.

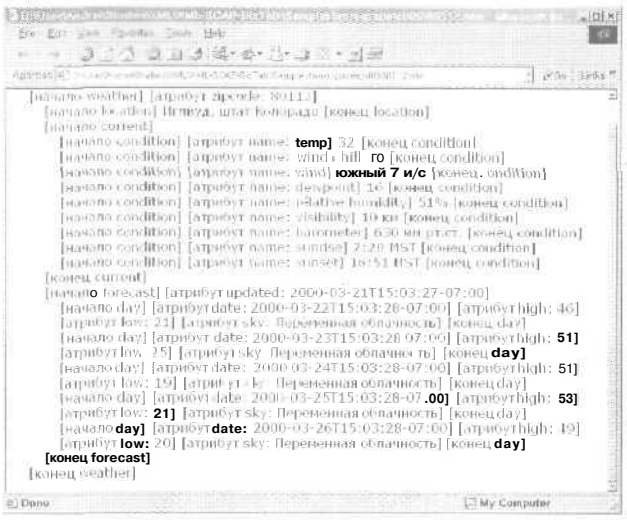


Рис. 6-4. Иерархическое представление документа XML после преобразования с помощью XSL-шаблона Generic.xsl.

Пример использования XSLT

Рассмотрим другие элементы XSLT. Не забывайте о рекурсивной природе XSLT, которую можно представлять себе в виде наполненной ящичками комнаты. Рассмотрим для примера документ XML Duckone.xml (листинг 6-4).

Листинг **6-4. Документ** XML **Duckone.xml, используемый в примере.**

```
Duckone.xml

<?xml version="1.0" encoding="windows-1251"?>
<?xml-stylesheet type="text/xsl" href="joke.xsl"?>
<Jokes xml:space="default">

   <Joke type="one-liner" author="Брайан">
      Приходит гусь в бар и говорит бармену:<quote>
      Налей мне пива и запиши на мой счет</quote>.
```

```
    </Joke>

    <Joke type="story" author="Монти">
        <Scene num="1">У моего пса нет нюха.</Scene>
        <Scene num="2">Как же он тогда нюхает?</Scene>
        <Scene num="3">Приходится нюхать вместо него!</Scene>
    </Joke>

    <Joke type="knock-knock" author="Миллер">
        <Line>Тук-тук.</Line>
        <Line>Кто там?</Line>
        <Line>Сто грамм.</Line>
        <Line>А может еще и косячок?</Line>
        <Line>Тсс. нельзя же так громко.</Line>
    </Joke>

</Jokes>
```

В документе три элемента *Joke*, но с разными атрибутами *type*. Каждый из этих элементов будет обработан по-своему правилами шаблона XSLT Joke.xsl (листинг 6-5).

Листинг 6-5. Шаблон Joke.xsl преобразует документ XML в формат HTML для просмотра в обозревателе.

Joke.xsl

```
1  <?xml version="1.0" encoding="windows-1251"?>
2  <xsl:stylesheet
3   version="1.0"
4   xmlns:msxsl="urn:schemas-microsoft-com:xslt"
5   xmlns:scripts="http://architag.com/scripts"
6     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
7
8     <xsl:output method="html" version="4.0"/>
9
10    <msxsl:script implements-prefix="scripts">
11        <![CDATA[
12        function showXML(nodelist)
13            {
14            txtResult = "<SPAN "+
```

```
15            "STYLE='background-color:lightyellow;" +
16           "cursor:hand;font-size:8pt;'    +
17            "TITLE='"
18        replaceText = nodelist.nextNode().xml
19        replaceText = replaceText.replace(/&/g, '&amp;')
20         replaceText = replaceText.replace(/</g, '&lt;')
21        replaceText = replaceText.replace(/>/g, '&gt;')
22       replaceText = replaceText.replace(/'/g, '&rsquo;')
23        txtResult += replaceText
24        txtResult += "'>(Показать xml)</SPAN>"
25        return txtResult;
26        }
27    ]]>
28 </msxsl:script>
29
30 <xsl:template match="Jokes">
31    <HTML>
32        <HEAD>
33            <TITLE>Пример работы XSL</TITLE>
34            <STYLE>
35           P, UL {margin-top:2pt;margin-bottom;2pt;}
36          H1, H2. H4 {margin-top:2pt;margin-bottom;2pt;}
37            </STYLE>
38        </HEAD>
39        <BODY STYLE="font-family:Verdana;">
40            <xsl:value-of select="scripts:showXML(.)"
41                    disable-output-escaping="yes"/>
42            <H1>Мои любимые шутки</H1>
43                    <DIV STYLE=
44                            "border:1px solid black;padding:3pt;">
45                    <H4>Содержание:</H4>
46                    <xsl:for-each select="Joke">
47                            <A>
48                                    <xsl:attribute
49                                    name="HREF">#<xsl:value-of
50                                    select="generateid()"/>
51                            </xsl:attribute>
52                            <xsl:value-of select="@type"/>
53                            </A><BR/>
```

```
54                    </xsl:for-each>
55            </DIV>
56            <xsl:apply-templates/>
57         </BODY>
58      </HTML>
59
БС   </xsl:template>
61   <xsl:template name="email-author">
52      <xsl:param name="author-alias"/>
63       Автор:  <xsl:element name="A">
64         <xsl:attribute
55            name="HREF">mailto:<xsl:value-of
66            select="$author-alias"/>@jokes.architag.com
67         </xsl:attribute>
58         <xsl:value-of select="$author-alias"/>
69      </xsl:element><BR/>
70   </xsl:template>
71
72   <xsl:template name="set-name">
73      <A>
74         <xsl:attribute name="NAME">
75            <xsl:value-of select="generate-id()"/>
76         </xsl:attribute>
77     </A>
78   </xsl:template>
79
80   <xsl:template match="Joke[@type='story']">
81      <xsl:call-template name="set-name"/>
82      <H2>Смешная история
83      <xsl:value-of select="scripts:showXML(.)"
34         disable-output-escaping="yes"/>
85      </H2>
86      <xsl:call-template name="email-author">
87         <xsl:with-param name="author-alias">
88            <xsl:value-of select="@author"/>
89         </xsl:with-param>
90      </xsl:call-template>
91      <P>
92      <xsl:for-each select="Scene">
```

```
93          <B>Сцена <xsl:value-of select="@num"/>: </B>
94          <xsl:value-of select="."/><BR/>
95       </xsl:for-each>
96       <I><xsl:value-of select="$laughtrack"/></I></P>
97    </xsl:template>
98    <xsl:template match="Joke[@type='knock-knock']">
99       <xsl:call-template name="set-name"/>
100      <H2>Тук-тук
101       <xsl:value-of select="scripts:showXML(.)"
102         disable-output-escaping="yes" />
103      </H2>
104      <xsl:call-template name="email-author">
105         <xsl:with-param name="author-alias">
106            <xsl:value-of select="@author"/>
107         </xsl:with-param>
108      </xsl:call-template>
109      <UL>
110         <xsl:apply-templates/>
111      </UL>
112      <I><xsl:value-of select="$laughtrack"/></I>
113   </xsl:template>
114
115   <xsl:template match="Joke[@type='one-liner']">
116      <xsl:call-template name="set-name"/>
117      <H2>Коротко
118      <xsl:value-of select="scripts:showXML(.)"
119         disable-output-escaping="yes"/>
120      </H2>
121      <xsl:call-template name="email-author">
123         <xsl:with-param name="author-alias">
124            <xsl:value-of select="@author"/>
125         </xsl:with-param>
126      </xsl:call-template>
127      <P><xsl:apply-templates/><BR/>
128       <I><xsl:value-of select="$laughtrack"/></I></P>
129   </xsl:template>
130
131   <xsl:template match="Joke[@type='knock-knock']/Line">
132      <LI><xsl:apply-templates/></LI>
```

```
133   </xsl:template>
134
135   <xsl:template match="quote">
136      "<xsl:apply-templates/>"
137   </xsl:template>
138
139   <xsl:variable name="laughtrack">(смех)</xsl:variable>
140
141</xsl:stylesheet>
```

В табл. 6-3 приводится пошаговое описание преобразования файла Duckone.xml из листинга 6-4 с помощью шаблона из листинга 6-5.

Табл. 6-3. Выполнение преобразования с помощью шаблона Joke.xsl.

Строка	Описание
4-6	XSLT-шаблону нужно три пространства имен. Первое содержит созданные Microsoft расширения языка XSLT. Второе позволяет включить сценарии с помощью элемента *script*. Третье пространство имен требуется обработчику XSLT, чтобы знать, какая версия XSLT применяется. У нас используется рекомендация XSLT, принятая консорциумом W3C в ноябре 1999 г.
8	Элемент *xsl:output* объявляет тип потока вывода (в этом примере — код HTML версии 4.0).
10–28	Сценарий, встроенный в шаблон. Поскольку в спецификацию XSLT не включен сценарный интерфейс, разработчики реализаций XSLT решают проблему встраивания сценариев самостоятельно. Реализация фирмы Microsoft позволяет применять сценарии на нескольких языках, включая JScript. Наш сценарий используется для создания строки HTML, задающей структуру узла XML.
30-59	Синтаксический анализатор считывает XML-документ в дерево объектной модели документа DOM и начинает его обход с корневого элемента. Обработка начнется с элемента *Jokes*, поскольку именно он является корневым. Затем будут последовательно обработаны остальные элементы. Для первого порожденного события будет подобран первый шаблон, соответствующий корневому элементу *Jokes*. Это правило определяет структуру документа HTML, а также стиль CSS.

(*продолжение*)

40 Элемент *xsl:value-of* может содержать вызов сценария. Здесь вызывается функция *ShowXML*, объявленная в строке 12; она размещает содержимое текущего элемента (оно передается ей в аргументе, имеющего значение «.») в атрибуте TITLE тэга SPAN с текстом «Показать XML». Это делается, чтобы создать текстовую строку, «под» которой хранится другой текст, всплывающий при попадании курсора на строку «Показать XML». В данном случае этим текстом будет XML-код элемента,

41 По умолчанию элемент *value-of* заменяет все специальные символы разметки *в* обрабатываемом тексте соответствующими подстановочными строками. Так, если в тексте встретится символ левой угловой скобки (<), он будет заменен подстановочной строкой *<*. Такая замена позволяет избежать интерпретации обычного текста как разметки.

В нашем примере такая замена не нужна (это делается в сценарии), поэтому использован атрибут *disable-output-escaping*, отменяющий режим по умолчанию.

45–54 Этот фрагмент кода создает оглавление документа.

46 Элемент *xsl:for-each* создает набор элементов, описанных в атрибуте *select*. В данном случае создается набор, состоящий из трех узлов элемента *Joke*. Элементы набора в цикле обрабатываются директивами элемента *xsi.for-each*.

47-53 Каждый элемент обрамляется тэгом HTML *<A>*.

48 Элемент *xsl:attribute* добавляет в текущий открытый элемент еще один элемент. В нашем примере это тэг *<A>* и атрибут *HREF* со значением # и ...

49 ... уникальным идентификатором элемента набора. Синтаксический анализатор XML компании Microsoft присваивает локально уникальный идентификатор каждому элементу документа. Он доступен с помощью функции *generate-id*.

52 Здесь выполняется обработка значения атрибута *type* элемента *Joke*. Это значение появится внутри тэга <A>.

53 Обратите внимание на форму HTML-тэга разрыва абзаца (<BR/>). Табл. стилей должна быть правильным документом XML и, следовательно, должна состоять из конструкций, допустимых спецификацией XML. Тэг HTML
 — пустой элемент и прекрасно обрабатывается обозревателем без закрывающего тэта. Однако в правильном документе XML у пустых элементов должен быть особый синтаксис, поэтому в данном случае тэг разрыва превратился в <BR/>.

(продолжение)

Строка	Описание
56	Захваченное элементом *template* с атрибутом *match* содержимое узла с помощью элемента *xsl:apply-template* поступает на вход всех шаблонов таблицы стилей. Это ведет к дальнейшей обработке документа.
57–58	После обработки дочерних элементов будут закрыты тэги *BODY* и *HTML* языка HTML.
61–70	До сих пор остальные правила просто ожидали соответствующего события. Однако можно создать правило, которое работает, как подпрограмма, вызываемая из других правил. У таких правил нет атрибута *match*. Они называются именованными и вызываются по имени. Именованные правила похожи на подпрограммы обычных языков программирования и применяются тогда, когда нужно избежать многократного повторения одного и того же кода в разных местах.
62	Именованные правила способны обрабатывать параметры, передаваемые им вызывающим правилом. В данном примере передается параметр *author-alias*.
66	Этот параметр размещается в атрибуте *HREF* тэга *<A>*.
68	В этом месте значение *author-alias* выводится еще раз.
72–78	Еще одно именованное правило (на сей раз без параметров) называется *set-name*. Оно может быть вызвано откуда угодно, Необходимые для работы этого правила значения вычисляются внутри него.
80	Это правило обрабатывает элементы *joke* лишь в том случае, когда их атрибуту *type* присвоено значение *story* (смешная история). Обратите внимание на XPath-синтаксис строки.
81	Здесь вызывается именованное правило, определенное в строках 72-78.
83	Снова вызывается функция *showXML*, которой передается содержимое текущего элемента «.».
86–90	Элемент *xsl:call-template* вызывает именованные правила. Здесь он применяется для вызова именованного правила *email-author* с параметром *author-alias*; значение параметра равно значению атрибута *author* элемента *Joke*.
92–95	Этот тип элемента *joke* (*story*) состоит из нескольких сцен, которые по очереди обрабатываются элементом *xsl:for-each*.

(продолжение)

93 Строка «Сцена» и атрибут *number* выводятся шрифтом полужирного начертания,

94 Элемент *xsl:value-of* выводит текст сцены, заданной атрибутом *select=«.»*, после чего добавляется правильно оформленный тэг разрыва HTML.

139 В XSLT переменным можно присваивать постоянные значения. Элемент *xsl:variable* присваивает значение переменной *laughtrack*.

96 Элемент *xsl:value-of* может выводить значение объявленных ранее переменных. В данном случае сразу после текста анекдота курсивом выводится значение переменной *laughtrack*.

98 Здесь обрабатываются шутки типа *knock-knock* (тук-тук). Обработка аналогична обработке анекдотов.

115 Аналогично обрабатываются и однострочные шутки *one-liner*.

131–133 Шутки типа *knock-knock* состоят из нескольких строк, поэтому их структура напоминает структуру анекдотов типа *story*. Однако для обработки дочерних элементов шуток типа *story* использовался элемент *xsl:for-each*, а шутки *knock-knock* обрабатывается в ПО строке с помощью вызова правила из строки 131.

131 Обратите внимание на атрибут *match*. Он возвращает элемент *Line*, но лишь когда элемент *Line* является прямым потомком (этот отмечено разделителем /) элемента *Joke* и если у элемента *Joke* имеется атрибут *type* со значением *knock-knock*. Как видите, синтаксис XPath позволяет описывать шаблоны поиска очень подробно.

135–137 Последнее правило обрабатывает двойные кавычки, встретившиеся внутри других кавычек.

Преобразование документа XML из листинга 6-4 с помощью XSLT-шаблона из листинга 6-5 порождает следующий код HTML (листинг 6-6):

Листинг 6-6. **Код HTML, получившийся после преобразования документа из листинга 6-4** с помощью шаблона **из листинга 6-5.**

Joketrans.htm

```
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html;
charset=windows-1251">
```

```
<TITLE>Пример работы XSL</TITLE>
<STYLE>
   P, UL {margin-top:2pt;margin-bottom;2pt;}
   H1, H2, H4 {margin-top:2pt;margin-bottom;2pt;}
</STYLE>
</HEAD>
<BOOY STYLE="font-family:Verdana;">
   <SPAN STYLE='background-color:lightyellow;cursor:hand;
         font-size:8pt;'
      TITLE='&lt;Jokes xml:space="default"&gt;
   &lt;Joke type="one-liner" author="Брайан"&gt;
      Приходит гусь в бар и говорит бармену:&lt;quote&gt;
      Налей мне стаканчик виски и запиши на мой счет&lt;/quote&gt;.
   &lt;/Joke&gt;
   &lt;Joke type="story" author="Монти"&gt;
      &lt;Scene num="1"&gt;У моего пса нет нюха.&lt;/Scene&gt;
      &lt;Scene num="2"&gt;Как же он тогда нюхает?&lt;/Scene&gt;
      &lt;Scene num="3"&gt;Приходится нюхать вместо него!&lt;/Scene&gt;
   &lt;/Joke&gt;
   &lt;Joke type="knock-knock" author="Миллер"&gt;
      &lt;Line&gt;Тук-тук.&lt;/Line&gt;
      &lt;Line&gt;Кто там?&lt;/Line&gt;
      &lt;Line&gt;Сто грамм.&lt;/Line&gt;
      &lt;Line&gt;А может вам еще и косячок?&lt;/Line&gt;
      &lt;Line&gt;Тсс, нельзя же так громко.&lt;/Line&gt;
   &lt;/Joke&gt;
&lt;/Jokes&gt;'>(Показать xml)</SPAN>
<H1>Мои любимые шутки</H1>
<DIV STYLE="border:1px solid black;padding:3pt;">
<H4>Содержание:</H4>
<A HREF="#IDOJNN2">Коротко</A><BR>
<A HREF="#IDwONN2">Смешная история</A><BR>
<A HREF="#IDOVNN2">Тук-тук</A><BR>
</DIV>
<A NAME="IDOJNN2"/>
<H2>Коротко
      <SPAN STYLE='background-color:lightyellow;cursor:
         hand:font-size:8pt;'
         TITLE='&lt;Joke type="one-liner"
         author="Брайан"&gt;
```

```
        Приходит гусь в бар и говорит бармену:&lt;quote&gt;
        Налей мне пива и запиши на мой счет&lt;/quote&gt;.
    &lt;/Joke&gt;'>(Показать xml)</SPAN></H2>
        Автор: <A HREF="mailto:Brian@jokes.architag.com">Брайан</A><BR>
<P>Приходит гусь и бар v говорит бармену:
        "Налей мне пива и запиши на мой счет"
    <BR><I>(смех)</I></P>
<A NAME="IDwONN2"/>
<H2>Смеиная история
        <SPAN STYLE='background-color:lightyellow;
            cursor:hand;font-size:8pt;'
         TITLE='&lt;Joke type="story" author="Монти"&gt;
    &lt;Scene num="1"&gt;У моего пса нет нюха.&lt;/Scene&gt;
    &lt;Scene num="2"&gt;Как же он тогда нюхает?&lt;/Scene&gt;
    &lt;Scene num="3"&gt;Приходится нюхать вместо него!&lt;/Scene&gt;
&lt;/Joke&gt;'>(Показать xml)</SPAN></H2>
        Автор: <A HREF="mailto:Monty@jokes.architag.com">Монти</A><BR>
<P>
<B>Сцена 1: </B>У моего пса нет нюха.<BR><B>Сцена 2: </B>
Как же он тогда нюхает?<BR><B>Сцена 3; </B>Приходится нюхать вместо
него!<BR>
<I>(смех)</I></P>
<A NAME="IDOVNN2"/>
<H2>Тук-тук
        <SPAN STYLE='background-color:lightyellow;cursor:hand;
            font-size:8pt;'
            TITLE='&lt;Joke type="knock-knock"
            author="Миллер"&gt;
    &lt;Line&gt;Тук-тук.&lt;/Line&gt;
    &lt;Line&gt;Кто там?&lt;/Line&gt;
    &lt;Line&gt;Сто грамм.&lt;/Line&gt;
    &lt;Line&gt;А может вам еще и косячок?&lt;/Line&gt;
    &lt;Line&gt;То-с, нельзя же так громко.&lt;/Line&gt;
&lt;/Joke&gt;'>(Показать xml)</SPAN></H2>
        Автор: <A HREF="mailto:Miller@jokes.architag.com">Миллер</A><BR>
<UL>
<LI>Тук-тук.</LI>
<LI>Кто там?</LI>
<LI>Сто грамм.</LI>
<LI>А может вам еще и косячок?</LI>
```

```
<LI>Тц-ц, нельзя же так громко.</LI>
</UL>
<I>(смех)</I>
</BODY>
</HTML>
```

На рис. 6-5 приведен код HTML, получившийся в результате преобразования с помощью редактора XRay. Пример можно найти на прилагаемом к книге компакт-диске в файле \Samples\Ch06\Chapter6.xrw. При его запуске будут загружены документ XML, Табл. стилей XSL, окно преобразования и обозреватель HTML. Следующий раздел также обработан с помощью **XRay.**

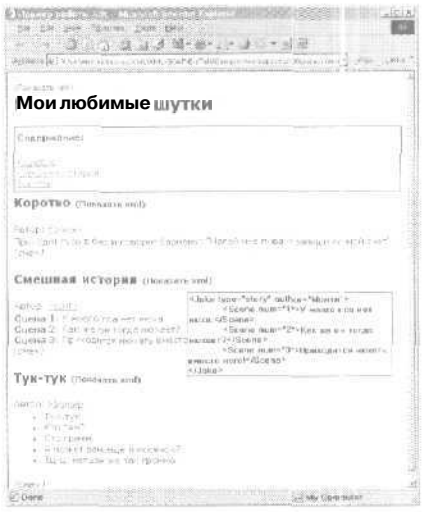


Рис. 6-5. Документ из листинга 6-6 в Web-обозревателе, Всплывающее окно-подсказка с текстом документа XML создано с помощью атрибута TITLE тэга SPAN.

Пример: преобразование бизнес-документа

Сейчас мы используем XSL, для преобразования структуры заказа, точнее, выполним преобразование одной схемы заказа в другую схему. Все файлы этого примера можно найти на прилагаемом компакт-диске в папке \Samples\Ch06\Workshop\. Преобразование выполняется с помощью встроенного XSL-транслятора редактора XRay.

Преобразования такого типа очень распространены. Допустим, компания использует в своей деятельности XML-документы, постро-

енные по определенным схемам. Пусть, скажем, заказы партнеров поступают в таком виде (листинг 6-7):

Листинг 6-7. Документ XML, содержащий описание заказа.

POReq.xml

```
<PurchaseOrder customer="H1553-7" PODate="12-Oct-1999">
   <LineItem>
      <Quantity>100</Quantity>
      <Item>SY120</Item>
   </LineItem>
   <LineItem>
      <Quantity>50</Quantity>
      <Item>SY100</Item>
   </LineItem>
   <LineItem>
      <Quantity>125</Quantity>
      <Item>AR238</Item>
   </LineItem>
   <LineItem>
      <Quantity>100</Quantity>
      <Item>JH877</Item>
   </LineItem>
</PurchaseOrder>
```

Однако один из крупных партнеров уже имеет систему, формирующую заказы в формате XML, но работающую на основе другой схемы. Поэтому заказы этого партнера выглядят так (листинг 6-8):

Листинг 6-8. Пришедший от партнера документ XML с заказом.

POIn.xml

```
<PO date="12-Oct-1999">
 <Customer>H1553-7</Customer>
 <Item qty="100">SY120</Item>
 <Item qty="50">SY100</Item>
 <Item qty="125">AR238</Item>
 <Item qty="100">JH877</Item>
</PO>
```

Для преобразования заказов к нужному виду можно написать программу на C++, BASIC, Perl или еще на каком-нибудь языке, но проще это сделать с помощью XSLT. В листинге 6-9 приведен XSLT-шаблон, *преобразующий* документ XML из листинга 6-8 в документ XML из листинга 6-7.

Листинг 6-9. Шаблон, преобразующий документ XML из листинга 6-8 в документ XML из листинга 6-7.

POTrans.xsl

```
1 <?xml version="1.0"?>
2 <xsl:stylesheet
3  version="1.0"
4  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
5  xml:space="default">
6
7  <xsl:template match="PO">
8    <PurchaseOrder>
9       <xsl:attribute name="customer">
10         <xsl:value-of select="Customer"/>
11      </xsl:attribute>
12      <xsl:attribute name="PODate">
13         <xsl:value-of select="@date"/>
14      </xsl:attribute>
15      <xsl:apply-templates/>
16   </PurchaseOrder>
17 </xsl:template>
18
19 <xsl:template match="Item">
20   <LineItem>
21      <Quantity>
22         <xsl:value-of select="@qty"/>
23      </Quantity>
24      <Item>
25         <xsl:value-of select="text()"/>
26      </Item>
27   </LineItem>
28 </xsl:template>
29
30 <xsl:template match="Customer"/>
```

```
31
32 </xsl:stylesheet>
```

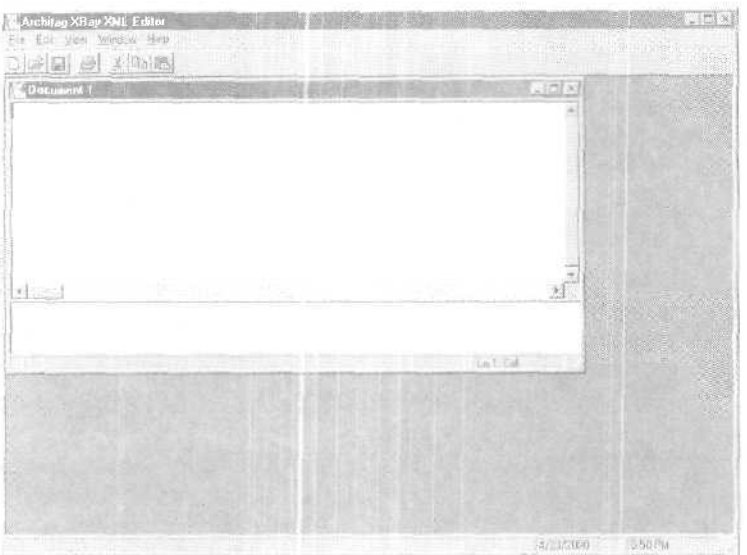
Код шаблона прокомментирован в табл. 6-4.

Табл. 6-4. Комментарий к преобразованию из листинга 6-9.

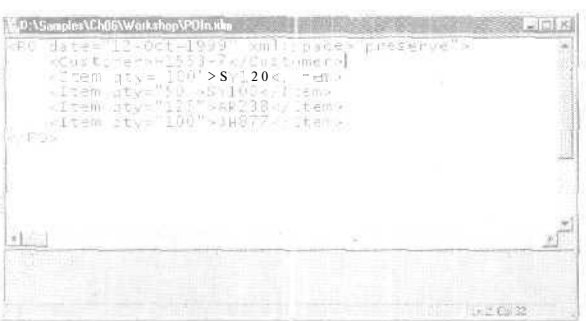
Строка	Описание
7	Из входного потока нужно выбрать два элемента: *PO* и *Item*. Это правило выбирает элемент *PO*.
8	Элемент *PO* входного файла преобразуется в элемент *PurchaseOrder* выходного файла. В этом месте создается открывающий тэг <PurchaseOrder>. Отметим, что XSL-транслятор обрабатывает все фрагменты шаблона, имеюшие префикс *xsl:*. Все остальное копируется в поток вывода без изменения. Поскольку у тэга <PurchaseOrder> нет префикса, он будет записан в файл вывода без изменений.
9-11	У элемента *PurchaseOrder* два атрибута, которые нужно вывести. Элемент *xsl:attribute* добавляет к текущему элементу имя и значение атрибута. В данном случае к элементу *PurchaseOrder* добавляется атрибут *customer co* значением элемента <Customer> потока ввода.
12-14	К элементу *PurchaseOrder* добавляется второй атрибут — *PODate*, которому присваивается значение атрибута *date* входящего документа.
15	Правило *<xsl:apply-templates/>* передает содержимое элемента, обработанного правилом *match=«PO»*, остальным правилам шаблона. В частности, оно будет обработано правилом для элемента <Item>.
16	По окончании обработки элемента *PurchaseOrder* выводится закрывающий тэг.
19	Правило *match=«Item»* предназначено для обработки всех элементов *Item* входного файла. Эти элементы преобразуются в элементы *Line Item*.
20	У элемента *LineItem* два дочерних элемента: *Quantity* и *Item,*
21–23	Элемент *Quantity* содержит значение атрибута *qty* входного документа.
24-26	Элемент *Item* содержит текст входного элемента *Item*.

Теперь разберем по шагам сам процесс преобразования.

1. Запустите XRay командой Stan. Programs, XRay.

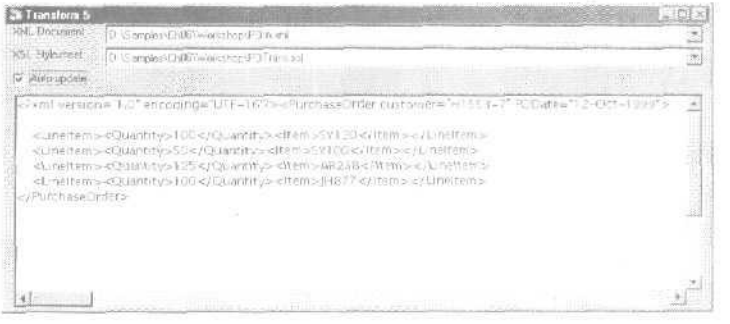


2. Командой File. Open откройте документ Samples\Ch06\Workshop\POIn.xml.



3. Командой File. Open загрузите шаблон Samples\Ch06\Workshop\POTrans.xsl.

4. Откройте новое окно преобразования командой File, New XSL Transform. В списках XML Document и XSL Stylesheet выберем XML-документ POIn.xml и шаблон POTrans.xsl соответственно. В окне появится результат преобразования.



Поэкспериментируйте с правилами шаблона. При этом результат преобразования будет изменяться. Если установить режим Auto-Update, преобразование будет выполняться в автоматическом режиме при любом обращении к окну преобразования. Однако этот режим не прощает ошибок — чтобы преобразование было выполнено, все должно быть безукоризненно правильным. Если что-то не так, преобразование не выполняется, а содержимое окна не меняется.

Средства преобразования XSL (точнее, XSLT) — мощный инструмент, позволяющий изменять структуру документов XML. Один из компонентов сервера BizTalk компании Microsoft, BizTalk Mapper, позволяет создавать XSLT-шаблоны для автоматического преобразования входящих документов в исходящие. В главе 12 мы разберем пример создания шаблона с помощью BizTalk Mapper.

Часть II

Обмен XML-сообщениями

Эта часть посвящена обмену XML-сообщениями между компьютерами. XML-сообщением может быть бизнес-документ, пересылаемый по сети (скажем, счет, заказ или медицинская карта), или запрос к удаленному компьютеру. XML — идеальный синтаксис для обмена сообщениями, поскольку не зависит от платформы и самодокументирован. В этой части вы познакомитесь с двумя стандартами из сферы XML-сообщений: *упрощенным протоколом доступа к объектам* (Simple Object Access Protocol, SOAP) и *BizTalk*, универсальным языком электронного документооборота.

Глава 7

Сетевые службы

Как мы уже знаем, основу Интернета состовляют разработки и стандарты, созданные за последние десять лет. Каждая такая разработка, основанная на успешном применении предыдущей, ведет к сегодняшней возможности написания сетевых программ, использующих преимущества доступа к любым данным и процессам в сети (рис. 7-1).

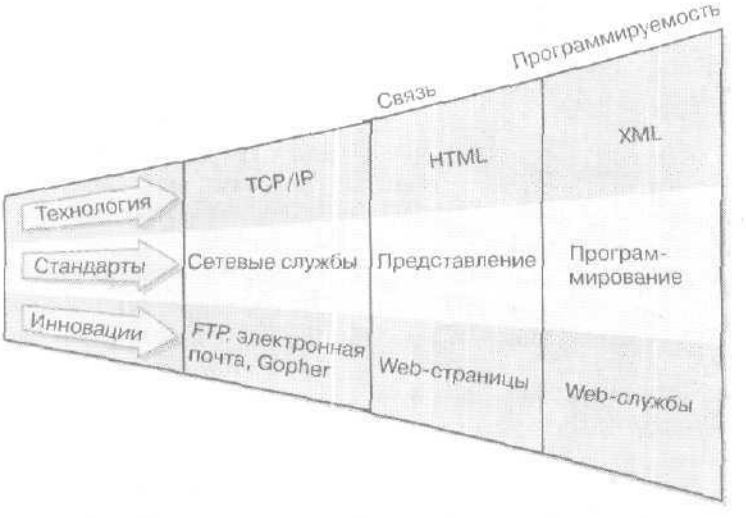


Рис. 7-1. Интернет, который мы видим, возник благодаря нескольким технологическим инновациям.

В 80-е гг. в Министерстве обороны США был разработан TCP/TP (Transmission Control Protocol/Internet Protocol) — простой протокол, обеспечивший связь между локальными сетями путем передачи пакетов через сети общего пользования. Как только стало можно связывать компьютеры, появились и приложения, реализующие преиму-

шества этого подхода. Первые такие технологии — протокол передачи файлов (File Transfer Protocol, FTP) и электронная почта — находятся на вершине стека протоколов TCP/IP и предоставляют нам возможности для связи. Это была революция в бизнесе.

Но, как ни хороши были передача файлов и электронная почта, они не могли обеспечить стандартный метод непротиворечивого представления информации. HTML был разработан как простой метод разметки документов для создания удобного для пользователя представления информации в программе, которую мы сегодня называем обозревателем. Один из тэгов HTML — link — реализовал преимущества новой глобальной сети, позволив переходить в другое место на странице или к новой странице.

Публикация документов в общедоступных сетях революционизировала наш взгляд на информацию, поскольку пользователи получили стандартное представление информации, независимое от платформы, операционной системы или программного обеспечения. Теперь любой мог работать в сети, используя одно-единственное приложение.

HTML — это целостная технология обмена информацией между людьми, но этого явно недостаточно, если мы хотим применить ее для обмена информацией между компьютерами. HTML служит прежде всего для создания двумерного визуального представления информации. Увы, компьютеры пока не имеют глаз, так что эта особенность HTML для них не имеет никакого значения. HTML вынуждает нас форматировать данные прежде всего для отображения на экране, оставляя толкование неоднозначности на усмотрение нашей способности к распознаванию.

Представьте себе агента бюро путешествий, работа которой заключается в том, чтобы понять, что именно надо клиенту, и предложить ему соответствующий набор услуг. Если я попрошу ее предложить мне место, где было бы «дешево, тепло и можно отдохнуть с семьей», ей хватит информации для начала. Она прекрасно понимает, что значит «тепло», потому что это достаточно стандартное понятие, хотя, конечно, она могла бы попросить меня внести ясность: тепло, как в Долине Смерти или как в Орландо? Возможно, мне также придется уточнить понятие «дешево» (чтобы убедиться, что она не ищет нам номер с удобствами во дворе!). Она может поинтересоваться, из кого именно — подростков или малышей — состоит моя «семья», после чего, наконец, сможет приступить к работе.

Прежде всего агент найдет те курорты, где будет тепло во время моего отпуска. Затем она прикинет стоимость авиабилетов, гостиниц, услуг фирмы по прокату автомобилей и добрую дюжину других факторов, влияющих на цену. Для этого агент может использовать брошю-

ры, предоставленные гостиницами или туроператорами, или обратиться на один из Web-сайтов, предлагающих подобную информацию.

Мой агент становится сборщиком информации в поисках набора услуг, удовлетворяющего мои запросы. Она как бы находится в центре информационной вселенной (рис. 7-2).

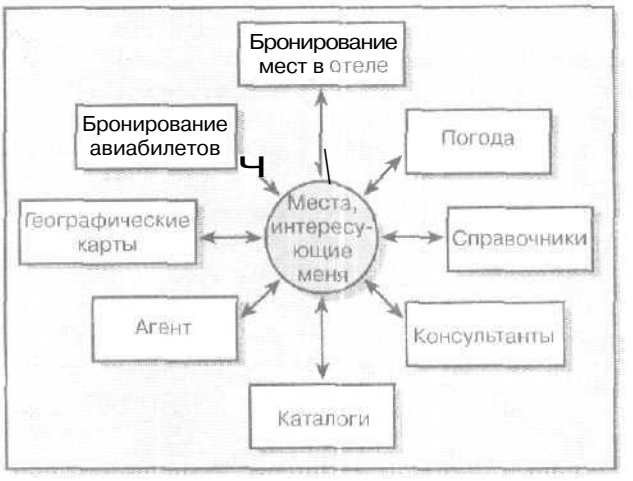


Рис. *7-2. Агент бюро* ***путешествий*** *собирает информацию т различных* ***источников*** *и* ***создает*** *пакет услуг для* ***заказчика.***

После сбора и анализа информации из различных источников мой агент готов к принятию решения. К примеру, температура на острове Большой Кайман может быть 85°F, но отели дороги, а число авиарейсов из Денвера ограничено. На Сент-Томасе — 82°F, но довольно дешево, да и с транспортом лучше. Однако на Сент-Томасе маловато развлечений для детей. На Багамах дороже и не так тепло, но для детей там рай. Иначе говоря, агенту придется искать компромиссы, поскольку удовлетворить все мои требования не удастся. В конце концов она выберет несколько подходящих вариантов и предложит их мне.

Мой агент — человек, и может заниматься одновременно только несколькими делами. Кроме того, существуют и другие заказчики; она не может заниматься целый год моим заказом. С учетом этих ограничений агент может оценить лишь несколько вариантов и выбрать один или два оптимальных. Очень может быть, что где-то существует маленький, теплый остров, чьи дружелюбные жители старательно снижают цены, чтобы привлечь больше туристов. Однако если эта информация не дойдет до моего перегруженного агента, она не сможет учесть ее, а я потеряю возможность прекрасно отдохнуть.

Не правда ли, было бы здорово, если б компьютеры могли повысить эффективность работы агента бюро путешествий, позволяя ей рассмотреть большее число вариантов? Замечательная идея, жаль только, не очень осуществимая. Проблема заключается в формате данных.

Если агент получает данные из Сети, они скорее всего представлены в формате HTML для визуального восприятия человеком при посредстве обозревателя. Многие HTML-страницы, которые просматривает агент, сгенерированы «на лету» при помощи приложений промежуточного уровня. Когда агент обращается на сайт отеля, чтобы проверить наличие свободных номеров, некая программа обращается с запросом к базе данных, содержащей информацию о номерах, а затем создает Web-страницу.

Если бы я писал программу, которой нужен доступ к информации, как моему агенту, у компьютера возникли бы проблемы. Он воспринял бы HTML-страницу как строку символов разметки, а вовсе не как двумерное представление информации. Например, когда вы видите текст «76°F», вы понимаете, что это значит 76° по Фаренгейту, а многие сразу чувствуют приятное тепло. Однако с точки зрения компьютера, это всего лишь четыре ASCII-символа, и он не может интерпретировать их так, как человек.

Возьмем, к примеру, Web-страничку отеля. Мы знаем, что там где-то есть очень ценные данные. Приложение, выполняющееся на сервере, адаптирует данные так, чтобы человек мог их интерпретировать. Неплохо было бы как-нибудь обойти визуальное форматирование и позволить компьютерам обмениваться данными напрямую, не правда ли?

Если б моя программа как-нибудь сообщила сайту отеля, что информация будет интерпретироваться не человеком, а компьютером, ответ мог бы быть гораздо проще для обработки и выглядеть, например, так:

```
<avg-temperature deg="f">76</avg-temperature>
```

Программа, обеспечивающая этот вид межкомпьютерных коммуникаций, называется *сетевой службой* (Web service) и это — следующий важнейший шаг в развитии инфраструктуры Сети.

Существование Сети с миллионами подключенных к ней компьютеров позволило компаниям всего мира размещать информацию так, чтобы она была доступна заказчикам и потенциальным покупателям. Громадный объем данных, которыми манипулируют серверные программы, ожидает запросов от обозревателей.

Если ваша компания обладает ценной информацией, почему бы вам не переформатировать ее и не продавать другим способом? Если

вашей компании нужна информация, не дешевле ли купить ее у кого-нибудь, чем собирать самим?

Думайте о сетевых службах как о приложениях, которые позволяют использовать данные, доступные в Сети, точно так же, как вы используете данные на своем компьютере. Я думаю, что сетевое программирование в ближайшие годы изменит наш способ работы в Интернете.

Значение интеграции

Интеграторы информации были всегда. Агент бюро путешествий — лишь один пример, существует масса других. Независимый страховой брокер собирает информацию у страховых компаний, чтобы найти лучший вариант для заказчика. Ипотечный брокер делает то же самое в отношении заимев. Любой человек, в название профессии которого входит слово «брокер», является интегратором информации.

Подобно упомянутому агенту бюро путешествий, эти интеграторы могут рассматривать только ограниченное количество вариантов просто потому, что люди не способны быстро обрабатывать информацию. Появление Интернета и сетевых служб предоставляет новые возможности интегрирования информации, с которыми человек вряд ли когда-либо смог бы справиться.

Один из примеров новых подходов к интегрированию информации — «обратные аукционы». Согласно этой модели покупатель объявляет о своем желании приобрести что-либо. Поставщики, способные предоставить соответствующую продукцию, конкурируют друг с другом, пока цена не станет минимально возможной. Для сделок большого объема, таких как муниципальные контракты, эта модель уже функционирует. Однако если я хочу купить ящик пива, стоимость проведения аукциона будет гораздо больше, чем возможная экономия.

А если бы я смог использовать компьютеры для автоматизации процесса, стоимость проведения аукциона снизилась бы до долей цента. Если я сэкономлю несколько центов на каждом ящике пива, который покупаю, общая экономия станет значительной. Как добиться резкого снижения стоимости торгов? При помощи открытых стандартов и их умелого применения.

Слабосвязанные службы

Компьютеры взаимодействуют, используя набор интерфейсов. Эти интерфейсы обычно определяются операционной системой и языком программирования, на котором написано приложение. Современные объектно-ориентированные системы позволяют создавать приложе-

ния, которые могут повторно использоваться другими программами. Пример — архитектура Microsoft COM. Обычно COM-компонент — это библиотека динамической компоновки (Dynamic Link Library, DLL). Для программиста такие компоненты — нечто вроде строительных блоков. Вспомните диалоговое окно, которое вы видите каждый раз, когда хотите распечатать документ в текстовом процессоре Microsoft Windows. Программист, написавший текстовый процессор, не писал код этого диалогового окна. Оно построено из компонентов, доступных в ОС Windows и вызываемых текстовым процессором. Именно поэтому диалоговые окна в Windows выглядят, как правило, одинаково.

Объект — это фрагмент кода, выполняющий определенные задачи. Объект автономен, что позволяет различным приложением совместно использовать его. Эта автономность снижает затраты на разработку, поскольку программисты могут не дублировать уже выполненную работу.

Объекты обычно имеют свойства и методы. Свойство — это некий атрибут объекта, который вы можете запросить или установить. Например, можно установить свойства шрифта в диалоговом окне выбора шрифта так, чтобы при открытии окна отображался только список моноширинных шрифтов.

Метод — это функция, которую объект исполняет по запросу. Метод может, например, называться *getStockPrice* или *multiplyTwoNumbers*. В зависимости от метода объект выполняет различные действия для получения результата. Например, для получения цены акций может потребоваться подключиться к базе соответствующих данных. Процесс обработки, необходимый для получения результата, меня не интересует: все, что мне нужно, — сам результат. Эта модель «черного ящика» представляет собой очень важный момент в движении к истинной кросс-платформенности.

Иногда методы требуют передачи параметров. Вам не нужна случайная выборка цен на акции; вы хотите указать конкретную компанию, цена акций которой вас интересует. В этом случае вы можете передать методу в качестве параметра название компании. Метод может предоставить вам возможность передать в качестве параметра дату или время для получения информации за соответствующий период.

На UNIX-платформах реализация повторно используемых объектов работает по тому же принципу, что и COM-объекты. Java-платформы используют технологию Enterprise Java Beans (EJB). Программа, написанная на языке Java, может вызывать EJB для выполнения методов так же, как Windows-программа использует в этих целях вы-

зов метода COM-объекта. Однако поскольку у этих программ разные языковые интерфейсы, они обычно испытывают определенные сложности при вызове друг друга.

Системы, использующие фирменные (патентованные) интерфейсы и объекты, называются *сильно связанными* (tightly coupled), поскольку они полагаются на хорошо описанные, но закрытые интерфейсы. Если связь между приложением и объектом прерывается, «результат может быть непредсказуемым» (цитата из руководства программиста для мэйнфреймов IBM) .

Другая проблема сильно связанных систем заключается в межкомпьютерных коммуникациях. Windows -программа, вызывающая COM-объект для решения какой-то задачи, предполагает, что этот объект находится на локальном компьютере (т. е. там же, где запущена сама программа). Например, Web-сервер может использовать службу, реализованную в виде COM-объекта, которая обращается к локальной БД для получения информации об акциях компании за определенный период. Подобная конфигурация прекрасно работает, пока вызывающая программа и объект исполняются на одном и том же компьютере. Однако если один компьютер предоставляет сведения об акциях, а Web-сервер входит в состав кластера с распределением нагрузки или серверной «фермы», запрашивающая программа уже не находится на том же компьютере. Для решения этой проблемы компания Microsoft разработала *распределенную модель компонентных объектов* (Distributed Component Object Model, DCOM). DCOM позволяет получить доступ к службам COM-сервера через сетевое соединение и обмениваться с ними информацией по тому же соединению. Такая распределенная модель тоже сильно связана, однако менее тесно, чем в случае обыкновенной модели COM, которая требует нахождения объекта и запрашивающего приложения на одном компьютере. Модель DCOM реализована и на других платформах, но получить доступ к службам на других платформах по-прежнему сложно. Например, Windows-приложение не может организовать DCOM-запрос к компоненту EJB, работающему на Linux-платформе на другом конце города, хотя именно выполнения таких транзакций мы и ждем от Web-служб.

Компании во всем мире обладают данными, интересными для других. Однако если попытаться организовать прямой доступ к объекту, генерирующему данные, возникает проблема кросс-платформенной совместимости. Я не могу напрямую обратиться к вашему компоненту EJB из моего приложения, написанного на Visual Basic, а вы не можете напрямую вызвать метод моего COM-объекта из вашей Java-программы. Для организации связи между этими объектными моде-

лями мы должны найти способ абстрагироваться от специфических требований платформ к объектам, создающим представление данных. Таким способом является *простой протокол доступа к объектам* (Simple Object Access Protocol, SOAP). SOAP решает проблемы несовместимости платформ при доступе к данным.

SOAP представляет собой синтаксис для создания приложений, использующих вызовы методов удаленных объектов. SOAP позволяет обойти требования, согласно которым две системы должны исполняться на однотипных платформах или быть написанными на одном языке программирования. Вместо нестандартного двоичного протокола вызова методов, принятого в COM, SOAP использует открытый стандартный синтаксис организации вызова методов. Этот синтаксис — XML. (Вы не удивились?)

Вся информация, пересылаемая между приложением и объектом, представляет собой XML-поток. Этот поток является текстовым и может передаваться по протоколу HTTP через большинство существующих брандмауэров (рис. 7-3).

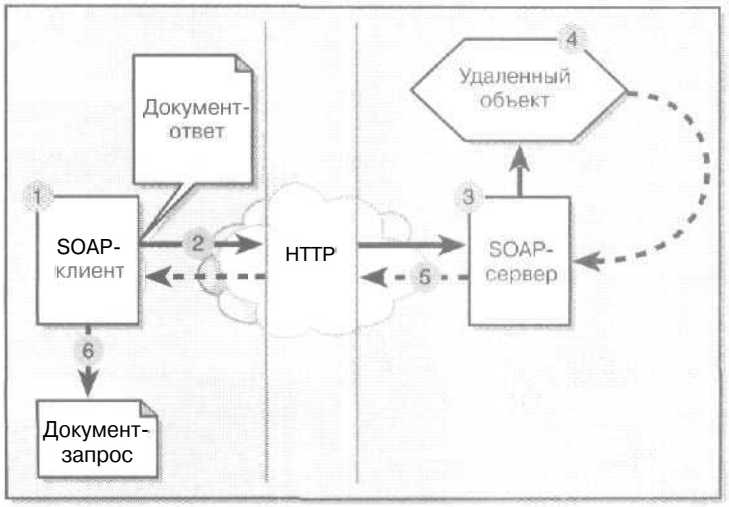


Рис. 7-3. SOAP позволяет** вызывать **методы удаленных** объектов **по HTTP-соединению. Комментарий к схеме см. в табл. 7-1.

Как видите, SOAP-клиент передает документ-запрос по протоколу HTTP «слушающему» SOAP-серверу на другой сайт. Сервер получает сообщение, интерпретирует запрос и вызывает метод локального объекта. Объект возвращает запрошенную нашим приложением информацию, отвечая через SOAP-сервер. Эти шаги пояснены в табл. 7-1 ,

Табл. 7-1. Описание шагов, изображенных на рис. 7-3.

Шаг	Описание
1	Программа, играющая роль SOAP-клиента, создает XML-документ, содержащий информацию, нужную для вызова метода удаленного объекта. Не думайте, что SOAP-клиент обязательно должен быть традиционным. В этой роли может выступать Web-сервер или другое серверное приложение, даже приложение рабочего стола. SOAP-клиент — это любое приложение, запрашивающее SOAP-сервер.
	SOAP-клиент упаковывает XML-документ в SOAP-конверт и преобразует его в POST-запрос протокола HTTP,
2	Этот пакет передается через стандартное HTTP-соединение.
3	«Слушающее» приложение — SOAP-сервер — получает сообщение. Это приложение обычно представляет собой Web-сервер, выбирающий SOAP-запросы из потока запросов HTML-страниц. SOAP-сервер проводит синтаксический анализ запроса и вызывает соответствующий метод нужного объекта, передавая ему параметры, приведенные в XML-документе.
4	Объект выполняет вызванный метод и возвращает результат SOAP-еерверу. Тот преобразует результат в XML-документ и упаковывает его в SOAP-конверт.
5	Ответ возвращается запрашивающему компьютеру. SOAP-конверт снова пересылается в формате HTTP-ответа.
6	Получив ответ, SOAP-клиент «вскрывает» конверт SOAP и пересылает документ запрашивавшему приложению.

Поскольку документы SOAP пересылаются по протоколу HTTP (порт 80), информация может пройти через любой брандмауэр (большинство из них разрешают такой трафик). В результате кросс-платформенные вызовы не требуют изменения конфигурации брандмауэров.

Важно отметить, что вызываемый объект в модификации не нуждается. SOAP-сервер играет роль интерпретатора SOAP-конвертов и вложенных в них документов XML, получаемых через HTTP-соединение, преобразуя их к виду, понятному объекту. SOAP-сервер, иначе говоря, понимает как язык SOAP, так и язык объекта, и может переводить документы с одного языка на другой. Поэтому объект может быть написан на любом языке программирования и работать на любой платформе. **Все** «общение» осуществляется при помощи XML-документов.

Глава 8

Создание Web-службы с помощью SOAP

Простой протокол доступа к объектам (Simple Object Access Protocol, SOAP) построен на основе двух открытых стандартов: HTTP и XML. Синтаксис SOAP очень прост и состоит всего из нескольких элементов и атрибутов, определяющих *конверт*, в который вкладываются передаваемые данные. Элементы SOAP должны иметь префикс пространства имен *SOAP:*.

В этой главе мы разберем пример, состоящий из двух частей. Сначала мы вручную напишем клиент SOAP на JScript, используя объект *XMLHTTP*. Я выбрал этот путь, несмотря на его сложность, чтобы вы во всех подробностях разобрались в работе SOAP-клиента. Объект *XMLHTTP* обеспечивает взаимодействие по протоколу HTTP между приложением — в нашем примере это Web-страница с вложенным сценарием на JScript, которая позволит отправлять на сервер XML-документ, — и другим сайтом. Основное преимущество такого решения состоит в независимости обмена данными по протоколу HTTP от брандмауэров. Это позволяет передавать документы XML так же просто, как и обычные формы HTML.

Во второй части примера мы напишем SOAP-сервер и Web-службу на кросс-платформенном сценарном языке OmniMark. Он может работать не только на Windows-платформах, но и в различных реализациях UNIX, включая Linux. Я использовал OmniMark в этом примере, чтобы продемонстрировать возможность разработки Web-службы на разных языках и на разных платформах.

НИР

Для понимания работы Web-служб, использующих HTTP, сначала нужно разобраться в основах HTTP. Этот довольно простой протокол основан на передаче информации в обычном текстовом виде по любому транспортному протоколу. Соединение по HTTP представляет собой обмен текстовыми сообщениями — *запросами* и *ответами* — между сервером и клиентом. Стандарт HTTP определяет несколько типов запросов; чаще всего используются запросы *GET* и *POST*.

Метод *GET*

Когда пользователь задает в строке обозревателя указатель нужного ему ресурса, обозреватель выполняет запрос *GET*. Например, ввод запроса *http://www.myduckjokes.com/latest.htm* заставляет обозреватель создать пакет HTTP запроса и передать его на IP-адрес соответствующего сервера. Заголовок выглядит примерно так:

```
GET /latest.htm HTTP/1.1
Host: www.myduckjokes.com
Content-Type: text/html
(пустая строка)
```

Пакет HTTP содержит:

- строку инициализации;
- необязательные строки заголовка;
- пустую строку (например, просто символ конца строки);
- необязательное тело сообщения — например, файл, запрашиваемые данные или результат запроса.

В строке инициализации указывается назначение пакета. В данном случае пакет содержит запрос *GET*, который требует от сервера файл */latest.htm*. Строка заканчивается указанием используемой версии протокола HTTP; у нас это 1.1.

Следующие строки заголовка задают параметры запроса в виде пар параметр — значение. Каждая такая строка начинается с названия параметра, затем идут двоеточие, пробел и значение параметра. Параметр заканчивается символом конца строки.

Пакет с запросом HTTP может содержать десятки и даже сотни параметров, включая информацию о типе обозревателя или версии операционной системы, данные из файла-жетона и т. д. Очевидно, что чем больше пакет, тем медленнее он будет пересылаться на сервер. Заголовок заканчивается двумя символами конца строки подряд, Весь остальной текст воспринимается как тело пакета HTTP.

Получив такой пакет, сервер **HTTP** (Web-сервер) пытается найти файл /latest.htm. Если он найден, то после необходимой обработки клиенту будет направлен HTTP-ответ с кодом HTML, Вот пример такого ответа, состоящий из заголовка HTTP, пустой строки и собственно запрашиваемого ресурса (листинг 8-1). Чаще всего запрашиваются документы HTML.

Листинг **8-1. Пакет с ответом HTTP.**

```
HTTP/1.1 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 392
<HTML>
   <HEAD>
      <meta http-equiv="Content-Type" content="text/html;
charset=windows-1251">
      <TITLE>Свежие анекдоты про гусей</TITLE>
   </HEAD>
   <BODY>
      <H1>Свежие анекдоты про гусей</H1>
      <P>Приходит гусь в бар. Бармен ему и говорит:
      "Здесь гусям не место." А гусь отвечает:
      "Да я и сам не очень-то люблю гусей! Налей-ка мне
      лучше пива".</P>
    <P></P>
   </BODY>
</HTML>
```

Обратите внимание на первую строку с информацией о пакете и несколько пар параметр — значение, которые сообщают обозревателю, что делать с этим пакетом.

Первая строка содержит код ответа. Самый распространенный код — 200, означающий, что запрос получен и успешно выполнен. Часто встречаются и код 404 — запрашиваемый ресурс не найден, и код 500 — сообщение о внутренней ошибке сервера. Числовой код предназначен для компьютера, а следующий за ним текст — это сообщение для пользователя. Коды ответа делятся на несколько катэгорий:

- *1xx* — информационное сообщение;
- *2xx* — успешное выполнение запроса;
- *3xx* — переадресация запроса другому серверу;

- *4xx* — ошибка в запросе;
- *5xx* — ошибка сервера.

Все ответы HTTP (кроме ответов с кодами из второй сотни) должны включать дату в заголовке Date:. Дата и время всегда указываются по Гринвичу (Greenwich Mean Time, GMT).

Метод *POST*

Когда пользователь нажмет кнопку Submit формы на странице сайта. информация из формы передается Web-серверу методом *POST*. Для ввода информации в HTML предусмотрены специальные элементы. В качестве примера рассмотрим форму с двумя элементами *INPUT,* Пользователь введет информацию, которая будет передана на сервер методом *POST*. Файл 1040.htm (листинг 8-2) — типичный пример страницы, позволяющей пользователю ввести данные. Этот файл можно найти на прилагаемом компакт-диске в папке /Samples/Ch08.

Листинг 8-2. Документ HTML с элементом FORM.

```
1040.htm
<HTML>
   <HEAD>
      <meta http-equiv="Content-Type" content="text/html;
charset=windows-1251">
      <TITLE>Вычисление подоходного налога</TITLE>
   </HEAD>
   <BODY STYLE="font-family:Verdana;">
      <H1>Вычисление подоходного налога</H1>
      <FORM METHOD="POST" ACTION="/cgi-bin/processForm.pl">
         <TABLE>
            <TR>
               <TD>Введите свой ИНН:</TD>
               <TD><INPUT TYPE="TEXT" NAME="socSecNum"></TD>
            </TR>
            <TR>
               <TD>Укажите ваш годовой доход:</TD>
               <TD><INPUT TYPE="TEXT" NAME="income"></TD>
            </TR>
            <TR>
               <TD>Для вычисления налога нажмите кнопку "Подсчет"</TD>
               <TDXINPUT TYPE="SUBMIT" VALUE="Подсчет">
            </TR>
```

```
        </TABLE>
      </FORM>
    </BODY>
</HTML>
```

Элемент *FORM* может включать такие элементы пользовательского интерфейса, как поля ввода *INPUT*, текстовые поля *TEXTAREA* и кнопки *SUBMIT*. В нашем примере у элемента *FORM* два атрибута: *METHOD* и *ACTION*. *METHOD* указывает метод, используемый для передачи данных на сервер (в данном случае — метод HTTP *POST*). *ACTION* задает указатель ресурса сценария на сервере, который будет обрабатывать полученные данные на Web-сервере. Расширение имени файла .pl указывает, что это сценарий на языке Perl, хранящийся в папке /cgi-bin. Сценарий получит введенную пользователем информацию в виде пар название — значение, которые попадут на сервер в пакете HTTP, и (будем надеяться) сделает с ними что-то полезное.

В обозревателе код из листинга 8-2 отображается так, как это показано на рис. 8-1. Предполагается, что пользователь уже ввел данные.

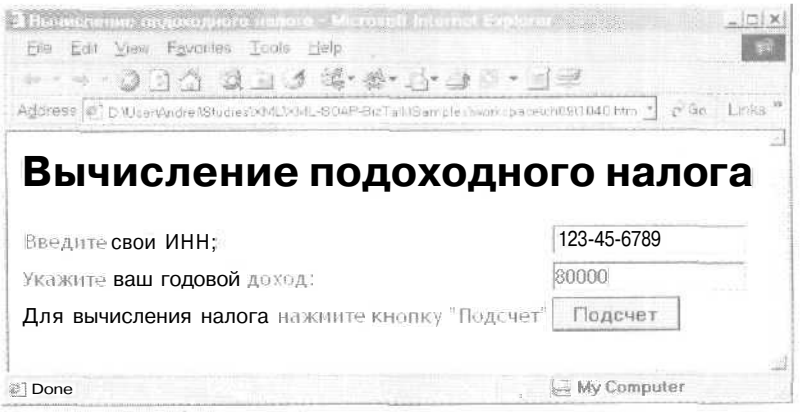


Рис. 8-1. Форма HTML в Web-обозревателе.

Когда пользователь нажмет кнопку «Подсчет», обозреватель создаст и передаст серверу пакет (листинг 8-3).

Листинг **8-3. Пакет HTTP** с информацией, введенной в форме **HTML.**

```
POST /cgi-bin/processForm.pl HTTP/1.1
Host: www.irs.gov
Content-Type: application/x-www-form-urlencoded
```

```
Content-Length: 35
socSecNum=123-45-6789&income=30000
```

Заголовок Content-Type указывает серверу, что пакет представляет собой упакованные данные формы, которые надо распаковать перед передачей обрабатывающему приложению. Тело пакета содержит названия полей ввода и введенные пользователем значения. Пары название — значение разделены символом &.

Получив такой пакет, сервер выделяет пары название — значение из тела пакета и передает их сценарию как аргументы командной строки. Сценарий обрабатывает полученные таким образом данные и передает их серверу, который возвращает результаты пользователю.

Порты

Для обеспечения взаимодействия по протоколу IP серверы всегда отслеживают состояние портов IP. Сервер FTP по умолчанию отслеживает поступление запросов на порт 21. У протокола HTTP порт по умолчанию — 80, поэтому Web-сервер отслеживает поступление на него запросов. Порт по умолчанию можно поменять на другой, но тогда возникнет множество проблем несовместимости, так как приложения считают, что запросы HTTP *GET* и *POST* следует передавать именно на порт 80.

Защищенная версия протокола HTTP — HTTPS — обеспечивает криптографическую защиту канала пользователь — сервер. Если сервер поддерживает HTTPS, то по умолчанию он отслеживает поступление пакетов на порт 443. При обмене данными по HTTPS сервер и обозреватель зашифровывают отправляемые данные и расшифровывают принятые.

Поскольку HTTPS сложнее, чем HTTP, мы не будем останавливаться на нем подробнее. Следует лишь помнить, что соединение между пользователем и сервером можно защитить на уровне протокола. При этом с точки зрения пользователя данные будут передаваться, как обычно.

Другие службы по умолчанию используют другие порты. Например, RealServer компании RealNetworks работает с портами 554 и 7070 (а также и с портом 80, когда нужные порты недоступны), а Telnet использует порт 23. Администратор, настраивая брандмауэры и прокси-серверы, могут ограничивать доступ, блокируя поступление пакетов на соответствующие порты.

Брандмауэры

Я так подробно рассказываю об HTTP потому, что XML-данные передаются по этому протоколу через стандартный порт 80. Поскольку основная часть Web-серверов работает именно через порт 80. большинство брандмауэров разрешает передачу любых данных через этот порт. Обычно пакеты, идущие через этот порт, даже не фильтруются. так как число различных типов данных огромно и постоянно растет, Как следствие, HTTP решает проблемы распределенной модели компонентных объектов (см. главу 7).

Метод *POST*— элегантный путь обмена данными между клиентом и сервером; к тому же он прекрасно вписывается в инфраструктуру Web. Поскольку XML основывается на тех же стандартах кодирования информации, что и HTML, все это делает передачу XML через Интернет тривиальной.

Структура сообщения SOAP

Документ SOAP можно представить в виде электронного конверта, в который вложена информация. Она в свою очередь состоит из последовательности тэгов, описывающих данные и метод, который нужно к ним применить (рис. 8-2).

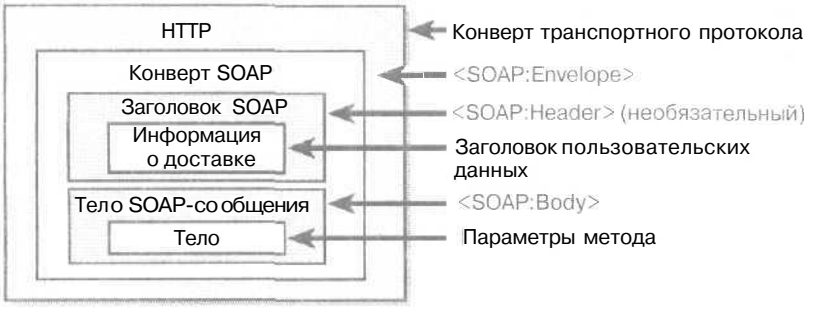


Рис. 8-2. Документ SOAP ~ это конверт, в который вложен запрос, передаваемый SOAP-серверу.

Конверт SOAP включает два элемента: *SOAP:Header* и *SOAP:Body*. *SOAP:Header* содержит информацию о транзакции. Эту информацию задает пользователь в зависимости от того, чего именно он хочет достичь. В приведенном ниже примере элемент *SOAP:Header* содержит элемент *Transaction* со значением 5. Получив эту информацию, серверное приложение выполнит какое-то определенное действие. От-

метим, что элемент *Transaction* находится в заданном пользователем пространстве имен *trans.*

```
<SOAP:Header>
   <trans:Transaction
      xmlns:trans="http://schemas.architag.com/transaction.xsd"
      SOAP:mustUnderstand="1">5</trans:Transaction>
</SOAP:Header>
```

Элемент SOAP может иметь глобальный атрибут *SOAP:mustUnderstand.* Он указывает, обязан ли получатель пакета обработать заголовок SOAP. Атрибут необходим для дальнейшего развития стандарта и может принимать значение 0 или 1. В результате, скажем, можно будет передавать документы следующих версий SOAP серверам, поддерживающим только предыдущие версии SOAP. Элемент *SOAP:Header* не является обязательным.

ПРИМЕЧАНИЕ SOAP отлично работает по протоколу HTTP, однако спецификация SOAP не фиксирует конкретный транспортный протокол.

Второй элемент конверта — тело сообщения *SOAP:Body.* В теле запроса содержатся тэги, описывающие параметры вызываемого метода. В теле ответа элемент *SOAP:Body* применяется для передачи пользователю результатов выполнения запроса. Если сервер не может выполнить запрос, он передает необязательный элемент *SOAP:Fault* с сообщением об ошибке.

Кроме того, протокол SOAP предоставляет широкие возможности для указания типа кодировок, описания массивов и других типов данных. Подробная информация содержится в спецификации протокола (см. приложение Г).

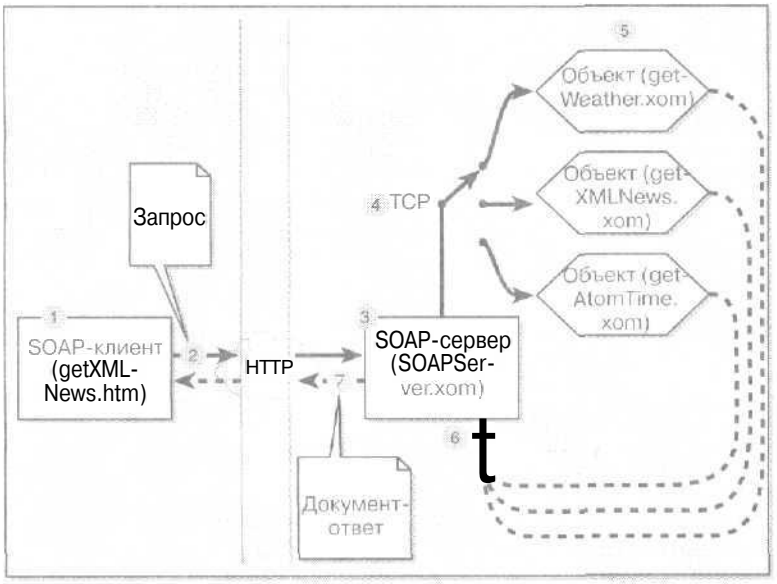
Пример: пишем сервер новостей

В качестве примера разберем работу с сайтом новостей об XML для разработчиков. Такой сайт действительно существует (*http://architag.com/xmlnews.html*). Набрав в обозревателе этот указатель ресурса, мы попадем на страницу ASP, которая связывается с базой данных и выдает ее содержимое пользователю в формате HTML.

Предположим, что нужно обращаться к этой БД, но так. чтобы получать не HTML-страницу, а исходные данные, которые можно будет оформить по своему вкусу. Для этого используем SOAP.

SOAP-клиент

Нам нужен клиент, обращающийся к Web-службе. Мы напишем его на JScript и завернем в HTML-страницу, которую можно открыть в любом обозревателе. На рис. 8-3 показана логика работы клиента; пояснения см. в таблице 8-1. На рис. 8-4 — клиентская страница getXMLNews.htm в окне обозревателя, ее код — в листинге 8-4. Сценарий JScript — в листинге 8-5. Все листинги приведены в папке \Samples\Ch08\ компакт-диска книги.



***Рис,** 8-3, Документ **SOAP** с запросом передается от обозревателя серверу **SOAP,** где он обрабатывается и возвращается в виде документа **SOAP с** ответом. **Этот** процесс прокомментирован в **табл.** 8-1.*

Рис. 8-4. Страница getXMLNews.htm в окне обозревателя.

Листинг 8-4. Код HTML, реализующий пользовательский **интерфейс SOAP** (шаг 1 в **табл. 8-1).**

```
getXMLNews.htm
<HTMLXHEAD>
   <meta http-equiv="Content-Type" content="text/html;
charset=windows-1251">
</HEAD>
<BODY STYLE="font-family:Vercana;">
   <HR>
   <TABLE FRAME="VOID" BORDER="1" RULES="ROWS" WIDTH="100%">
      <TR>
         <TD WIDTH="80" VALIGN="TOP"><B>Дней</B></TD>
         <TD VALIGN="TOP"><INPUT TYPE="TEXT" SIZE="5"
            ID="newsDays" VALUE="23"></TD>
```

```
        </TR>
        <TR>
            <TD WIDTH="80" VALIGN="TOP"><B>Формат</B></TD>
            <TD VALIGN="TOP"><SELECT ID="newsFormat">
                <OPTION VALUE="XML">XML</OPTION>
                <OPTION SELECTED VALUE="HTML3">HTML 3</OPTION>
                <OPTION VALUE="HTML4">HTML 4 - Netscape</OPTION>
                <OPTION VALUE="HTML4">HTML 4 - Explorer</OPTION>
                <OPTION VALUE="xHTML">xHTML</OPTION>
                <OPTION VALUE="Text">Текст</OPTION>
                </SELECT>
            </TD>
        </TR>
        <TR>
            <TD WIDTH="80" VALIGN="TOP"><B>Поля</B></TD>
            <TD VALIGN="TOP">
                <INPUT CHECKED TYPE="CHECKBOX" NAME="newsFields"
                    ID="fldHeadline" VALUE="headline">Заголовок</INPUT>
            <BR><INPUT CHECKED TYPE="CHECKBOX" NAME="newsFields"
                ID="fldLocation" VALUE="location">Местонахождение</INPUT>
            <BRXINPUT CHECKED TYPE="CHECKBOX" NAME="newsFields"
                ID="fldAbstract" VALUE="abstract">Содержание</INPUT>
            <BR><INPUT CHECKED TYPE="CHECKBOX" NAME="newsFields"
                ID="fldURL" VALUE="url">адрес URL</INPUT>
            </TD>
        </TR>
    </TABLE>
    <HR>
    <BR><INPUT TYPE="BUTTON" VALUE="Получить новости"
        ONCLICK="ResultDiv.innerHTML=getXMLNews();">
    <H2>Запрос SOAP</H2>
    <DIV ID="SOAPDiv" STYLE="border:1pt black solid;"> </DIV>
    <H2>ОтветSOAP</H2>
    <DIV ID="ResultDiv" STYLE="border:1pt black solid;"> </DIV>
</BODY>
</HTML>
```

Табл. 8-1. Процесс обработки запроса **80AP** на рис. **8-3.**

Шаг	Описание
1	Создадим клиент SOAP в виде Web-страницы. Такая страница обычно ненамного сложнее приведенного примера. Ее основная цель — формирование запроса SOAP к серверной программе, которая выбирает информацию из различных источников.
2	Документ SOAP передается по протоколу HTTP командой *POST*. В нашем примере в качестве адресата запроса указан локальный сервер (*localhost*).Это значит, что запрос передается на Web-сервер на локальном компьютере.
3	SOAP-сервер получает запрос. Запрос адресован CGI-сценарию, написанному на языке OmniMark. Сервер распаковывает запрос и проверяет, известен ли ему запрошенный метод. Если метол известен...
4	... сервер открывает TCP-соединение с соответствующей службой и перелает документ ей на обработку.
5	Эта служба представляет собой другую программу на языке Omni-Mark, запущенную в окне консоли и ожидающую поступления пакетов на определенный порт. Получив запрос, служба выполняет необходимые действия и передает результат серверу.
6	Сервер создает конверт SOAP и «упаковывает» в него ответ службы.
7	Документ с ответом передается клиенту, ожидающему HTTP-ответ.

Чтобы наша страница заработала, Web-служба должна возвращать несколько новостей из группы новостей об XML. Мы напишем такую службу ниже.

Файл getXMLNews.htm дает пользователям возможность запрашивать информацию с Web-сервера. Пользователь может выбрать формат получаемых данных и задать интервал (в днях) с момента публикации новостей. Наш сценарий на JScript должен включить в запрос выбранные пользователем параметры и список полей, которые следует включить в результирующий документ.

Когда пользователь нажимает кнопку «Получить новости», выполняются следующие действия.

1. Вызывается функция на JScript. создающая документ SOAP с запросом к Web-службе.
2. Документ SOAP отправляется Web-службе методом *POST* протокола HTTP.

3. Служба Web:
   - считывает документ SOAP и выделяет из него собственно запрос;
   - анализирует запрос и выбирает параметры вызова метода;
   - выполняет запрос к БД с соответствующими параметрами;
   - создает пакет с результатом;
   - оформляет пакет в виде конверта SOAP;
   - передает результаты обозревателю пользователя как HTTP-ответ.
4. Функция на JScript получает ответ и преобразует его с помощью шаблона XSLT.

В листинге 8-5 приведен код функции, передающей данные из формы методом *POST* серверу SOAP. Код описан в табл. 8-2. Web-страница целиком показана на рис. 8-4.

Листинг 8-5. Сценарий JScript, передающий XML-документ серверу SOAP.

```
1  <SCRIPT LANGUAGE="JScript">
2     var xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
3
4     var objStyle = new ActiveXObject("MSXML2.DOMDocument");
5     objStyle.async = false;
6     objStyle.load("ShowXML.xsl")
7
8     var SOAPRequest = new ActiveXObject("MSXML2.DOMDocument");
9     SOAPRequest.async = false;
10
11    var SOAPResponse = new ActiveXObject("MSXML2.DOMDocument");
12    SOAPResponse.async = false;
13
14    function getXMLNews()
15       {
16       xmlhttp.Open("POST",
17          "http://architag.com/scripts/SOAPServer.xar", false);
18       strXML =
19       "<SOAP:Envelope " +
20       "  xmlns:SOAP='urn:schemas-xmlsoap-org:soap.v1'>" +
21       "  <SOAP:Body>" +
22       "  <getXMLNewsRequest" +
23       "  xmlns='urn:schemas-architag-com:getXMLNews'>" +
24             <days>" + document.all.newsDays.value + "</days>" +
25             <format>" + document.all.newsFormat.value + "</
```

```
format>" +
26          "       <fields>"
2?        if (fldHeadline.checked) strXML += "headline "
23        if (fldLocation.checked) strXML += "location "
29        if (fldAbstract.checked) strXML += "abstract "
30        if (fldURL.checked) strXML += "url "
31        strXML += "                    </fields>" +
32              </getXMLNewsRequest>" +
33        "   </SOAP:Body>" +
34        "</SOAP:Envelope>"
35
36        SOAPRequest.loadXML(strXML)
37        SOAPDiv.innerHTML =
38            SOAPRequest.transformNode(objStyle.documentElement)
39
xmlhttp.setRequestHeader("SOAPMethodName", "getXMLNewsRequest")
40        xmlhttp.Send(SOAPRequest.xml);
41       SOAPResponse, loadXML(xmlhttp.responseXML.xml)
42        return SOAPResponse.transformNode(objStyle.documentElement)
43        }
44 </SCRIPT>
```

Табл. 8-2. Пошаговое описание кода JScript из листинга 8-5.

Строка	Описание
1	Начало сценария.
2	С помощью конструктора *new ActiveXObject* создается объект *XMLHTTP.*
4	Нам понадобится несколько экземпляров анализатора XML. Сначала создается экземпляр с нашим шаблоном XSLT.
5	Свойство *async* указывает синтаксическому анализатору, как поступить с загружаемым документом. Если значение этого свойства равно *false,* синтаксический анализатор начнет анализ документа только после его загрузки. Значение *true* разрешает начать обработку в фоновом режиме, не дожидаясь конца загрузки. При этом возрастает сложность обрабатывающей программы, поскольку придется создать дополнительную функцию, которая будет выполнена по окончанию загрузки документа. Для простоты в нашем примере атрибуту *async* присвоено значение *false.*

(*продолжение*)

6	Метод *load* загружает документ XML, расположенный по указателю, заданному *к* параметре. В нашем примере это шаблон XSLT из текущего каталога.
8–9	Вторая копия анализатора XML нужна для хранения документа с запросом SOAP.
11–12	Третья копия анализатора XML нужна для хранения документа с ответом сервера SOAP,
14	Функция *getXMLNews* будет вызываться в коде Web-страницы.
16–17	Метод *Open* объекта *XMLHTTP* создает соединение с сервером. В качестве сервера выступает программа на языке Omni Mark, работающая на локальном сервере. Второй параметр метода *Open* аналогичен свойству *async* синтаксического анализатора XML. Ему присвоено значение *false*, т. с. программа будет ждать полной загрузки ответа.
18–34	Строка *strXML* будет содержать документ XML в текстовом виде, включая значения, выбранные пользователем на Web-странице. Такая техника создания XML-документов в коде называется *сцеплением* (XML by concatenation). Документ XML можно создать и с помощью API DOM. добавляя по очереди узлы с элементами и атрибутами. Этот метод корректнее с теоретической точки зрения, но и учебных целях я применил сцепление строк.
36	Созданный XML-документ загружается в анализатор. Теперь мы можем манипулировать документом с помощью свойств и методов API DOM.
37–38	Здесь создается фрагмент пользовательского интерфейса. Только что созданный документ XML с помошью XSLT преобразуется в HTML и выводится на странице. В тесте HTML использован элемент DIV с атрибутом *ID*, имеющим значение *SOAPDiv*. Метод *transformNode* объекта *SOAPRequest* вызывает XSLT-процессор, который создает объект HTML. Этот объект отображается на странице с помощью присвоения значения свойству *innerHTML* объекта *SOAPDiv*.
39	Спецификация SOAP рекомендует указывать параметр *SOAPMethodName* в заголовке HTTP-запроса. Значение этого параметра — название вызываемого метода.

(продолжение)

Строка	**Описание**
40	Метод *Send* перелает пакет серверу SOAP, соединение с которым установлено в строке 16. Помните: объект XML *SOAPRequest* хранится в памяти в двоичном представлении. С помощью свойства *.xml* мы превращаем его текстовый XML-поток. Метод *Send* сначала создает заголовок HTTP с параметром *SOAPMethodName*, затем добавляет пустую строку (она должна отделять заголовок от тела), а после нее — собственно документ XML.
41	Ждем ответа. Поскольку свойству *async* присвоено значение *false,* после строки 40 выполнение программы приостанавливается до поступления ответа сервера. Он будет иметь вид обычного документа XML в виде последовательности символов Unicode. Поскольку к документу нужно обращаться из сценария, этот текстовый поток нужно будет преобразовать в объект XM L DOM с древовидной структурой. Именно это и делает метод *LoadXML*.
42	Ответ сервера нужно представить в формате HTML. После этого документ можно возвратить создавшей запрос функции в качестве результата. Для этого в нашем примере функция устанавливает значение *innerHTML* другого элемента *DIV* на странице.

Пишем SOAP-сервер на языке OmniMark

Сервер SOAP, соединение с которым устанавливается в строке 16 листинга 8-5, представляет собой программу на языке OmniMark. Она в свою очередь вызывает другую программу на OmniMark — SOAP-Server.xom, которая играет роль HTTP-сервера. Прежде чем разбираться в программах, рассмотрим OmniMark подробнее.

OmniMark — событийно-ориентированный язык программирования четвертого поколения. Он может работать на большинстве современных платформ. Язык появился в начале 90-х гг. как средство для преобразования данных. С тех пор его возможности значительно расширены, и теперь это реальная альтернатива языку Perl в области создания профессиональных сетевых приложений.

Довольно простой синтаксис OmniMark напоминает обычный английский язык. В него встроены средства синтаксического анализа и обработки XML, а также лучший в мире процессор поиска и замены по шаблону.

OmniMark поставляется бесплатно. На прилагаемом к книге компакт-диске он хранится в папке \OmniMark. Там же вы найдете и прекрасную интегрированную среду разработки для этого языка. В отличие от языка среда бесплатна для домашнего использования и обучения. Для использования ее на работе придется заплатить несколько сотен долларов.

Вместе с OmniMark поставляется подробная документация, упрощающая обучение и работу с ним. На компакт-диске приведен первый том моей книги «OmniMark в действии. Введение в язык». Я писал ее до появления последней версии OmniMark, поэтому упомянутые там средства сетевого программирования ограничены. Однако в книге подробно изложены основы архитектуры OmniMark, типы данных, управляющие структуры и методы работы. Более современное описание языка см. в приложении А.

В этой главе мы рассмотрим две программы на OmniMark. Первая - CGI-программа SOAPServer.xom — вызывается из нашего сценария, анализирует сообщение SOAP и выясняет, какая из запущенных служб должна обработать поступивший запрос. Вторая — SOAP.xin — работает в качестве службы, отслеживая определенный IP-порт и активизируется, когда на этот порт поступает сообщение нашей CGI-программы в виде потока данных XML. После обработки запроса и создания ответа служба направляет его CGI-программе.

В приведенном примере документ XML передается между программой на OmniMark. выполняющей роль сервера SOAP, и обработчиком OmniMark. который работает в качестве Web-службы. Однако важно отметить, что Web-службы могут ничего не знать об XML. Задача SOAP-сервера как раз и состоит в преобразовании запроса SOAP в вид, удобный для службы. Это означает, что службы можно писать на чем угодно, хоть на языке Кобол. В этом случае сервер SOAP просто должен будет преобразовать запрос SOAP в форму, понятную для программы на Коболе, а затем преобразовать ответ этой программы в документ SOAP.

Программа SOAPServer.xom

SOAPServer.xom представляет собой CGI-программу и вызывается так же, как программы на Perl. Ее можно активизировать, задав указатель URL, или, как это сделано в примере, обратившись к ней из файла с аргументами. В данном случае файл с аргументами называется SOAPServer.xar (листинг 8-6). Этот метод вызова удобен тем, что позволяет задать любую строку с аргументами и запустить программу на OmniMark в произвольном каталоге по умолчанию (например, на другом компьютере).

У OmniMark развитый интерфейс командной строки. Файл SOAP-Server.xar содержит ключи запуска программы из командной строки. В первой строке указан исходный файл (ключ *-sb)* SOAPServer.xom. Ключ *-г* во второй строке задает путь к импортируемым файлам, а ключ *-х в* третьей строке указывает путь к библиотекам. Все сообщения об ошибках, а также стандартный поток #error перенаправляются в файл, заданный в четвертой строке с помощью ключа *-log.*

Листинг 8-6. Файл для вызова сервера 50АР, написанного на языке OmniMark.

```
SOAPServer.xar
-sb SOAPServer.xom
-i "c:\program files\omnimark\xin\"
-x "c:\program files\omnimark\lib\=L.dll"
-log SOAPServer.log
```

OmniMark поддерживает описание вызова исполняющей системы в псевдокомментарии в первой строке файла (наподобие Perl). Эта строка обычно выглядит примерно так:

```
#!/usr/bin/omnimark/bin/omnimark -sb helloworld.xom
```

Текст CGI-программы SOAPServer.xom (листинг 8-7) описан в табл. 8-3.

Листинг 8-7. SOAP-сервер на OmniMark в виде приложения CGI.

```
SOAPServer.xom
1   include "SOAP.xin"
2
3  process
4      local stream methodName? initial {""}
5      local stream SOAPResponse
6      local stream formVars variable initial-size 0
7      local stream remote-host initial {"localhost"}
8      local TCPConnection myConnection
9      local stream myRequest
10     local stream responseXML
11
12     cgiGetEnv into formVars
13     set methodName to UTIL_GetEnv ("HTTP_SOAPMETHODNAME")
14
```

```
15    open SOAPResponse as buffer
16    do scan #main-input
17       match any{formVars{"CONTENT_LENGTH"}}
18              => inputData
19          do when methodName = ""
20            put SOAPResponse "%n<SOAP:Fault>"
21                      || "%n  <faultcode>101</faultcode>"
22                      || "%n  <faultstring>запрошен неизвестный
                        метод"
23                      || "%n</faultstring>"
24                      || "%n</SOAP:Fault>"
25          else
26              do when portSOAP has key methodName
27                      set myConnection to TCPConnectionOpen
28                                      on remote-host
29                                      at portSOAP {methodName}
30                      open myRequest as TCPConnectionGetOutput
31                      myConnection
32                      put myRequest inputData || crlf
33                      close myRequest
34
35                      do when TCPConnectionIsConnected myConnection
36                              set responseXML to TCPConnectionGet
37                              Characters myConnection
38                              put SOAPResponse responseXML
39                      done
40            else
41                      put SOAPResponse "<SOAP:Fault>"
42                              || "    <faultcode>101</faultcode>"
43                              || "    <faultstring>запрошенный
                              метод не поддерживается:"
44                              || " %g(methodName)</faultstring>"
45                              N "</SOAP:Fault>"
46              done
47          done
48    done
49    close SOAPResponse
50
51    output "Content-Type: text/xml" || crlf
```

```
52        || "Cache-control: private"
53        || crlf || crlf
54        || "<?xml version='1.0'?>"
55        || "%n<SOAP:Envelope "
56        || "%n xmlns:SOAP='urn:schemas-xmlsoap-org:soap.v1'>"
57        || "%n <SOAP:Body>"
58        || SOAPResponse
59        || "%n </SOAP:Body>%n</SOAP:Envelope>%n"
50
61    catch #external-exception
52          identity catch-it
53          message catch-msg
64          location catch-loc
65      output 'В SOAPServer произошло внешнее исключение%n'
66          || '%g(catch-loc)%n'
37          | '%g(catch-id) : %g(catch-msg)%n'
```

Табл. 8-3. Описание кода сервера **SOAP из листинга 8-7.**

Строка	**Описание**
I	Команда *include* включает файл SOAP.xin в текст программы. Файл SOAP.xin определяет константы и функции, которые будут использованы программой и Web-службой, к которой будет обращаться программа,
3	OmniMark может работать в двух режимах. Первый основан на обработке событий в соответствии с правилами, т. е. когда событие (обычно поступающие из входного потока) вызывает выполнение некоторых правил. При этом OmniMark работает как типичный язык программирования четвертого поколения, Второй — последовательный режим — аналогичен работе языков программирования третьего поколения (например, C). Наша программа CG1 работает во втором режиме. При этом операции выполняются в последовательности, указанной в программе,
4-10	В OmniMark четыре основных типа переменных: счетчик *counter* целого типа, поток *stream* типа строка, переключатель *switch* логического типа и шаблон *pattern*. Помимо них, во внешних функциях OmniMark можно объявлять собственные типы данных, называемые непрозрачными. Мы используем непрозрачные типы для связи с базой данных и выборки записей в ответ на запрос.

(*продолжение*)

Все переменные, кроме шаблонов, должны быть объявлены как локальные или глобальные. Глобальные переменные доступны из любого места программы, локальные — только в зоне действия своего объявления (объявление переменных подробно освещено в главе 5 книги «OmniMark в действии» на прилагаемом к книге на компакт-диске). Для сбора информации, необходимой для выяснения причин неверной работы программы, применяется отладочный поток,

Переменные следует объявлять в самом начале области их действия. В нашем примере область действия переменных — обрабатывающая программа. При этом переменные будут доступны внутри программы вплоть до ее завершения. Если в области действия переменной окажется объявление другой переменной с тем же именем, во второй области действия ссылки будут относиться ко второй неременной. По окончании действия второго объявления ссылки снова будут относиться к внешней переменной. Добро пожаловать в мир иерархического программирования!

4 Переменной *methodName* будет присвоено название вызываемого метода.

6 Любые переменные OmniMark можно объединять в *полки* (shelf). Полка представляет собой несколько однотипных объектов, к которым можно обратиться по индексу (объект 1, объект 2 и т. д.) или по ключу. Полка аналогична ассоциативному массиву Perl.

Здесь мы создаем несколько потоков с общим именем *formVars*. Ключевое слово *variable* указывает, что это не просто переменная, а полка с однотипными элементами.

8 Непрозрачный тип *TCPConnection* определен в одной из внешних библиотек, подключенных с помощью файла SOAP.xin. Здесь этот тип служит для создания локальной переменной *myConnection* — дескриптора TCP-соединения.

12 Функция *cgiGetEnv* возвращает значение всех переменных среды CGI. Значения этих переменных помещаются на полку с потоками в формате название — значение; доступ к объектам на полке осуществляется по ключу. В результате полка *form Vars* содержит набор потоков, каждый из которых соответствует одной переменной среды. Эти переменные перечислены в табл. 8-4.

(*продолжение*)

Строка	Описание
	Для доступа к переменной нужно указать объект на полке. Например, для доступа к переменной QUERY_STRING в коде OmniMark понадобится оператор:
	`set queryString to cgiGetEnv["QUERY_STRING"]`
13	Функция *cgiGetEnv* работает со стандартными переменными среды. Протокол HTTP 1.1 позволяет создавать свои переменные среды. Мы используем одну такую переменную, которая передается в виде параметра *SOAPMethodName*. Функция *UTIL_GetEnv* предоставляет доступ к любому параметру из заголовка HTTP. Для выбора конкретного параметра следует задать его название прописными буквами с префиксом *HTTP_*.
15	Команда *set* устанавливает значение для потока. Поток можно открывать, закрывать и что-то в него добавлять, т. е. работать с ним, как с обычным файлом. В данном случае открывается поток *SOAPResponse*. Теперь в него можно записывать данные, как в файл в памяти.
16	CGI-интерфейс OmniMark инкапсулирует информацию, переданную методом *POST*, в поток *#main-input*, открытый только для чтения.
	Команда *do scan* передает строку последовательности шаблонов поиска *(match)*. Как только в потоке обнаруживается соответствие какому-то шаблону, выполняется соответствующий оператор (следующий за оператором *match)*.
17	Этому шаблону соответствует весь поступивший запрос. Шаблон *any* воспринимает только один символ. Если же после *any* указано целое число в фигурных скобках, обрабатывается соответствующее количество символов. В данном примере происходит опрос переменной среды *CONTENT_LENGTH* заголовка HTTP, после чего в переменную *inputData* помещается столько символов из входного потока, сколько их в запросе.
19	Команда *do when* аналогична условному оператору *if*. В данном случае если переменная *methodName* пуста, выводится сообщение о запросе неизвестного метода.
20–24	В случае ошибочных запросов используется элемент *SOAP:Fault*. Его элементам *faultcode* и *faultstring* присваиваются коды, позволяющих клиенту SOAP разобраться в причинах ошибки.
	Символы |[позволяют составить строку из нескольких частей.

(продолжение)

Это делает программу более читабельной. «%n» — символ новой строки.

26 Переменная *portSOAP* — массив записей с ключами. инициализируемый в файле SOAP.xin. Массив содержит сведения о названиях имеющихся служб, а также номер порта, с которым каждая из этих служб работает. Описание выглядит примерно так:

```
global stream portSOAP variable initial {
        "5432" with key "getWeather",
        "5433" with key "getXMLNewsRequest",
        "5434" with key "getAtomTimet"}
```

Найдя на полке с потоками запись, соответствующую запрашиваемому метолу *method Name,* мы выясняем, в какой порт отправить пакет.

27–31 Теперь для взаимодействия со службой надо установить соединение с указанным портом.

32 После открытия порта команда *put* передает данные через установленное соединение. В нашем случае объект OmniMark знает об XML и о SOAP, поэтому можно передать документ SOAP целиком. Однако мы могли бы сделать то же самое, даже если бы объект ничего не знал об XML и SOAP, — в задачи сервера SOAP входит преобразование поступившего запроса в форму, понятную для службы, к которой обрашен этот запрос.

В этом месте обработку запроса начинает служба, ожидавшая поступления данных в указанный порт. Пример такой программы gelXMLNews.xom см. в листинге 8-9. Эта программа получает пакет, выполняет обработку и возвращает результат серверу. В данном случае программа getXMLNews.xom возвратит документ XML, содержащий сформированный службой ответ.

35 Циклическая проверка проверяет активность соединения.

36–37 Сервер ожидает, когда служба вернет ответ. Ответ службы поступает в поток *responseXML.*

40–45 Если запрашиваемый метод *methodName* не опознан, в пакете *SOAP:Fault* окажется сообщение об ошибке.

49 Поток OmniMark похож на файл. Как и файл, поток нужно закрыть, прежде чем читать его по окончании записи. Документ, возвращаемый пользователю, содержится в потоке *SOAPResponse.*

(продолжение)

Строка	Описание
51–59	Команда *output* отправляет данные в поток вывода. OmniMark позволяет направлять вывод в разные потоки. У нас поток вывода направляется клиенту, вызвавшему CGI-программу.
51–52	Нам нужно присвоить значения нескольким переменным заголовка HTTP.
53	Два символа начала строки обозначают пустую строку, отделяющую заголовок HTTP от тела сообщения.
	Поскольку создается документ XML, в него включается объявление XML. Далее следует элемент *SOAP:Envelope*. Две вертикальные черты служат для объединения строк.
61-67	Среда OmniMark прекрасно обрабатывает исключения. Команды в операторе *catch* будут выполняться при возникновении любой критической ошибки. В OmniMark все иерархично, в том числе и обработка исключений. Это позволяет использовать единую процедуру обработки исключений, способную передавать обработку исключений другим процедурам.

Табл. 8-4. Серверные переменные, используемые командой cgiGetEnv.

Переменная	**Описание**
AUTH_TYPE	Протокол аутентификации пользователя. Эта переменная определена, только когда сервер поддерживает аутентификацию и она нужна для запуска CG1-программы.
CONTENT_ LENGTH	Размер запроса в байтах, переданного Web-сервером программе CG1. Эта переменная удобна при передаче методом POST данных HTML-форм на обработку.
CONTENT_TYPE	Тип данных, переданных Web-сервером программе CGI.
HTTP_CON-NECTION	Тип соединения между клиентом и сервером (например, HTTP_CONNECTION = Keep-Alive).
HTTP_HOST	IP-адрес или доменное имя сервера, которому адресован запрос.
HTTP_USER__ AGENT	Тип обозревателя и операционной системы клиента.

(*продолжение*)

QUERY_STRING	Упакованные данные заполненной пользователем формы, переданной серверу методом *GET*. В случае метода *POST* эта переменная не используется, так как тогда данные передаются CGI-программе через стандартный поток ввода (в терминах OmniMark это #process-input).

Программа getXMLNews.xom

В листингах 8-8 и 8-9 приведен код службы, обрабатывающей запрос *getXMLNewsRequest*. Соответствующий файл называется getXM LNews. хот. Код из листинга 8-8 (описание см. в табл. 8-5) циклически опрашивает соответствующий порт, считывает поступивший пакет и вызывает функцию, которая и выполняет роль службы. В листинге 8-9 (описание см. в табл. 8-6) приведен код функции, реализующей службу. Эта функция открывает базу данных, выбирает записи и формирует ответ в виде документа XML.

Листинг 8-8. Служба, обрабатывающая запросы getXMLNewsRequest.

```
getXMLNews.xom
1  include "$OAP.xin"
2  declare catch server-die
3
4  process
5      local TCPService myService
6      local TCPConnection myConnection
7      local stream caughtData
8      local stream responseXML
9
10     set myService to TCPServiceOpen
11        at portSOAP{"getXMLNewsRequest"}
12     put #error "getXMLNews listening on "
13        || portSOAP{"getXMLNewsRequest"} ||"%n"
14     throw server-die
15        unless TCPService-is-working myService
16
17     repeat
18        set daysWanted to 0
19
20        set myConnection
```

```
21          to TCPServiceAcceptConnection myService
22       throw server-die
23          unless TCPService-is-working myService
24       set caughtData to
25          TCPConnectionGetCharacters myConnection
25
27       put #error "getXMLNews caught and processed data on "
28          || portSOAP{"getXMLNewsRequest"}
29          || "%n--%n%g(caughtData)%n--%n"
30
31       clear fieldsWanted
32       using group processXML do
33          do xml-parse instance scan caughtData
34              suppress
35          done
36       done
37
38       open responseXML as TCPConnectionGetOutput myConnection
39       put responseXML getXMLNews(daysWanted, formatType,
40          fieldsWanted) j | crlf
41       close responseXML
42       catch #program-error
43          put #error "вызвало #program-error%n"
44
45       catch ttexternal-exceptior
45              identity catch-id
47              message catch-msg
48              location catch-loc
49          output 'Внешняя ошибка.%n'
50          output '%g(catch-loc)%n'
51          output '%g(catch-id) : %g(catch-msg)%n'
52    again
53
54    catch server-die
55       open responseXML as
56          TCPConnectionGetOutput myConnec Lion
57       put responseXML "<msg>getXMLNews Server killed</msg>"
58          | | crlf
59       close responseXML
50       halt
```

```
61 ;----------------------
62 group processXML
63 ;----------------------
64 element #implied
55    suppress
66
57 element format
58    set formatType to "%c"
69
70 element days
71    set daysWanted to "%c"
72
73 element fields
74    repeat scan "%c"
75       match [any except " "]+ => field
76          set new fieldsWanted key field to field
77              unless fieldsWanted has key field
78       match any
79    again
SO
81 element die
32    do scan "%c"
33        match content-start POISON-PILL content-end
84          put «error "Остановка%13#%10#"
35           throw server-die
86 done
```

Табл. 8-5. Описание кода из листинга 8-8.

Строка	Описание
1	**Команда** *include*, **как ей и положено, включает в файл кол из файла** SOAP.xin, **чтобы код программы и Web-службы мог использовать стандартные константы и функции.**
2	**Встроенные средства перехвата и обработки исключений** OmniMark **позволяют программно** обрабатывать **ошибки и исключения. В данном примере обрабатывается исключение, которое останавливает выполнение программы при передаче серверу определенной строки.**
4	**Запускается отдельный процесс.**

(продолжение)

Строка	Описание
5–6	Необходимо объявить две переменные: для соединения и для службы TCP. Эти переменные образуют канал для взаимодействия с сервером SOAP.
10–11	Служба отслеживает порт, который указан в потоке, объявленном в файле SOAP.xin.
12–13	Доступный только для записи поток *tterror* обычно выводится в окно, из которого запущена программа. Сразу после запуска выдается сообщение.
14–15	При неудачном завершении запуска службы TCP возникнет исключение (оно обрабатывается блоком catch),
17	В отличие от приведенной ранее в этой главе программы CG1, эта программа выполняется, пока не поступит предопределенная строка (или пока ее не остановит диспетчер служб). Структура *repeat...again* — это цикл, обрабатывающий полученные запросы по одному.
20-21	При каждом выполнении тела цикла нужно установить соединение со службой TCP.
22–23	Нужно проверить, работает ли служба TCP. Если нет, генерируется исключение.
24-25	Сервер ждет — функция *TCPConnectionGetCharacters* ожидает сообщений от программы CGI SOAPServer.xom.
27–29	Пакет выводится на консоль, чтобы можно было наблюдать за работой программы.
31	В пакете SOAP указаны запрашиваемые поля. Они находятся на полке с потоками. Поскольку это серверное приложение, программа будет обрабатывать очень много запросов, и мы должны «убрать за собой». В этой строке полка очищается, после чего она не содержит объектов. Теперь при обработке следующего запроса полка будет вновь инициализирована.
32–36	Средства обработки XML встроены в OmniMark, поэтому обработка поступившего документа XML проста. Группа операторов *processXML* содержит правила, обрабатывающие элементы XML-документа. Данный код обрабатывает документ XML, хранящийся в переменной *caughtData*, вызывая синтаксический анализатор XML, после чего данные передаются правилам, обрабатывающим свои элементы. Рассмотрим этот процесс подробнее.

(продолжение)

62 Объявление *group* создает область программы, к которой можно обратиться по имени. В этой группе находятся правила, каждое из которых обрабатывает свой элемент. Эти правила вызываются, когда синтаксический анализатор обнаруживает во входном потоке очередной элемент. В нашем случае входной поток — данные, поступающие от команды *do xml-parse* из строки 33.

Обрабатываемый документ XML выглядит так:

```
<SOAP:Envelope
        xmlns:SOAP="urn:schemas-xmlsoap-org:soap.v1">
<SOAP:Body>
        <getXMLNewsRequest
                xmlns="urn:schemas-architag-com:getXMLNews">
                <days>10</days>
                <format>XML</format>
                <fields>headline url abstract</fields>
        </getXMLNewsRequest>
</SOAP:Body>
</SOAP:Envelope>
```

64-65 Правило *#implied* вызывается, когда событие не обрабатывается другими правилами. Это правило будет обрабатывать элементы из пространства имен SOAP, которые нас не интересуют. Нам нужно извлечь заданное пользователем число дней, формат вывода и список полей.

67–71 Это правило вызывается, когда поступает элемент *format*, и устанавливает формат выходных данных. Строка «*%c*» означает содержимое элемента. Любое правило, обрабатывающее элементы, должно ровно один раз вызвать либо «*%c*», либо команду *suppress*. Напомню: программа getXMLNews.xom работает только с XML-потоком.

Правило для обработки элемента *days* выбирает заданное пользователем число дней и присваивает это значение соответствующей переменной.

73 Это правило обрабатывает элемент *fields*.

74-79 Блок *repeat scan* похож на *do scan* за исключением того, что строка обрабатывается циклически, пока не встретится символ, отличный от указанного. Оператор *match* каждый раз сравнивает обрабатываемую строку с шаблоном. Как только сравнение даст положительный результат, выполняются приведенные ниже операторы, и цикл повторяется.

(продолжение)

Строка	Описание
	В данном случае **поток сканируется в поисках затребованных пользователем полей. При** обнаружении поля *field* на **полке создается** новый элемент ***fieldsWanted.***
81	**Наше** серверное **приложение работает постоянно, однако** нам нужен **способ прервать его выполнение.** Для **этого в запросе SOAP** передается элемент **со словосочетанием:** `<die>poison-pill</die>` **Фраза, завершающая** работу **сервера, задана в** файле SOAP.xin. **Ее лучше держать** в секрете **— любой знающий эту фразу может** остановить **работу** сервера.
82–83	Кодовая фраза должна быть единственной строкой элемента *die.* Это проверяется шаблонами *content-start* и *content-end.*
84–85	При поступлении кодовой фразы на консоль выдается **сообщение об** ошибке, **и создается исключение *server-die...***
54–59	... которое обрабатывается здесь. Эта процедура обработки исключения передает элемент *msg,* закрывает соединение и завершает работу программы.
38	Выбрав **из входного потока параметры запроса, можно создать ответ для клиента. Для этого создаем поток** *responseXML,* открываем **соединение,** передаем **в поток последовательность данных и закрываем поток. После закрытия потока данные** попадут к **клиенту.**
39–40	Функция *getXMLNews* описана ниже. Она возвращает документ XML, который содержит новости, отвечающие запросу пользователя.
42–43	Omni**Mark** располагает **широкими** возможности **обработки исключений. Строка 42 обрабатывает все исключительные** ситуации, **которые** могут **возникнуть в ходе выполнения программы** (например, **попытки чтения или записи данных в поток, который не был открыт).**
45–51	Обработчик прерываний в 45 строке перехватывает внешние исключения — например, попытки доступа к закрытой базе данных или записи в файл, **открытый только для чтения.**
52	Здесь заканчивается цикл, начатый в строке 17.

Листинг 8-9. Функция, возвращающая документ XML с запрошенными новостями.

```
1  macro outField token tagName is
2     do when fields has key tagName
3        put newsXML "%n <%@(tagName)>"
4           || cleanString
5              (dbFieldValue rsXMLNews key tagName)
6           | "</%@(tagName)>"
7     done
8  macro-end
9
10 define stream function getXMLNews (
11       value counter daysDesired,
12       value stream format,
13       read-only stream fields) as
14    local stream newsXML initial {""}
15    local dbDatabase dbXMLNews
15    local dbField rsXMLNews variable
17    local stream strSQL
18    local stream minusDate
19    local stream finalDate
20
21 set dbXMLNews to dbOpenODBC "XMLNews"
22 set minusDate to add-to-ymdhms now-as-ymdhms days -daysOesired
23 set finalDate to format-ymdhms "=xM/=xD/=xY" with date minusDate
24    set StrSQL to "SELECT - FROM News "
25       || "WHERE Date >= #%g(finalDate)# ORDER BY Date DESC"
26    dbQuery dbXMLNews sql StrSQL record rsXMLNews
27
28    open newsXML as buffer
29    do when dbRecordExists rsXMLNews
30       put newsXML "%n<getXMLNewsResponse days='%d(daysWanted)'>"
31       repeat
32          exit unless dbRecordExists rsXMLNews
33          put newsXML "%n <item date='"
34             | j dbFieldValue rsXMLNews key "date" || "'>"
35
36          outField "url"
37           outField "location"
38           outField "headline"
```

```
39          outField "abstract"
40
41          put newsXML "%n </item>"
42          dbRecordMove rsXMLNews
43       again
44
45       put newsXML "%n</getXMLNewsResponse>"
46    else
47       put newsXML "%n<getXMLNewsResponse days='%d(daysWanted)'/>"
48    done
49
50    close newsXML
51    return newsXML
52
53    catch #external-exception
54          identity catch-id
55          message catch-msg
55          location catch-loc
57       output 'Произошло внешнее исключение.%n'
58       output '%g(catch-loc)%n'
59       output '%g(catch-id) : %g(catch-msg)%n'
```

Табл. 8-6, Описание функции из листинга 8-9.

Строка	Описание
10–13	В OmniMark есть два иетода создания вызываемого кода. Здесь мы создаем функцию *getXMLNews*. Функции можно вызывать из любого места программы. *getXMLNews* возвращает переменную типа поток. У функции три аргумента: счетчик, поток и полка с потоками.
14	Поток *newsXML* служит для создания документа XML, который будет возвращен в качестве ответа SOAP,
15–16	Библиотека работы с базами данных OmniMark (OmniMark Database, OMDB) содержит объекты и функции, необходимые для взаимодействия с базами данных. В программе создаются два таких объекта: один для связи с ODBC, а другой для хранения набора записей, получаемых в результате выполнения SQL-запроса к базе данных.
21	Функция *dbOpenODBC* открывает источник данных *XMLNews*.

(продолжение)

22–23 Для создания запроса SQL нужно вычислить дату, соответствующую заданному пользователем числу дней назад. Это целое число передается в аргументе *daysDesired.* Стандартная библиотека OmniMark для работы с датами и временем легко позволяет получить желаемый результат. Даты всегда хранятся в формате *ymdhms* (год, месяц, день, час, минута, секунда). Сначала нужно определить день, начиная с которого ищутся новости. Для этого из текущего дня вычитается значение *daysDesired.* После этого дату следует перевести в формат для SQL-запроса, т. е. привести к виду 4/26/2000.

24–25 Вычисленная дата включается в SQL-запрос. Теперь запрос готов, и его можно передать СУБД с помощью функции *dbQuery.* Результат выполнения запроса в виде набора записей будет возвращен в переменной *rsXMLNews.* Он представляет собой полку с данными непрозрачного типа, определенного в библиотеке OMDB.

28 Поток *newsXML* открыт так, что можно добавлять в него данные.

29 Функция *dbRecordExists* проверяет, есть ли в результирующем наборе хоть одна запись.

30 Если удовлетворяющие запросу записи есть, мы начинаем создавать документ XML с корневого элемента *getXMLNewsResponse.*

31-43 Цикл *repeat...again* выполнится для каждой записи набора.

32 Когда будет обработана последняя запись, функция *dbRecordExisls* возвратит значение *false,* и цикл *repeat...again* завершится.

34 Доступ к полям набора записей аналогичен доступу к полкам с ключом — мы ищем элемент, соответствующий указанному значению ключа, для чего вызывается функция *dbFieldValue.* В этой строке выводится значение поля *date* текущей записи.

36–39 Макрокоманда *outField* вызывается четыре раза, передавая названия полей базы данных.

1–8 Простые, но очень эффективные макрокоманды представляют собой участки кода, который подставляется вместо названия макрокоманды на этапе компиляции. Данная макрокоманда проверяет, указал ли пользователь соответствующее поле в запросе. Если да. макрокоманда создает элемент и помешает в него значение соответствующего поля.

42 Функция *dbRecordMove* загружает в *rsXMLNews* следующую запись.

45 Эта строка закрывает документ *getXMLNewsResponse.*

(продолжение)

Строка	**Описание**
47	Если запрос не вернул ни одной записи, создается пустой элемент, сообщающий о том, что за указанный интервал времени новостей не было,
50-51	Поток закрывается и возвращается вызвавшему приложению.
53-59	Этот обработчик исключений предназначен для обработки ошибок внешних интерфейсов. Ошибка может возникнуть, например, при отсутствии нужного файла или при сбое СУБД.

Тестирование службы

Для проверки нашего приложения нужно сделать следующее.

1. Запустите службу getXMLNews командой:

```
omnimark -sb getXMLNews.xom -i «c:\omnimark\xin\»
-l «c:\omnimark\lib\=L.dll» -log getXMLNews.log
```

   Путь к библиотекам (*-l*) и импортируемым файлам (*-i*) зависит от того, как вы установили Omni Mark.

2. Загрузите документ getXMLNews.htm в обозреватель.
3. Укажите количество дней, формат данных (наша программа поддерживает только данные XML) и перечислите желаемые поля. Теперь щелкните кнопку «Получить новости» и проследите за работой службы по сообщениям, которые выводятся на консоль (рис,8-5).

***Рис.** 8-5. **Окно** консоли с выводом программы getXMLNews.xom.*

Глава 9

Инфраструктура BizTalk

Что произойдет, если сотни компаний будут создавать собственные схемы для описания счетов? Каковы шансы, что эти схемы будут совместимы между собой? Схемы могут содержать одни и те же элементы, но при этом каждая будет использовать собственные названия элементов. Например, счета обычно имеют номер, но имя элемента, содержащего номер счета, может быть *InvNum*, *InvNo*, *InvoiceNo*, *InvoiceNum* и т. д. В результате мы столкнемся с сотнями XML-документов, описывающих однотипные данные. Каждый документ будет иметь свою структуру, затрудняющую обмен данными между компаниями. Если каждый из ваших бизнес-партнеров будет использовать свою схему, вы вынуждены будете создавать отдельные программы для обработки каждого XML-документа. Понятно, что этого хотелось бы избежать — гораздо лучше, если все будут использовать одну и ту же структуру.

Приведенный пример иллюстрирует необходимость создания инфраструктуры для разработки схем электронного документооборота. Такая инфраструктура должна описывать общий набор элементов для некоторых объектов, сохраняя при этом достаточную гибкость, позволяющую разработчикам сотрудничать при создании схем. Этой цели и служит инфраструктура BizTalk.

BizTalk предлагает набор правил и исходный набор тэгов для создания схем электронного документооборота (прежде всего в секторе «бизнес — бизнес»). Инициатива BizTalk ставила цель содействовать распространению XML и облегчить его использование для построения общепринятых схем. Единый набор правил создания схем должен способствовать росту числа схем, что в свою очередь должно облегчить партнерам процесс выбора схемы в каждой конкретной ситуации. BizTalk предоставляет широкой аудитории возможность выработать единый подход к использованию XML.

Для обработки документов BizTalk нужен *BizTalk-сервер*, позволяющий независимым поставщикам программного обеспечения и разработчикам проще и быстрее связывать бизнес-процессы и, следовательно, ускорить внедрение электронного документооборота во всех областях бизнеса, использующих XML.

Инфраструктура BizTalk едина в четырех лицах.

- **Независимая спецификация документа инфраструктуры BizTalk** (BizTalk Framework Independent Document Specification) — описывает набор тэгов, определяющий метод пересылки документов и вызов процедур их обработки каждой из сторон. В этой ипостаси BizTalk работает как своеобразный виртуальный конверт для пересылки бизнес-документов между партнерами. Вы можете ознакомиться с текстом этой спецификации в приложении Б или в каталоге \References компакт-диска книги.
- **BizTalk.org** — сайт, посвященный технологиям электронного бизнеса, XML вообще и BizTalk в частности. Здесь работают дискуссионные группы, обеспечивающие поддержку разработчиков схем и помощь в интеграции в инфраструктуру BizTalk. (Подробнее о сайте BizTalk.org см. приложение В.) На этом сайте ведется депозитарий схем, где каждый может бесплатно опубликовать свои схемы для использования всеми желающими.
- **Сервер BizTalk** — для обработки BizTalk-документов требуется BizTalk-сервер — приложение, которое читает и обрабатывает документы. Действия по обработке зависят от конкретной системы и задачи. Рассмотрим другие серверные приложения, например, сервер электронной почты. Реализаций серверов электронной почты масса: для разных платформ, от бесплатных до весьма дорогостоящих. Точно так же сервер BizTalk может быть написан на любом языке программирования и функционировать на любой платформе. Некоторые компании стремятся создать серверы, обрабатывающие документы BizTalk и интегрированные с корпоративными СУБД. Пример простейшего сервера BizTalk мы рассмотрим в главе 11.
- **Microsoft BizTalk Server 2000** — это BizTalk-сервер для платформы Microsoft Windows 2000 Server. Поскольку инфраструктура BizTalk является открытым стандартом, вы можете создавать сообщения BizTalk на сервере, написанном на Java и работающем под управлением Linux, и читать эти сообщения на сервере, работающем под управлением Windows 2000 Advanced Server, используя Microsoft BizTalk Server 2000. Microsoft BizTalk Server 2000 представляет собой нечто большее, чем просто приложение, способное создавать

и обрабатывать документы BizTalk. Этот программный продукт образует полнофункциональную среду для интеграции взаимодействия с партнерами на базе **XML.**

Спецификации документов и сообщений BizTalk

Инфраструктура BizTalk поощряет применение XML в B2B-транзакциях, предлагая структуру для создания XML-документов. В этом разделе мы познакомимся с дополнительными возможностями BizTalk.

Говорят, что кодирование данных с помощью XML подобно применению ASCII в текстовых документах: формат настолько гибок, что нескоординированные действия не являются помехой для совместимости. Допустим, вы создаете текстовый документ, используя ASCII. Следовательно, я должен суметь прочитать его. так? Ладно, теперь допустим, вы пишете по-французски, а я читаю на английском. Даже при том, что мы оба используем **ASCII,** вовсе не факт, что мы поймем друг друга. Единая техника кодирования подразумевает всего лишь то, что я смогу прочитать ваш файл на моей машине. Полное взаимопонимание возможно, лишь когда я буду знать, какую именно структуру вы задействовали при создании информации. Та же проблема возникает и в случае XML. Сам факт использования вами XML при создании документа вовсе не означает, что я смогу понять его. Это значит только то, что я смогу убедиться, создан ли документ по правилам XML и, если есть схема, действителен ли документ, т. е. соответствует ли он структуре, определенной схемой.

Диапазон вопросов, возникающих при применении XML для описания информации простирается отрешения, что, собственно, описывать, до того, как назвать каждое поле. Решая эти проблемы, пользователь должен разметить данные так, чтобы противоположная сторона или приложение смогли однозначно решить, что именно ими получено.

Инфраструктура BizTalk предоставляет разработчикам возможности написания приложений, значительно облегчающих обработку документов BizTalk. Она предлагает набор тэгов и рекомендации по созданию собственных наборов тэгов для обмена информацией с бизнес-партнерами. Иначе говоря, BizTalk определяет единый поход к применению XML. В результате вы легко определите тип полученного XML-документа и выделите содержащуюся в нем информацию.

BizTalk рассматривает документы как сообщения, которыми обмениваются бизнес-процессы, включая методы создания документов и их маршрутизацию. Можно сказать, что BizTalk обеспечивает механизм «почтовой рассылки» XML-документов.

Во многих отраслях промышленности ведется работа над схемами для решения проблем информационного обмена в конкретной области. Такие схемы совершенно необязательно должны быть межотраслевыми. Схема для передачи медицинской информации, например, не годится для финансовых операций, а схема для обмена сведениями о страховых исках в целом бесполезна для операций с недвижимостью. Такие отраслевые группы обычно сосредоточены на определении структур данных, а не на том, как именно информация доставляется потребителю.

Как вы увидите, передаваться может любой документ, использующий любую схему. Таким образом, BizTalk является «обложкой» для бизнес-документов и работает подобно обычному почтовому конверту. Рассмотрим бизнес-операцию, выполненную привычным образом, и попытаемся выяснить, как применить XML для ее выполнения.

Использование BizTalk в транзакциях «бизнес — бизнес»

Рассмотрим сценарий B2B-транзакции, использующей инфраструктуру BizTalk для создания документа XML.

Допустим, наша компания производит серьезный товар — радиоуправляемые игрушечные автомобили. Наши машинки работают на бензине и довольно велики, чтобы заинтересовать собак и детей, живущих по соседству. Мы покупаем комплектующие и услуги у ряда поставщиков. В свою очередь у этих компаний свои поставщики и покупатели.

Для связи с нашими поставщиками и покупателями, включая наших дилеров и потребителей нашей продукции, мы хотим использовать Интернет. И, естественно, мы хотели бы максимально задействовать имеющиеся технологические возможности.

Подготовка заказа

У любой бизнес-транзакции есть инициирующее событие — скажем, заказ поставщику на поставку продукции. Допустим, моя компания собирается закупить у вашей 10 моторчиков для машин. Мой первый шаг — создание заявки на покупку, которая содержит поля, необходимые соответствующему персоналу для обработки запроса. Это внутренний документ, созданный для взаимодействия между подразделениями моей компании.

Заполнив требование на покупку, я отдаю его сотруднице, отвечающей за обработку требований и оформление заказов на закупку. Ее зовут Джейн. Джейн знает бизнес-правила обработки подобных документов и будет применять их при работе с моим запросом.

Первое, в чем должна убедиться Джейн, — имею ли я право тратить деньги компании. Она проверяет утвержденный бюджет и убеждается, что мое имя есть в списке и я обладаю достаточными полномочиями для совершения этой закупки.

Затем Джейн выясняет, нет ли на нашем складе подобных изделий. Покупать изделия, имеющиеся в наличии, не выгодно, и это бизнес-правило предотвращает возникновение подобной проблемы.

Убедившись, что с моим заказом все в порядке, Джейн кладет его в стопку других запросов на покупку, лежащую на ее рабочем столе. Другие сотрудники моей компании займутся закупкой продукции вашей компании. Согласно договоренностям между нашими компаниями заказ на покупку, если он не срочный, оформляется только при наличии минимум трех наименований в списке. У Джейн уже есть два заказа, и она знает, что пришло время оформить заказ на продукцию вашей компании. Джейн загружает в свой текстовый процессор форму, заполняет необходимые поля (адреса, финансовые реквизиты, сроки поставки, номера продукции по прейскуранту, описания и цены) и печатает ее.

Затем она готовит другой документ — стандартный почтовый конверт. Она печатает название и адрес моей компании в верхнем левом углу, а те же данные о вашей компании — в центре конверта. Указывать, что адрес в центре конверта означает «Кому», а другой — «От кого», незачем — мы к этому давным-давно привыкли. Над названием вашей компании Джейн помещает важную информацию: *вниманию подразделения по обработке заказов.* Позже мы увидим, как этот элемент инициирует определенные действия в вашей компании.

Джейн вкладывает в конверт сложенный листок с заказом, запечатывает и наклеивает нужную марку. Затем она идет к ближайшему почтовому ящику и опускает письмо. Благодаря почтовой службе США конверт совершает таинственное путешествие, заканчивающееся в вашем почтовом ящике.

Если бы две наши компании должны были бы создавать собственную систему доставки заказов (т. е. если бы почтовой службы не было), мы потратили бы на это массу времени вместо того, чтобы заниматься заказами. Мы должны были бы разработать систему адресов, нанять курьеров или договориться с другими курьерскими службами, побеспокоиться о страховании на случай непредвиденных обстоятельств и решить тысячи других вопросов. К счастью для нас, почтовая инфраструктура существует, и мы можем использовать ее за сравнительно небольшую плату.

Необходимо также отметить, что у Джейн несколько вариантов передачи сообщения моей компании в вашу. Она использовала по-

чтовую службу США, но могла прибегнуть к услугам FedEx, Airborne Express или даже курьера, если наши офисы расположены неподалеку. Кроме того, она могла отправить заказ по факсу, по электронной почте или даже разместить его, используя Web-страницу вашей компании.

Пару дней спустя конверт оказывается в вашем почтовом ящике. Сотрудник, разбирающий почту, читает строку *Вниманию подразделения по обработке заказов,* которую Джейн поместила над вашим адресом, и переправляет письмо служащему, отвечающему за обработку заказов. Пусть его зовут Дэвид.

Обработка заказа

Дэвид смотрит на конверт, записывает адрес отправителя, открывает конверт и достает документ. Дэвид отвечает за обработку многих видов документов, в том числе заказов на покупку, счетов и предложений о сотрудничестве (Requests for Proposals, RFP).

Увидев, что присланный документ — заявка на покупку, Дэвид приступает к его обработке. Для начала он должен убедиться, что между нашими компаниями имеется соглашение о проведении подобных операций. Подобные соглашения обычно содержат описание сроков и условий, например, времени доставки, условий возврата и деталей оплаты. Затем он проверяет, оплатила ли моя компания ранее выставленные счета и нет ли спорных вопросов по предыдущим сделкам. Ясно, что поставка продукции компании, не оплачивающей счета, недопустима.

Убедившись, что его компания действительно сотрудничает со мной, Дэвид подключается к системе обработки заказов и вводит данные из нашего заказа, руководствуясь бизнес-правилами, принятыми в компании. Одно из таких правил требует запросить склад, чтобы выяснить, есть ли в наличии продукция, перечисленная в заказе.

Дэвид обнаруживает, что два из трех видов продукции, перечисленных в заказе, имеются на складе и могут быть отправлены немедленно, а третьего нет — его нужно заказать повторно. По соглашению его компания обязалась подтверждать заказы вроде того, что Джейн отправила Дэвиду. Поэтому Дэвид загружает в свой текстовый процессор соответствующую форму и заполняет необходимые поля. Он указывает, что два вида продукции будут отправлены сегодня, а третий необходимо заказать повторно.

Заполнив форму, Дэвид готовит конверт и надписывает два адреса. Он обращается к полученному конверту и меняет местами адреса, помещая адрес своей компании в центр, а адрес моей — в верхний левый угол.

Приготовив конверт, он вкладывает в него подтверждение, запечатывает конверт, наклеивает марку и опускает его в почтовый ящик. Пару дней спустя Джейн получает подтверждение и обрабатывает его согласно бизнес-правилам моей компании. Одно из них требует от нее связаться с тремя служащими, подававшими заявки на покупки, и сообщить им о состоянии их заказов. Тот, чья заявка осталась неудовлетворенной, может попросить Джейн отправить письмо об отмене заказа и оформить новый заказ другому поставщику комплектующих. Подобный вариант может привести к применению других бизнес-правил с обеих сторон.

Это только один из сценариев. Джейн может использовать другие варианты оформления заказов. Например, не связывать срочный заказ с двумя другими, срочно отправить заказ Дэвиду по факсу или позвонить ему, чтобы подтвердить получение. Наконец, она могла бы отправить заказ и необходимые документы с курьером. Джейн может принять любое решение, способное повысить эффективность ее работы. Она принимает такие решения на основании бизнес-правил компании.

В каждой компании свои бизнес-правила. Появление XML не требует их изменения. Все. чего мы ждем от XML и инфраструктуры BizTalk, — это помочь нам повысить эффективность бизнеса.

Эволюция обмена информацией

Компания, описанная в примере, разработала набор приложений и бизнес-процессов, улучшающих ведение дел. Мы подобрали компьютерные программы, определили обязанности и наняли служащих, чтобы все это работало.

Вы также купили набор приложений и бизнес-процессов, позволяющий вам успешно вести дела. Однако, возможно, наши системы несовместимы. Наши компании используют бумажные документы в качестве универсального средства обмена информацией, но для обработки этих документов требуется вмешательство людей.

Сотрудник, обрабатывающий информацию, должен знать бизнес-правила и уметь интерпретировать документы. В нашем случае эту роль исполняют Джейн и Дэвид.

Операция, описанная мной, происходит не в вакууме. Перед тем, как я отправил заказ на покупку, между нашими компаниями было заключено соглашение о совместной деятельности. Такой контракт описывает условия оплаты, поставки, порядок решения спорных вопросов и пр. В частности, это соглашение описывает и операцию, которую мы рассматриваем. Соглашение указывает, какая именно ин-

формация необходима для успешной обработки документа, и мы оговорили, в каком формате я хотел бы получать подтверждение заказа.

Этот тип операций успешно применялся веками, но автоматизированная система, заменяющая его, позволила бы сделать операции гораздо эффективнее.

Заказ с использование инфраструктуры BizTalk

Документ BizTalk (рис. 9-1) представляет собой сообщение в формате SOAP 1.1, в теле которого содержится один или несколько бизнес-документов (о формате конверта SOAP см. главу 8).

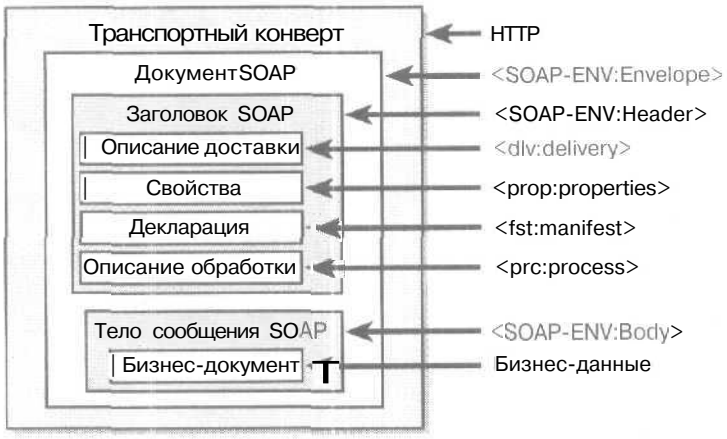


Рис. 9-1. Документ BizTalk представляет собой сообщение в формате SOAP 1.1, элементы которого описывают маршрутизацию и обработку бизнес-документов.

Документ BizTalk можно рассматривать как расширение документа SOAP. Спецификация BizTalk содержит несколько пространств имен, каждое из которых позволяет решить определенные задачи. Тэги этих пространств имен имеют общее название *бизнес-тэги* (BizTags).

Документ содержится в элементе SOAP *Envelope* — корневом элементе документа. Как вы знаете из главы 8 и рис. 9-1, в состав документа входит необязательный элемент *Header* и обязательный элемент *Body*. В сообщении BizTalk элемент *SOAP:Header* обязателен, поскольку содержит некоторые обязательные бизнес-тэги. Элемент *SOAP:Body* содержит бизнес-данные, состоящие из одного или нескольких документов.

В табл. 9-1 перечислены бизнес-тэги заголовка *SOAP:Header* BizTalk. Каждый из них относится к собственному пространству имен; они подробно описаны на сайте *schemas.biztalk.org*.

Табл. 9-1. Элементы заголовка BizTalk SOAP:Header.

Название бизнес-тэга	Обязательный	Пространство имен
delivery	Да	*http://schemas.biztalk.org/btf-2-0/delivery*
properties	Да	*http://schemas.biztalk.org/btf-2-0/properties*
manifest	Нет	*http://schemas.biztalk.org/btf-2-0/manifest*
process	Нет	*http://schemas.biztalk.org/btf-2-0/process*

Документ BizTalk сам не выполняет никаких функций, пока не отправлен. Транспортировка документа — задача транспортного конверта, в роли которого может выступать практически любой протокол электронной передачи, включая HTTP, SMTP и FTP. Отметим, что спецификация BizTalk 2.0 описывает только транспортировку документов с использованием HTTP.

Рассмотрим набор тэгов инфраструктуры BizTalk в действии. В листинге 9-1 показан документ BizTalk, выполняющий работу Джейн и Дэвида. Содержание документа подробно описано в таблице 9-2. Подробное описание документов и сообщений BizTatk Framework 2.0 см. в приложении Б.

Листинг 9-1. Документ BizTalk, описывающий заказ на покупку,

NewPO.xml

```
1  <?xml version="1.0"?>
2  <SOAP-ENV:Envelope
3     xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
4  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
5  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
6  <SOAP-ENV:Header>
7     <dlv:delivery SOAP-ENV:mustUnderstand="1"
8     xmlns:dlv="http://schemas.biztalk.org/btf-2-0/delivery"
9     xmlns:agr="http://www.trading-agreements.org/types/">
10       <dlv:to>
11          <dlv:address xsi:type="agr:department">Toy Car Parts
            </dlv:address>
12       </dlv;tO>
```

```
13        <dlv:from>
14           <dlv:address xsi: type="agr:organization">Toi Carz
             </dlv:address>
15        </dlv:from>
16        <dlv:reliability>
17           <dlv:confirmTo>http://toicarz.com/biztalk/confirm.asp
             </dlv:confirmTo>
18           <dlv:receiptRequiredBy>2000-07-7T08:00:00+08:00
          </dlv:receiptRequiredBy>
19        </dlv:reliability>
20     </dlv:delivery>
21     <prop:properties SOAP-ENV:mustUnderstand="1"
22     xmlns:prop="http://schemas.biztalk.org/btf-2-0/properties">
23       <prop:identity>uuid:74b9f5d0 33fb 4a81 b02b 5b760641c1d6
          </prop:identity>
24        <prop:sentAt>2000-05-14T03:00:00+08:00</prop:sentAt>
25        <prop:expiresAt>2000-05-15T04:00:00+08:00</prop:expiresAt>
25        <prop:topic>http://toycarparts.com/BTServer.xar</prop:topic>
27     </prop:properties>
28     <fst:manifest
       xmlns:fst="http://schemas.biztalk.org/btf-2-0/manifest">
29     <fst:reference fst:uri="#PurchaseOrderBody">
30        <fst:description>Заказ на покупку</fst:description>
31     </fst:reference>
32     <fst:reference fst:uri="toicarz.sig">
33         <fst:description>Цифровая подпись</fst:description>
34     </fst:reference>
35     </fst:manifest>
36  </SOAP-ENV:Header>
37
38  <SOAP-ENV:Body>
39     <PurchaseOrder_1.0
40     xmlns="x-schema:http://toicarz.com/schemas/PurchaseOrder.xdr"
41     id="PurchaseOrderBody"
42     type="заказ"
43     PONumber="10-01-2118">
44     <Item number="122-11" quantity="100"/>
45     <Item number="237-82" quantity="10"/>
45     <Item number="811-91" quantity="25"/>
47     </PurchaseOrder_1.0>
```

```
48 </SOAP-ENV:Body>
49</SOAP-ENV:Envelope>
```

Табл. 9-2. Описание документа BizTalk, приведенного в листинге 9-1.

Шаг	Описание
1	Это документ XML, следовательно, он начинается с объявления XML.
2	Документ XML должен иметь уникальный корневой элемент, открывающий тэг которого стоит в начале документа, а закрывающий — в конце. Поскольку это конверт SOAP, его корневой элемент — *SOAP-ENV:Envelope*.
3-5	Этот документ использует три пространства имен. Они указывают, какие именно схемы используются в конверте SOAP.
6-39	Бизнес-тэг *SOAP-ENV:Header* содержит идентификатор документа и информацию о его маршрутизации.
7–20	Бизнес-тэг *dlv:delivery* содержит информацию о доставке документа и о необходимости подтверждения.
7	Атрибут *mustUnderstand* сообщает приложению-получателю о необходимости анализа получаемой информации. В случае обнаружения некорректной информации приложение должно вернуть ошибочный элемент SOAP отправителю и не пытаться продолжить обработку.
8	Бизнес-тэги доставки входят в пространство имен, описанное в этом объявлении.
9	Объявление еще одного пространства имен, содержащего набор соглашений с торговым партнером.
10	Бизнес-тэг *dlv:to* указывает получателя документа.
11	Атрибут *xsl:type* описывает тип данных согласно второй части спецификации схем XML.
12	Тип данных *to* определен в схеме соглашения с торговым партнером.
13–15	Адрес *from* имеет ту же форму и содержит те же атрибуты, что и адрес *to*.
16–19	Сервер BizTalk, создающий документ, может запросить подтверждение получения сообщения с помощью необязательного бизнес-тэга *dlv:reliability*.
17	Бизнес-тэг *dlv:confirmTo* указывает адрес, по которому необходимо направить подтверждение.

(продолжение)

Шаг	Описание
18	Если система-отправитель не получает подтверждение за время, указанное в бизнес-тэге *dlv:receiptRequiredBy*, документ отправляется повторно.
21–27	В разделе свойств определяются некоторые свойства документа.
22	Бизнес-тэги элемента *prop:properties* принадлежат объявленному здесь пространству имен.
23	Бизнес-тэг *prop:identity* содержит уникальный идентификатор (Universally Unique Identifier, UUID). Большинство платформ поддерживают генерацию уникальных идентификаторов.
24	Бизнес-тэг *prop:sentAt* указывает время и дату отправки документа (она имеет тип datetime.tz).
25	Бизнес-тэг *prop:expiresAt* указывает время окончания срока действия документа. После этого документ считается недействительным и не подлежит обработке получателем.
26	Бизнес-тэг *prop:topic* указывает BizTalk-сервер, который будет обрабатывать документ.
28–35	Необязательный бизнес-тэг *fst:manifest* описывает содержание документа BizTalk.
29–31	Бизнес-тэг *fst:reference* может появляться один или несколько раз. Он содержит унифицированный идентификатор ресурса (Uniform Resource Identifier, URI) и описание содержимого документа, вложенного в конверт. Первый элемент *fst:reference* указывает на документ, вложенный в элемент *Body* элемента *SOAP Envelope*.
32-34	В документ также включен файл цифровой подписи, гарантирующий подлинность документа. Файл подписи, вероятнее всего, будет включен в транспортный конверт в виде MIME-вложения.
38-48	Элемент *SOAP-ENV:Body* следует за элементом *SOAP-ENV:Header*. Элемент *Body* — аналог вложенного в конверт письма из нашего примера — содержит ваш бизнес-документ и определен с использованием вашей схемы,
40	Поскольку ваш документ использует свою схему, необходимо объявить пространство имен. В нашем случае мы используем схему PurchaseOrder.xdr, размещенную на нашем сервере. Приложение-получатель сможет найти эту схему, если оно имеет доступ в Интернет.
41–49	Остальная часть документа представляет собой заказ на покупку, за которым следуют закрывающие тэги элементов *PurchaseOrder_1.0*, *SOAP-ENV:Body* и *SOAP-ENV:Envelope*.

Документ BizTalk фактически представляет собой два документа. Один заменяет бумажный конверт, другой — бумажный документ, вложенный в него. Новый подход с использованием пространств имен и схем XML позволяет создавать виртуальный электронный конверт с виртуальным документом внутри,

ПРИМЕЧАНИЕ Синтаксис определения типа документа (DTD) допускает наличие только одного определения. Документ BizTalk фактически содержит два типа документа; для этого мы используем сокращенный синтаксис схем данных (XML Data Reduced, XDR). Обработка транзакций BizTalk на основе синтаксиса определения типа документа невозможна.

Когда в вашей компании будет получен XML-документ (рис. 9-1), сервер BizTalk приступает к его чтению и анализу. Прежде всего он считывает бизнес-тэг *dlv:to*, чтобы выяснить, кому адресован документ. Затем сервер BizTalk, возможно, займется синтаксическим анализом и обработкой оставшихся данных с целью создания соответствующей записи в базе данных регистрации заказов. Наконец, сервер выделит собственно заказ — данные, содержащиеся в элементе *SOAP-ENV:Body*, — и передаст его приложению, которое обработает заказ. Тип обработки целиком определяется бизнес-процессами конкретной компании.

Уведомление о получении заказа

Поскольку мы включили в документ бизнес-тэг *dlv:reliability*, сервер-получатель должен подтвердить поступление заказа (листинг 9-2).

Листинг 9-2. Уведомление, посылаемое сервером *BizTalk,*

```
Receipt.xml
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header>
   <rct:receipt SOAP-ENV:mustUnderstand="1"
   xmlns:rct="http://schemas.biztalk.org/btf-2-0/receipt">
   <rct:receivedAt>2000-05-15T04:08:10-05:30</rct:receivedAt>
            </rct:receipL>
         <prop:properties SOAP-ENV:mustUnderstand="1"
         xmlns:prop="http://schemas.biztalk.org/btf-2-0 oroperties">
            <prop:identity>
            uuid:74b9f5d0-33fb-4a81-b02b-5b7606387dc1
            </prop:identity>
```

```
            <prop:sentAt>2000-05-14T03:00:00+08:00</prop:sentAt>
            <prop:expiresAt>2000-05-15T04:00:00+08:00</prop:expiresAt>
            <prop:topic>http://toycarparts.com/POProc.asp</prop:topic>
        </prop:properties>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body/>
</SOAP-ENV:Envelope>
```

Заголовок этого документа сходен с заголовком полученного сообщения. Единственные отличия — смена уникального идентификатора в бизнес-тэге *prop:identity* и отметка о дате и времени получения.

Заметьте: элемент *SOAP-ENV:Body* пуст. Хотя он и обязателен, в содержимом нет никакой необходимости — ведь это всего лишь уведомление об успешном получении документа.

Подтверждение заказа

Как только заказ будет проведен через систему обработки заказов, вы должны отправить заказчику подтверждение. Соответствующий документ представлен в листинге 9-3, а его описание — в табл. 9-3.

Листинг 9-3. Документ BizTalk, подтверждающий поступление заказа из одной компании в другую.

```
Confirm.xml
1  <?xml version="1.0"?>
2     <SOAP-ENV:Envelope
3  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
4  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
5  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
6  <SOAP-ENV:Header>
7     <dlv:delivery SOAP-ENV:mustUnderstand="1"
8     xmlns:dlv="http://schemas.biztalk.org/btf-2-0/delivery"
9     xmlns:agr="http://www.trading-agreements.org/types/">
10       <dlv:to>
11       <dlv:address xsi:type="agr:department">Toi Carz</dlv:address>
12       </dlv:to>
13       <dlv:from>
14          <dlv:address xsi:type="agr:organization">
         Toy Car Parts</dlv:address>
15       </dlv:from>
15       <dlv:reliability>
17          <dlv:confirmTo>
            http://toycarparts.com/BTServer.xar</dlv:confirmTo>
```

```
18          <dlv:receiptRequiredBy>
            2000-07-7T08:00:00+08:00</dlv:receiptRequiredBy>
19       </dlv:reliability>
20    </dlv:delivery>
21    <prop:properties SOAP-ENV:mustUnderstand="1"
22    xmlns:prop="http://schemas.biztalk.org/btf-2-0/properties">
23       <prop:identity>uuid:74b9f5d0-33fb-4a81-b02b-5b7606434d3
         </prop:identity>
24       <prop:sentAt>2000-05-16T03:00:00+08:00</prop:sentAt>
25       <prop:expiresAt>2000-05-17T04:00:00+08:00</prop:expiresAt>
26       <prop:topic>
         http://toicarz.com/biztalk/confirm.asp</prop:topic>
27    </prop:properties>
28 </SOAP-ENV:Header>
29
30 <SOAP-ENV:Body>
31    <PurchaseOrder_1.0
32    xmlns="x-schema:http://toicarz.com/schemas/PurchaseOrder.xdr"
33    type="подтверждение"
34    PONumber="10-01-2118">
35       <Item number="122-11" quantity="100" shipped="100"/>
36       <Item number="237-82" quantity="10" shipped ="0"
37       backordered="10"/>
38       <Item number="811-91" quantity="25" shipped="25"/>
39    </PurchaseOrder_1.0>
40    </SOAP-ENV:Body>
41 </SOAP-ENV:Envelope>
```

Табл. 9-3. Описание документа, приведенного в листинге 9-3.

Шаг	Описание
1	Это документ XML, начинающийся с объявления XML.
6—28	Структура элемента *SOAP-ENV:HEADER* практически та же, что и у документа, инициировавшего процесс, но этот документ содержит ответную информацию.
11, 14	Адреса *to* и *from* поменялись местами.
30	Начало элемента *Body* документа **SOAP.**

(*продолжение*)

Шаг	Описание
31–39	Документ-подтверждение использует ту же схему, что и заказ на покупку (NewPO.xml). Это не обязательно; подтверждающий документ может использовать любую другую схему элемента Body. В нашем случае мы указываем назначение документа, определяя значение атрибута *type* в строке 33 как «подтверждение».
40–41	Закрывающие тэги элементов *Body* и *Envelope*.

XML — мощное средство для выполнения бизнес-транзакций. Как вы увидели, XML описывает транзакцию, применяя две независимые схемы: конверт сообщения BizTalk и наш заказ на покупку.

Поскольку BizTalk тесно связан с XML, он совместим с любыми спецификациями бизнес-документов. BizTalk — это открытый стандарт, доступный всем организациям, занятым электронным бизнесом.

Когда компания получит этот конверт по адресу *http://toycar-parts.com*, запускается сценарий обработки заказа BTServer.xar. Он заменяет Дэвида из нашего примера. Сценарий применит те же бизнес-правила, что и Дэвид, и отправит подтверждение в форме документа BizTalk в мою компанию.

Возможности автоматизации

Сценарий, проиллюстрированный документами BizTalk — запросом, подтверждением и ответом, — дублирует ручную обработку, которой долгое время занимались Джейн и Дэвид. Большое достижение нашего сценария — замена ручного труда компьютерными приложениями, позволяющими вести дела быстрее и эффективнее. Разумеется, при замене людей компьютерами ошибки распространяются гораздо быстрее, так что отладка приложений приобретает важнейшее значение.

Описанный здесь подход хорош, но он не позволяет задействовать все преимущества новой технологии. Сервер BizTalk, заменивший Джейн, по-прежнему получает каталоги от поставщиков. (Разумеется, они также являются сообщениями BizTalk, что обеспечивает автоматическое добавление информации в список продукции, которую мы можем заказать.) Они поступают, когда поставщик считает это нужным или когда изменяется номенклатура поставляемой им продукции. Добавьте к этому еще то, что большую часть заказываемых товаров моя компания получает от нескольких поставщиков.

Каждый раз, когда система, используемая моей компанией, принимает решение о размещении заказа у того или иного поставщика, возникает проблема возможного отсутствия нужного товара на складе, и возрастает риск отказа в поставке.

Раньше возможности Джейн по подбору оптимального сочетания цены и условий поставки для каждого вида продукции были ограничены. Она сэкономила бы, увеличив число возможных поставщиков до нескольких дюжин, и сэкономила бы еше больше, увеличив число вариантов до нескольких тысяч.

Если бы каждый поставщик, с которым мы имеем дело, предлагал нам свои каталоги и информацию о наличии продукции на складах *в* режиме реального времени, мы могли бы запросить необходимые данные об интересующей нас продукции перед размещением заказа. Мы могли бы использовать SOAP для того, чтобы быстро запрашивать информацию у тысяч поставщиков. Найдя оптимальное сочетание качества. наличия и цены, мы могли бы сделать заказ, применив BizTalk. Это позволило бы сэкономить немало денег и избежать случаев, когда нужного товара нет на складе.

Подобный сценарий, уже действующий в некоторых компаниях, в корне меняет отношения с поставщиками. Такой процесс называется *обратным аукционом:* поставщики конкурируют за право выполнить заказ, снижая цены. Службы Сети, работающие в режиме реального времени, могут сделать обратный аукцион реальностью. SOAP и BizTalk — открытые стандарты, поддерживаемые любыми платформами, помогут в развитии и широком применении подобной практики.

Часть III

Сервер BizTalk

Сервер BizTalk — это программное обеспечение, обрабатывающее документы BizTalk. В качестве простейшего сервера BizTalk может выступать обычный фильтр, который обрабатывает документ, выделяет определенную информацию и передает ее другому приложению. С другой стороны, в этой роли может выступать сложное приложение масштаба предприятия, которое обеспечивает взаимодействие с бизнес-партнерами, отслеживает проводку документов, преобразует данные и отвечает за взаимодействие с традиционными и новыми системами.

В этой части мы рассмотрим вопросы, встающие при подготовке к разработке собственного сервера BizTalk или при оценке коммерческой реализации. В качестве примера мы создадим два простых сервера BizTalk: для среды Windows DNA и на платформе, отличной от Windows.

Глава 10

Требования к серверу BizTalk

Сервер BizTalk представляет собой ПО, способное считывать и обрабатывать документы BizTalk. Поскольку инфраструктура BizTalk базируется на XML и, как следствие, является самодокументированной, BizTalk-сервер должен поддерживать XML. Иначе говоря, сервер BizTalk должен уметь загружать документы XML и работать с их содержимым. Это лишь первое требование. Естественно, что сервер, умеющий лишь загружать документы XML и работать с их содержимым, не принесет много пользы в поддержке бизнес-процессов. В спецификации документов и сообщений BizTalk Framework 2.0 базовые задачи выполняет *BizTalk-совместимый сервер* (BFC, BizTalk Framework Compliant).

В этой главе я опишу требования, предъявляемые к коммерческой реализации сервера BizTalk и разработке собственного сервера. Если следовать данным здесь определениям, то сервер BizTalk может быть как очень простым, так и очень сложным. В последнем случае на его основе можно организовать практически всю деятельность, связанную с партнерскими отношениями.

BizTalk-сервер можно создать на любом языке программирования и в любой ОС, позволяющих обратиться к XML-анализатору. Сервер BizTalk может работать и в среде Microsoft Windows, и под управлением UNIX. Он может быть создан и на кросс-платформенном языке, таком как Java, Python или Omni Mark.

Сервер BizTalk должен быть достаточно гибок для работы в быстро меняющейся бизнес-среде. Он должен обладать гибким интерфейсом, позволяющим сосуществовать и взаимодействовать с традиционными приложениями. При разработке сервера стоит изучить мне-

ние будущих пользователей. Поскольку пользователи работают в разных системах, сервер должен обеспечивать взаимодействие с разными платформами и разными языками программирования. В этой главе мы обсудим важнейшие функции сервера BizTalk, а в главе 11 рассмотрим примеры его реализации.

Надежная пересылка по любому протоколу

Чтобы использовать документ BizTalk, его надо доставить получателю. Если последний — партнер по бизнесу, требуется система доставки документов. Поскольку мы имеем дело с электронной информацией, разумно использовать средства доставки, передающие биты из одного места в другое.

Проще всего передать эти биты по HTTP, Этот протокол повсеместно распространен и без помех проходит практически через все брандмауэры. Однако не все партнеры имеют возможность получать запросы по HTTP, а некоторые опасаются передавать по нему незащищенные данные. Альтернативой служит протокол FTP, поддерживающий аутентификацию по имени пользователя и паролю. В этом случае риск получить фальшивое сообщение значительно снижается. Если партнер не поддерживает FTP, можно использовать электронную почту, поскольку сервер BizTalk должен поддерживать передачу сообщений по *упрощенному протоколу передачи почтовых сообщений* (Simple Mail Transport Protocol, SMTP). Сервер SMTP передаст сообщения любому адресату в Интернете.

Конечно, если партнеры по бизнесу все еще живут в XX веке, у них может не оказаться даже электронной почты. Поэтому BizTalk-сервер должен поддерживать и самые примитивные способы доставки документов. Тут наибольшую гибкость обеспечивает передача документов по факсу. Кроме того, удобно иметь возможность распечатать документы на принтере, чтобы затем переслать их по обычной почте. Эти «протоколы» предназначены для доставки документов в удобном для чтения виде, поэтому сервер BizTalk должен уметь представлять деловые документы в доступной человеку форме.

Основная проблема при работе по HTTP, FTP и SMTP в том, что они не защищены. К тому же эти протоколы используют в качестве среды доставки Интернет, и информация может попасть к любому, кто способен перехватывать пакеты TCP. Итак, мы подошли к следующему требованию — к защите информации.

Защита информации

Передача документов BizTalk по HTTP потенциально опасна. Поскольку документы HTTP представляют собой потоки текстовой ин-

формации, то кто угодно по пути следования документа может прочитать сообщение. Спецификация BizTalk Framework 2.0 позволяет защитить документ BizTalk с помощью протоколов S/MIME и PKCS. Кроме того, можно задействовать и дополнительные уровни защиты.

Проблему защиты транзакций легко решить, применив протокол HTTPS (буква S как раз и означает защиту информации). Этот протокол широко применяется в Интернете для защиты транзакций, затрагивающих важную информацию, например, банковские счета и номера кредитных карточек. При передаче пакетов HTTPS между клиентом и сервером используется асимметричное шифрование. Когда клиент — обычно обозреватель — запрашивает защищенный документ, сервер Web передает ему ключ. Этот ключ позволяет клиенту шифровать всю исходящую информацию в течение сеанса. Расшифровать данные, поступившие от клиента, может только сервер. Асимметричное шифрование основано на сложной математике с применением умножения больших простых чисел. HTTPS прекрасно работает, а степень зашиты определяется длиной ключа. Для защиты очень важной информации целесообразно использовать самые длинные ключи.

Сервер BizTalk должен защищать пересылаемый документ. Кроме того, нужно решить проблему получения партнером по бизнесу фальшивых документов. Например, кому-нибудь может прийти сообщение от вашего имени, отправленное не вами. Такое часто случалось на заре электронной почты. Решить эту проблему позволяет поддерживаемая BizTalk-сервером электронная подпись, которая добавляется к документу. Для этого предназначен описанный в спецификации документа BizTalk элемент *вложение:* кроме того, можно выбирать способ добавления электронной подписи в зависимости от способа доставки документа получателю.

Маршрутизация документов

Наиболее важная способность сервера BizTalk заключается в передаче документов из одной системы в любую другую. Это может быть сделано просто, например, путем передачи запроса Web-серверу по HTTP, а может иметь очень сложную реализацию, учитывающую прохождение документов через последовательность брандмауэров, и т. п. Кроме того, полученный от партнера ответ обычно нужно автоматически переадресовать в соответствующее подразделение компании.

Рассмотрим пример. Придя в муниципальное учреждение, посетитель сперва заполняет необходимую форму. Одни формы связаны с оплатой транспорта и парковок, другие предназначены для получе-

ния разрешения на пристройку флигеля к дому. Есть формы для заключения брака и т. д. Чиновник, работающий в приемной, получает форму и направляет ее в соответствующий отдел. Это избавляет посетителя от размышлений над тем, в какую комнату и на каком этаже ему надо идти с заполненной формой.

Сервер BizTalk — тот же чиновник из приемной: он должен, распознав тип поступившего документа, направить его на обработку. Маршрутизация документов обычно включает стадию соответствующей обработки документов.

Документооборот

Полученный документ BizTalk должен быть проанализирован, после чего приложение сможет выяснить его тип. Например, приложению могут понадобиться данные из полей *address* и *state*, на основании чего будет сделан вывод, откуда поступил документ и как его обрабатывать. Поле *address* — лишь часть механизма, определяющего тип обработки, которую BizTalk-сервер должен выполнить при поступлении документа.

Выяснив тип документа и его назначение, можно приступать к обработке. К примеру, чтобы обработать заказ на покупку товара, нужно сначала определить, работали ли мы с этим заказчиком ранее. Если да, проверить, нет ли у него задолженности. Затем проверяется наличие запрашиваемого товара на складе. Если товара нет, то для выполнения заказа его следует купить или произвести.

Для всех входящих документов сервер BizTalk должен иметь несколько вариантов их обработки. Серверы высокого класса должны включать средства создания потоков обработки и управления ими. При этом приложение должно обеспечивать простой доступ к этим возможностям, чтобы пользователи могли эффективно управлять работой системы.

Синхронная и асинхронная связь

В главе 9 мы рассмотрели структуру сообщений BizTalk. Сообщения пересылаются партнерам несколькими способами. Созданное сообщение BizTalk нужно поместить в конверт, соответствующий способу передачи или, иначе говоря, транспортному протоколу.

Сервер BizTalk должен иметь возможность синхронной отправки сообщений. При синхронной связи сервер посылает сообщение получателю и ждет ответа. Синхронная связь — основной тип обмена сообщениями и обязательно должна поддерживаться сервером.

Однако реальному приложению иногда требуется посылать десятки, сотни, а то и тысячи сообщений. Если отслеживать подтвержде-

ние каждого запроса, системе может просто не хватить ресурсов. Поэтому сервер должен поддерживать и асинхронный метод передачи.

Очередь сообщений

Сервер BizTalk должен обеспечивать постановку сообщений в очередь, чтобы иметь возможность переключиться на выполнение другой задачи до окончания обработки сообщения. Вообще постановка сообщений в очередь нужна для разгрузки системы передачи. Однако при этом BizTalk-сервер должен иметь возможность проверять статус очереди. Например, если сервер помещает сообщение в очередь, а затем сразу переключается на выполнение следующего задания, потом он должен суметь узнать, успешно ли отправлено сообщение.

Пакетный режим

Документы BizTalk могут создаваться круглые сутки. Некоторые партнеры могут потребовать присылать им транзакции только раз в день в определенное время или только в определенные дни месяца. BizTalk-сервер должен поддерживать назначение параметров доставки документов BizTalk и их передачу в пакетном режиме в соответствии с требованиями партнера.

Отслеживание документов

Применяя механизм очередей, крайне важно сообщать серверу о сбоях. Скажем, сервер должен знать, что передача данных не удалась, чтобы повторить или отменить транзакцию. Кроме того, желательно обеспечить оператору возможность отслеживать статус сообщений и просматривать статистику обработки.

Сервер BizTalk должен иметь развитые возможности отслеживания выполнения транзакций, включая:

- отслеживание отдельных документов и пакетов документов;
- выявление неверно доставленных или недоставленных документов;
- возможность опрашивать БД протоколирования по отдельным документам, по партнерам и по виду деятельности;
- протоколирование времени выполнения всех действий с документами;
- хранение данных, выбранных пользователем, например, общей суммы заказов, на основе которых можно создавать отчеты о работе с партнерами.

Все эти данные должны быть доступны в течение всего жизненного цикла документа и даже по его окончании, если нужно создавать отчеты о транзакциях или есть вероятность претензий по выполненным транзакциям.

Управление взаимодействием с партнерами

Составляя контракт с партнером, нужно учитывать такие особенности, как вид платежа, скидки, способ доставки и т. п. Все эти факторы в конечном итоге описывают взаимодействие с партнером.

Для любого взаимодействия BizTalk-сервер должен уметь отслеживать все аспекты создания и обработки транзакций именно с этим партнером. Кроме того, он должен уметь различать партнеров, с которыми уже налажены деловые отношения, и новых. Например, при проведении переговоров с потенциальным партнером вы должны иметь возможность смоделировать работу с ним и проверить, соответствуют ли возможности вашего сервера BizTalk требованиям партнера.

Масштабируемость

Сегодня все системы должны быть масштабируемыми, и сервер BizTalk не исключение. Начинать масштабирование желательно с транспортного протокола. Кроме того, стоит разместить всю бизнес-логику на среднем уровне и обращаться к ней по интрасети.

Сервер BizTalk должен обеспечивать выполнение необходимого числа транзакций в день с учетом перспектив расширения бизнеса. При этом следует учитывать не количество нынешних партнеров, а число компаний, с которыми потенциально можно наладить связи, Если планируется активно использовать Web-технологии, следует ожидать бурного роста передаваемой деловой информации. По сравнению с традиционными реализациями электронного обмена данными затраты на XML и BizTalk невысоки. Это позволит без труда наладить связи с большинством бизнес-партнеров. Более того, низкая себестоимость электронных транзакций способствует расширению круга партнеров.

Преобразование документов

В каждой отрасли для взаимодействия с партнерами используются свои схемы. Так. ассоциация страховых компаний ACORD (*http://www.acord.com)* разрабатывает собственные схемы страховых документов, включая страховые свидетельства и формы для описания страховых случаев. Работа турагентств связана со стандартами бронирования номеров в гостиницах и рейсов на самолет. Если ваш бизнес связан с этими отраслями, нужно предусмотреть возможность считывания документов этого типа из конверта BizTalk.

По всей книге я старался провести красной нитью идею бизнеса на стыке отраслей, при котором вполне вероятно, скажем, получить заказ от автомобильной компании, оформленный по стандартам ав-

томобильной отрасли. Структура такого заказа будет сильно отличаться от заказа, оформленного по стандартам авиационной индустрии и от заказа транспортной фирмы. Конечно, в заказах есть что-то общее, но структура будет абсолютно разной. Наш подход кардинально отличается от стандартного подхода EDI, требующего единой универсальной структуры заказа, без точного соблюдения которой заказ не будет обработан. Сервер BizTalk должен иметь доступ к документам любого типа. Для этого можно написать программу, преобразующую запрос в формате автомобильной индустрии в запрос, поддерживаемый сервером BizTalk, или. облегчив задачу, применить XSLT-преобразование.

Как говорилось в главе 6, оптимальный метод преобразования документов — шаблоны XSLT: ведь XSLT работает прямо с XML и создает XML-документ нужной структуры. Преобразование XSLT можно представить в виде функции, отображающей определенные элементы входящего документа в определенные элементы исходящего.

Сервер BizTalk должен уметь различать разные типы входящих документов и автоматически выполнять необходимые преобразования. Должна быть предусмотрена возможность выбора преобразований для каждого партнера, с которыми работает ваш сервер BizTalk.

BizTalk-сервер корпоративного масштаба должен поддерживать создание XSLT-преобразований, проверку их работоспособности и возможность их модификации.

Встраивание модулей других производителей

Сервер BizTalk должен быть открыт для встраивания расширений, повышающих эффективность взаимодействия с партнерами из конкретных отраслей или между определенными группами партнеров.

Такие интерфейсы могут создать любые сторонние фирмы и независимые поставщики ПО, желающие участвовать в развитии инфраструктуры BizTalk. Сервер BizTalk должен поддерживать такие интерфейсы и тем самым привлекать сторонних разработчиков в эту область.

Сервер должен уметь работать со схемами от сторонних разработчиков и предоставлять процессам, взаимодействующим с сервером BizTalk. интерфейс для доступа к схемам.

Взаимодействие с EDI-системами

Реализация традиционных EDI-систем требует высокой гибкости и широких возможностей интеграции. Многие компании уже создали собственные EDI-системы для работы с партнерами. Несмотря на сложность и дороговизну, эти системы работают, и ваш сервер должен уметь взаимодействовать с ними.

Для компаний, где используются EDI-системы, переход на XML и BizTalk может оказаться неприемлемым, но работать с такими компаниями (обычно крупными) все равно нужно. Поэтому сервер BizTalk должен уметь обрабатывать EDI-транзакции. Для этого можно либо просто создать таблицу преобразования документов EDI в документы BizTalk, либо поступить более формально и разработать подсистему, поддерживающую EDI-транзакции и взаимодействующую с сервером BizTalk.

Прикладные интерфейсы

Сервер BizTalk — это просто механизм, который создает и отслеживает документы BizTalk. В его задачи не входит поддержка всего документооборота компании. Сервер BizTalk должен обладать хорошо развитым интерфейсом, обеспечивающим взаимодействие с другими приложениями.

Прикладные интерфейсы сервера BizTalk должны представлять серверным процессам средства доступа к документам. В роли серверных процессов могут выступать система отслеживания выполнения заказов, бухгалтерская система, система контроля производственных процессов и любые другие системы, требующие взаимодействия с партнерами.

Адаптация к изменениям

BizTalk-сервер должен быть достаточно гибким, чтобы справиться с проблемами, возникающими при работе с разнообразными партнерами, и с изменением условий. Например, если сейчас заказчик использует **EDI,** но планирует переход на XML, ваш BizTalk-сервер должен обеспечить такой переход без существенного изменения партнерского соглашения. При этом должен поменяться лишь способ взаимодействия с партнером — все остальное должно остаться неизменным.

Ваш сервер BizTalk должен поддерживать расширение стандартных схем, применяемых в конкретной отрасли. Если вы работаете с десятками компаний, по мере появления новых схем вы должны иметь возможность немедленно использовать их для взаимодействия с партнерами. С другой стороны, если для работы с какой-то компанией нужна специализированная версия стандартной схемы, ваш сервер должен ее поддерживать.

XML позволяет добавлять в документы собственные поля, не нарушая схемы. Если партнеру эти поля не нужны, он их просто проигнорирует. Однако сервер BizTalk должен уметь управлять такими полями и учитывать их при создании специализированных представлений информации для каждого партнера.

Кроме того, ваш сервер должен учитывать возможность изменения параметров партнеров. Система должна быть готова к тому, что два партнера могут слиться в одну компанию, а один — разделиться на несколько компаний. Кроме того, не надо сбрасывать со счетов и то, что та же участь может постигнуть и компанию, где работает сервер BizTalk.

Ориентация на нужды пользователей

Поскольку сервер BizTalk предназначен для поддержки взаимодействия с партнерами, работать с ним будут не программисты, а простые пользователи. Поэтому они должны располагать понятными средствами ввода информации о партнерах и настройки параметров обработки документов.

Я не говорю, что программисты должны остаться в стороне от установки и поддержки сервера BizTalk. Напротив, на их долю выпадет непростая задача по интеграции сервера с бизнес-приложениями и традиционными системами.

И все же для создания соглашения с новым партнером нет нужды обращаться к программистам. Сервер должен обладать достаточно развитым интерфейсом, позволяющим пользователям самостоятельно выполнять подобные операции: создавать соглашения и изменять их параметры, описывать преобразование входящих и исходящих документов, изменять структуру данных (если это не влияет на работоспособность системы), создавать отчеты о работе с конкретными партнерами и о функционировании системы в целом.

И, наконец, система должна поддерживать удаленный доступ пользователей. Для этого, очевидно, понадобятся различные технологии среднего уровня, включая защиту соединений с применением шифрования.

Глава 11

Создание сервера BizTalk

Папа как-то сказал мне, что преодоление трудностей закаляет характер. Он говорил, что если л заработаю деньги сам, то буду ценить их больше, чем если бы просто получил их от него. Он, конечно, был прав, но мне почему-то казалось, что он просто не хочет давать двадцатку, которую я попросил.

В этой главе мы будем закалять характер, разрабатывая BizTalk-сервер самым сложным, но и самым полезным способом — вручную. Чтобы проиллюстрировать процесс создания сервера BizTalk, стоит поработать на уровне данных, составляя программы, которые создают документы BizTalk. Эти программы передают документы обрабатывающему их серверу BizTalk.

Мы рассмотрим процесс обработки заказа из главы 9. Это относительно простой пример, но он наглядно демонстрирует все концепции, понимание которых понадобится вам при создании собственного сервера BizTalk или при выборе подходящего продукта стороннего поставщика. Вы увидите также, насколько инфраструктура BizTalk упрощает обработку транзакций.

Так как инфраструктура BizTalk является расширением SOAP, для обмена данными между компаниями из нашего примера мы задействуем именно SOAP. Кроме того, мы используем *представительский сервис удаленных объектов* (Remote Object Proxy Engine, ROPE). Эта составная часть пакета разработчика Microsoft SOAP Toolkit для Visual Studio 6.0 реализует средства поиска Web-служб и обращения к ним через простой COM-интерфейс.

В нашем примере участвуют две компании, упомянутые в главе 9: Toy Carz, Inc. *(www.Toycarz.com)* и Toy Car Parts Corporation *(www.toycarparts.com)*. Как вы помните, Toy Carz производит игрушечные автомобили, а Toy Car Parts — один из ее поставщиков. Toy Carz использует платформу Windows, технологию активных серверных стра-

ниц (Active Server Pages, ASP) и Microsoft Visual Basic. Toy Car Parts использует серверы под управлением UNIX (Linux), Web-сервер Apache и OmniMark для создания трехуровневых Web-приложений.

В этом примере я хочу показать, как системы, написанные на разных языках программирования и работающие на компьютерах под управлением разных ОС, могут обмениваться сообщениями SOAP и BizTalk. Примеры будут работать в средах, которые я перечислил, но для большей наглядности и простоты я использую систему из нашего примера в среде Microsoft Windows.

Все примеры этой главы можно выполнить на одном компьютере. Я сделал это по нескольким причинам. Во-первых, это заметно упрощает вашу работу, поскольку вам не понадобится искать два свободных компьютера. Во-вторых, большую часть этой книги я писал в самолетах, вдали от каких бы то ни было сетей, поэтому я проектировал системы, которые легко можно переместить на любые Windows-компьютеры (Боже, благослови мой IBM ThinkPad!). Все, что вам потребуется сделать, — перенести программы и изменить указатели ресурсов в каждой из них. Модификация программ на OmniMark для работы под Unix не столь тривиальна, но и не слишком сложна. Самые большие изменения, которые вам придется внести в Unix-программы, связаны с доступом к базам данных. В этих примерах я использую интерфейс ODBC (Open Database Connectivity) для Windows. OmniMark поддерживает не только ODBC, но и аналогичный интерфейс OCI (Oracle Command Interface) для UNIX-платформ. Интерфейс OCI очень схож с ODBC и требует лишь косметических изменений в исходном коде.

Весь исходный код примеров этой главы приведен в каталоге \Samples\Chl 1 компакт-диска книги. Систему, работающую, как описано в этой главе, можно построить, следуя указаниям раздела «Установка». Подробно комментировать код примеров я не буду — полагаю, что вы умеете программировать на Visual Basic,

Однако я собираюсь «покопаться» в работе программ на OmniMark, гак как эгот язык многим читателям незнаком. Если вы хотите быстро освоить его, обратитесь к приложению А. С другой стороны, возможно, вам хватит моих объяснений, приведенных в этой главе. На компакт-диске приведена электронная копия моей книги «Работа с OmniMark. Том 1: Приступая к работе» *(OmniMark at Work, Volume 1: Getting Started)*. Это сэкономит вам 65 долларов; кстати, при встрече можете угостить меня пивом. OmniMark — отличный язык для разработки сетевых приложений, и вы не пожалеете о времени, затраченном на его изучение.

Информационные потоки

Давайте взглянем на схемы, иллюстрирующие работу нашего примера. Рис. 11-1 и 11-2 демонстрируют информационные потоки между компаниями Toy Carz и Toy Car Parts. Как было сказано в главе 9, именно в этих компаниях работают Джейн и Дэвид, которые создают и обрабатывают бизнес-документы. Их мы и собираемся заменить двумя компьютерными системами.

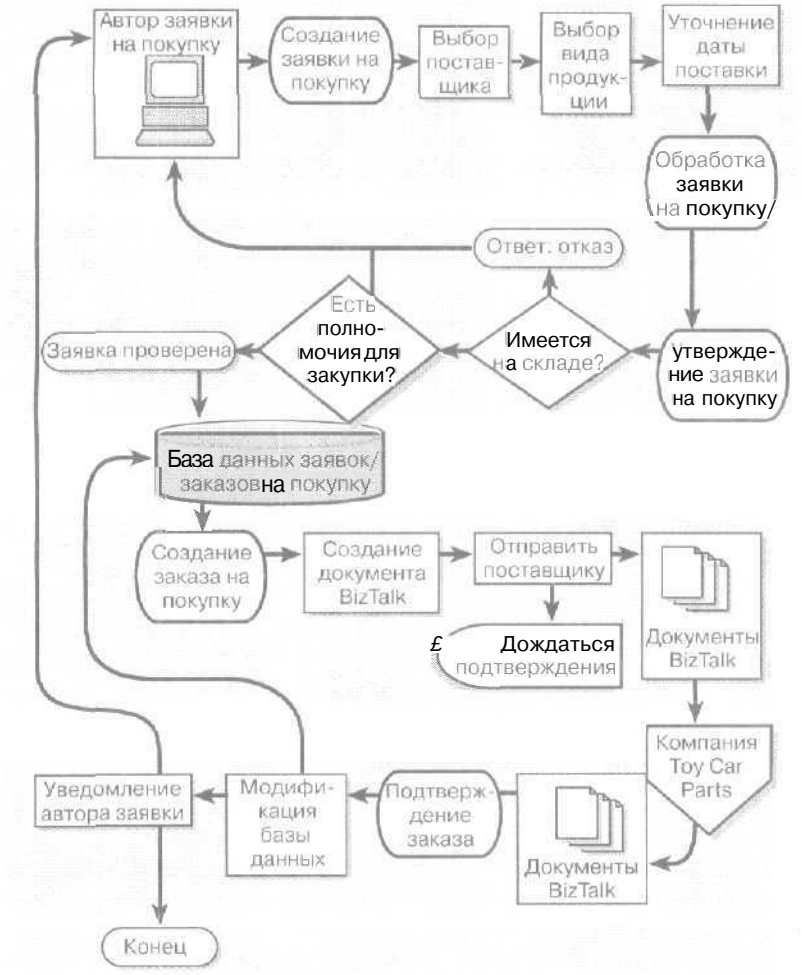


Рис. 11-1. Процесс прохождения заявки на закупку комплектующих.

Процесс протекает так: некое событие инициирует начало обмена информацией между двумя компаниями. Событие состоит в том, что служащий Toy Carz хочет закупить какую-то продукцию в компании Toy Car Parts (рис. 11-1). Он создает заявку на покупку, для чего выбирает поставщика, вид продукции и предпочтительную дату доставки.

Эти данные обработаны приложением, которое применяет бизнес-правила согласно внутренней политике Toy Carz. Сначала приложение выясняет через базу данных (БД) служащих, обладает ли автор заявки полномочиями для закупки требуемого вида продукции. Затем приложение проверяет наличие данной продукции на складе. Если продукция имеется в наличии, общее количество в заявке соответствующим образом уменьшается. Если проверка полномочий и наличия на складе прошли успешно, заявка на покупку помешается в БД. Эта база содержит все заявки на покупку, прошедшие подобную проверку.

Через определенные промежутки времени БД заявок на покупку проверяется с целью создания заказов на покупку для отправки поставщикам. Система, выполняющая эту операцию, просматривает БД поставщиков, выбирает их адреса и формирует документы BizTalk с соответствующим заказом. Эти документы отправляются поставщикам, одним из которых является компания Toy Car Parts. Программа, работающая в этой компании, и поток данных изображены на рис. 11-2.

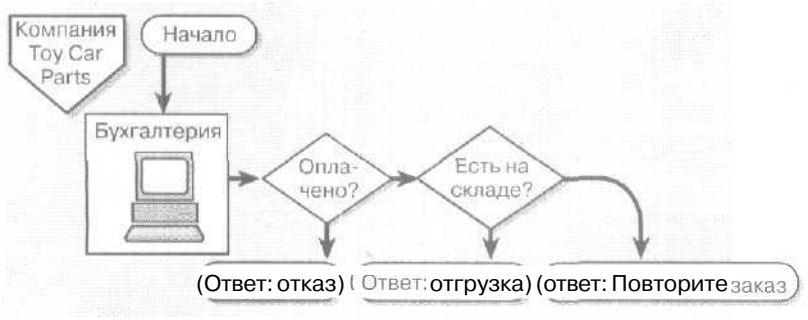


Рис. 11-2. Процесс обработки заказа в компании Toy Car Parts.

Получив документ BizTalk, программа, работающая в компании Toy Car Parts, обрабатывает его, выбирая нужную информацию, включая данные о заказе. При этом также проверяется статус заказчика (наличие договора с Toy Car Parts и своевременность оплаты выставляемых счетов).

Затем программа проверяет наличие заказанной продукции на складе. В случае ее отсутствия заказчику направляется сообщение о

необходимости повторить заказ через некоторое время. Если же продукция в наличии, соответствующая запись помешается в БД заказов, а содержимое инвентарной БД корректируется. И, наконец, заказчику отправляется документ BizTalk с подтверждением заказа.

Вернемся к рис. 11-1. Компания Toy Carz получает ответный документ BizTalk от компании Toy Car Parts. Приложение ASP, получившее этот документ, извлекает нужную информацию и выполняет две операции, завершающие процесс заказа: вносит в БД заказов сведения о том, что заказ сделан и подтвержден поставщиком, и сообщает об этом автору заявки.

Архитектура

В процессе обработки заказа компании Toy Carz и Toy Car Parts используют ряд программ, описанных в табл. 11-1 и 11-2. Тексты программ, применяемых в Toy Carz, см. в каталоге Samples/Ch11/ToyCarz, а программы Toy Car Parts — в каталоге /Samples/Chi 1/ToyCarParts компакт-диска книги.

Табл. 11-1. Программы, используемые в процессе создания заказа в компании Toy Carz.

Название	Тип	Описание
prEnter.asp	ASP	Создает заявку на покупку, связываясь с базами данных служащих и поставщиков для формирования списка продукции, доступной для заказа. Заявку обрабатывает программа prProcess.asp.
prProcess.asp	ASP	Обрабатывает заявку на покупку: сначала проверяет полномочия служащего на заказ продукции, а затем — наличие заказываемой продукции на складе.
poGen.exe	VB	Создает заказ: считывает из БД заявки на покупку и генерирует заказы, отправляемые поставщикам, один из которых — компания Toy Car Parts, которая и получит заказ. Эта программа использует ROPE для упаковки и отправки документа SOAP серверу SOAP.
poConfirm.asp	ASP	Получает документ BizTalk, содержащий подтверждение заказа на покупку; заносит в БД заказов подтверждение приема заказа и извещает служащего, сделавшего заявку, что его заявка обработана.

Табл. 11-2. Программы, обрабатывающие заказ в компании Toy **Cars Parts.**

Название	Тип	Описание
PlaceOrder.htm	HTML-страница	У компании Toy Car Paris есть «приложение» для ввода информации о заказах: при поступлении заказа служащий вводит соответствующие данные в форму на Web-странице.
PlaceOrder.xom	Omni-Mark	Сценарий CGI, указанный в атрибуте ACTION элемента FORM документа PlaceOrder.htm, Содержит бизнес-логику, проверяющую соответствие заказа внутренним правилам контроля. В случае удачной проверки программа помещает заказ в БД заказов. Toy Car Parts не хочет переписывать эту систему, успешно работающую долгие годы, так что BizTalk-сервер должен взаимодействовать с этим приложением.
BTServer.xom	Omni-Mark	Omni Mark-программа, оформленная как сценарий CGI. Сначала она выполняет роль сервера SOAP, получая документ SOAP, отправленный через интерфейс ROPE программой poGen.exe. Затем программа BTServer.xom «превращается» в сервер BizTalk и преобразует запрос в формате XML в документ, эмулирующий результат заполнения формы PlaceOrder.htm. Получив ответ от программы PlaceOrder.xom, она создает BizTalk-подтверждение заказа и перелает его Toy Carz методом POST активной страницы poConfirm.asp. Эта программа обрабатывает подтверждение, чем и заканчивает цикл обработки заказа.

Полный цикл обработки заказа показан на рис. 11-3.

Базы данных

Каждая компания использует свой набор БД для хранения сведений о продукции, заказчиках и состоянии заказов. Эти БД описаны в табл. 11-3 и 11-4. Инструкции по их установке см. в разделе «Установка системы» далее в этой главе. Это БД Microsoft Access, и вы можете открыть их и изучить структуру данных и связи.

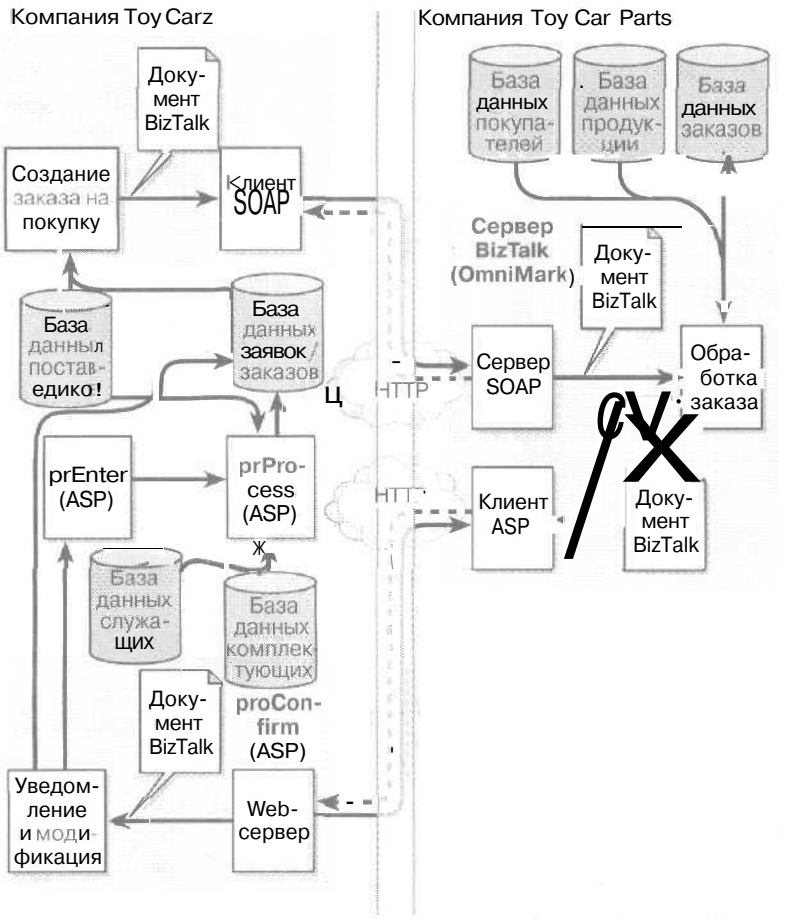


Рис. 11-3. Картина в целом: три сервера BizTalk, которые мы создадим в этой главе.

Табл. 11-3. Базы данных компании **Toy Carz.**

База данных	Описание
Employees.mdb	Содержит список служащих и их полномочий.
Inventory.mdb	Основная БД; содержит сведения обо всех вида продукции, имеющихся в наличии.

(продолжение)

База данных	Описание
Vendors.mdb	Список видов продукции, поставляемых всеми поставщиками, с которыми компания связана договорами. Эта БД модифицируется всякий раз, когда компания получает каталог от существующего поставщика или подписывает соглашение с новым поставщиком.
PurchaseOrders.mdb	Содержит две таблицы: PORequests (все заявки на покупку, созданные служащими) и POSent (все заказы, отправленные поставщикам).

Табл. 11-4. Базы данных компании Toy Car Parts.

База данных	Описание
Customers.mdb	Список всех заказчиков и состояние их счетов. Эта БД используется для проверки оплаты заказчиком текущих счетов перед отправкой ему следующей партии продукции.
Inventory.mdb	Основная БД с информацией о продукции, имеющейся в наличии, ее количестве и цене.
Orders.mdb	Хранит информацию о заказах, которую заносит программа PlaceOrder. Эта БД интегрирована с системами, распоряжающимися отгрузкой товара и ведущими учет.

ПРИМЕЧАНИЕ БД PurchaseOrders.mdb содержит ссылки на две таблицы БД Vendors.mdb: «catalog» и «vendors». Если вы установили базы данных в каталог, отличный от принятого по умолчанию (C:\TravisBook\Ch11\ToyCarz\), исправьте ссылки средствами Access.

Система обработки заказов в нашем примере использует ODBC для доступа к БД различных производителей. Все БД, описанные в табл. 11-3 и 11-4, — простые базы Microsoft Access, однако в этой роли могут выступать и БД Microsoft SQL Server или Oracle. Разумеется, подобные изменения потребуют модификации исходного кода, но не слишком кардинальной.

Установка системы

Для установки системы обработки заказов необходимо выполнить следующие шаги.

1. Запустите программу установки с компакт-диска книги для копирования файлов примера на ваш жесткий диск или скопируйте все файлы из каталога \Samples\Chl 1 в нужный каталог. По умолчанию программа установки копирует файлы в каталог C:\TravisBook\Ch11.
2. Создайте имена источников данных ODBC Microsoft Windows 2000 для баз данных нашего примера. DSN играет роль дескриптора для доступа к файлам БД. В Панели управления откройте папку Administrative Tools (Средства администрирования), найдите значок Data Sources (ODBC) (Источники данных ODBC), дважды щелкните его, откройте диалоговое окно ODBC Data Source Administrator (Администрирование источников данных ODBC). В других версиях Windows это приложение запускается двойным щелчком значка ODBC Data Sources (Источники данных ODBC) Панели управления.
3. Вам нужно будет создать 8 системных источников данных, описанных в табл. 11-5. На вкладке System DSN щелкните кнопку Add, выберите драйвер Microsoft Access Driver (*.mdb) и щелкните кнопку Finish. Введите имя источника данных из табл. 11-5, нажмите кнопку Browse и найдите файл БД (вы скопировали его на диск в п. 1). Нажмите кнопку OK. Повторите этот шаг для каждой БД из перечисленных в табл. 11-5.
4. Создайте две общих Web-папки. Из Проводника откройте папку C:\TravisBook\Ch11 (или ту папку, куда вы скопировали файлы). Вы увидите две папки, по одной для каждой компании. Щелкнув правой кнопкой папку ToyCarz, выберите в контекстном меню команду Sharing (Доступ). Откройте вкладку Web Sharing. Установите флажок Share This Folder (Разрешить совместное использование) и согласитесь со значениями по умолчанию в диалоговом окне Edit Alias (Редактировать название). Повторите эти шаги для папки ToyCarParts; для нее также нужно установить флажок Execute (Includes Scripts) [Исполнение (Включая сценарии)] — это позволит нам запускать программы на OmniMark в папке ToyCarParts.

Табл. 11-5. Информация для создания имен источников данных.

Имя источника данных	Местонахождение
dbTC-Employees	C:\TravisBook\Ch11\ToyCarz\Employees.mdb
dbTC-Inventory	C:\TravisBook\Ch11\ToyCarz\Inventory.mdb

(продолжение)

Имя источника данных	Местонахождение
dbTC-PurchaseOrders	C:\TravisBook\Ch11\ToyCarz\PurchaseOrders.mdb
dbTC-Vendors	C:\TravisBook\Ch11\ToyCarz\Vendors.mdb
dbTCP-Customers	C:\TravisBook\Ch11\ToyCarParts\Customers.mdb
dbTCP-Inventory	C:\TravisBook\Ch11\ToyCarParts\Inventory.mdb
dbTCP-Orders	C:\TravisBook\Ch11\ToyCarParts\Orders.mdb

Ввод заявки на покупку

Для ввода заявки на покупку используются две программы: prEnter.asp и prProcess.asp. Им посвящен этот раздел.

prEnter.asp

Эта активная страница предоставляет доступ к БД Vendors, которая содержит список поставщиков. На рис. 11-4 показана эта страница, когда пользователь выбрал данные.

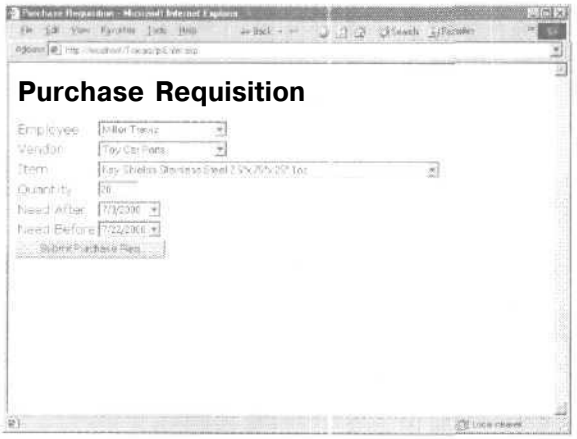


Рис. 11-4. С помощью страницы оформления заявки prEnter.asp служащий инициирует процесс заказа.

Программа считывает записи из БД Employees и Vendors и заполняет раскрывающиеся списки. Когда поставщик выбран, ASP-страница автоматически модифицирует раскрывающийся список комплектующих. Код программы приведен в листинге 11-1.

Листинг **11-1 Программа ввода заявки на** покупку **prEnter.asp.**

```
prEnter.asp
<%@language=vbscript%>
<%
Set oVendorConn = Server.CreateObject("ADODB.Connection")
oVendorConn.Open("dbTC-Vendors")
Function dateRange(fieldName, startDate, daysBackward, daysForward)
   If Request.Form(fieldName) - "" Then
      selectDate = FormatDateTime(startDate, vbShortDate)
   Else
      selectDate - Request.Form(fieldName)
   End If
   dateRange = vbCrLf & "<SELECT NAME='" & fieldName & "'>"
   Fo' i - -daysBackward Tc daysForward
      thisDate = FormatDateTime(startDate + i, vbShortDate)
      dateRange = dateRange & vbCrLf & vbCrLf & "<OPTION "
      If thisDate = selectDate Then dateRange = dateRange & _
         "SELECTED"
      dateRange = dateRange & " VALUE='" & thisDate & "'>" _
         i thisDate & "</OPTION>"
   Next
   dateRange = dateRange & vbCrLf & "</SELECT>"
End Function
%>
<HTML>
   <HEAD>
   <TITLE>Purchase Requisition</TITLE>
   <SCRIPT>
      function loadItems()
      {
         document.all.vendorForm.action="prEnter.asp"
         document.all.vendorForm.submit()
      }
      </SCRIPT>
   </HEAD>
   <BODY STYLE="font-family:Verdana;">
   <H1>Purchase Requisition</H1>
   <FORM METHOD="POST" id="vendorForm" ACTION="prProcess.asp">
      <TABLE>
```

```
<TR>
    <TD>Employee</TD>
    <TD>
    <SELECT NAME="requestorID">
    <OPTION VALUE="0">- Select an Employee -</OPTION>"
    <%
    Set oConn = Server.CreateObject("ADODB.Connection")
    oConn.Open("dbTG-Employees")
    Set rsEmployee - oConn.Execute( _
        "SELECT * FROM Employees ORDER BY LName")
    Do Until rsEmployee.EOF
        Response.Write ("<OPTION ")
        If Int(Request.Form("requestorID")) = _
                Int( rsEmployee("empID")) Then
                Response.Write (" SELECTED")
        End If
        Response.Write (" VALUE='" _
                & rsEmployee("empID") & "'>"_
                & rsEmployee("FName") & " " _
                S rsEmployee("LName") _
                S "</OPTION>")
        rsEmployee.MoveNext
    Loop
    Set rsEmployee = Nothing
    oConn.Close
    Set oConn - Nothing
    ft>
    </SELECT>
    </TD>
</TR>
<TR>
    <TD>Vendor</TD>
    <TD>
    <SELECT ID="vendorID" NAME="vendorID"
        ONCHANGE="javascript:loadItems();">
    <OPTION VALUE="0">- Select a Vendor -</OPTION>
    <%
    Set rsVendor = oVendorConn.Execute( _
        "SELECT * FROM Vendors ORDER BY vendorName")
    Do Until rsVendor.EOF
```

```
                Response.Write ("<OPTION ")
                If Int(Request.Form("vendorID")) - _
                        Int(rsVendor("vendorID")) Then
                        Response.Write (" SELECTED")
                End If
                Response.Write (" VALUE='" _
                        & rsVendor("vendorID") & "'>" _
                        & rsVendor("vendorName") & "</OPTION>")
                rsVendor.MoveNext
            Loop
            Set rsVendor = Nothing
            %>
            </SELECT>
            </TD>
        </TR>
        <TR>
            <TD>Item</TD>
            <TD>
            <SELECT NAME="itemNum">
            <OPTION VALUE="0">- Select an Item -</OPTION>
            <%
            If Request.Form("vendorID") <> 0 Then
                Set rsItem - oVendorConn.Execute( _
                        "SELECT - FROM catalog WHERE vendorID=" _
                        & Request.Form("vendorID") _
                        & " ORDER BY description")
                Do Until rsItem.EOF
                        Response.Write(vbCrLf & "<OPTION ")
If Request.Form("itemNum") = rsItem("itemNum") Then
                                Response.Write(" SELECTED")
                        End If
                        Response.Write(" VALUE='" _
                                & rsItem("itemNum") 5 "'>"_
                                & rsItem("description") _
                                & ". " S rsItem("color") & "</OPTION>")
                        rsItem.MoveNext
                Loop
                set rsItem - Nothing
            End If
```

```
            %>
            </SELECT>
            </TD>
        </TR>
        <TR>
            <TD>Quantity</TD>
            <TD><INPUT NAME="itemQty" VALUE="1" SIZE="5">
                </INPUT></TD>
        </TR>
        <TR>
            <TD>Need After</TD>
            <TD><%=dateRange("needAfter", Now(), 0, 30)%></TD>
        </TR>
        <TR>
            <TD>Need Before</TD>
            <TD><%=dateRange("needBefore", Now() + 21, 20, 30)%></TD>
        </TR>
    </TABLE>
    <INPUT TYPE="SUBMIT" VALUE="Submit Purchase Req"></INPUT>
</FORM>
</BODY>
</HTML>
<%
oVendorConn.Close
Set oVendorConn = Nothing
%>
```

Событие *ONCHANGE* раскрывающегося списка поставщиков вызывает Jscript-функцию *loadItems*, которая временно изменяет метод формы, вызывая перезагрузку страницы. При этом раскрывающийся список заполняется комплектующими в соответствии с выбранным поставщиком- При нажатии кнопки Submit страница prEnter.asp вызывает программу prProcess.asp, проверяющую заполнение заявки.

prProcess.asp

Содержит бизнес-логику проверки наличия у служащего полномочий для заказа данного вида продукции. Программа осуществляет и другие проверки, включая выяснение наличия товара на складе. Результат ее работы в случае успеха — подтверждение заявки — показан на рис. 11-5.

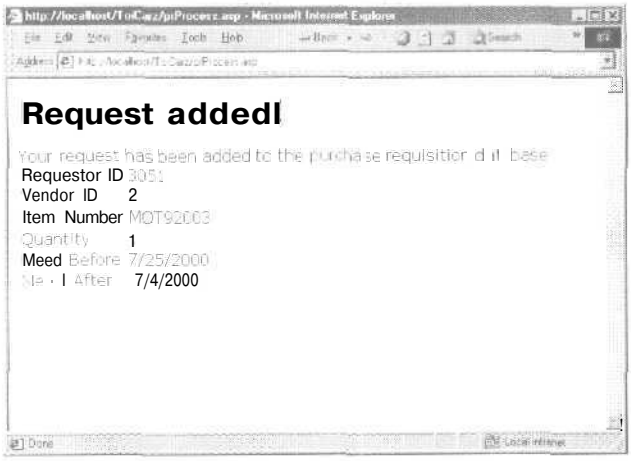


Рис. 11-5. ***Активная страница*** *подтверждает внесение* ***заявки*** *в базу* ***данных.***

Код программы prProcess.asp приведен **в листинге** 11-2.

Листинг 11-2. Программа проверки корректности заявки на покупку.

```
prProcess.asp
<%@ LANGUAGE = "VBScript" %>
<HTML><BODY STYLE="font-family:Verdana">
<%
' Открываем БД Employee
Set oEmpConn = Server.CreateObject("ADODB.Connection")
oEmpConn.Open("dbTC-Employees")
' Отрываем БД Vendor
Set oVendorsConn - Server.CreateObject("ADODB.Connection")
oVendorsConn.Open("dbTC-Vendors")
' Отрываем БД PO
Set oPOConn = Server.CreateObject("ADODB.Connection")
oPOConn.Open("dbTC-PurchaseOrders")
' Отрываем БД Inventory
Set oInventoryConn = Server.CreateObject("ADODB.Connection")
oInventoryConn.Open("dbTC-Inventory")
reqError - ""
' Проверка
Set itemRS = oVendorsConn.Execute( "SELECT price FROM Catalog " _
   & "WHERE itemNum='" _
```

```
   & Request.Form("itemNum") _
   &....)
If itemRS.EOF Then
   reqError = reqError _
      & "<BR>Cannot find item " _
      & Request.Form("itemNum")
End If
Set employeeRS = oEmpConn.Execute("SELECT purchaseLimit " _
   & "FROM Employees " _
   & "WHERE empID=" & Request.Form("requestorID"))
If employeeRS.EOF Then
   reqError = reqError _
      & "<BR>Cannot find employee " _
      & Request.Form("requestorID")
End If
' Проверка полномочий служащего
qty = Request.Form("itemQty")
price = itemRS("price")
totalPrice = qty * price

If CDbl(totalPrice) > CDbl(emoloyeeRS("purchaseLimit")) Then
   reqError = reqError _
      & "<BR>Your purchase authority of $" _
      & employeeRS("purchaseLimit") _
      & " is less than your request total of $" _
      & totalPrice _
      & ". Try again."
End If
' Проверка наличия товара на складе
Set inventoryRS = oInventoryConn.Execute( _
   "SELECT * FROM inventory " _
   & "WHERE vendorID=" S Request.Form("vendorID") _
   & " AND itemNum='" & Request.Form("itemNum") &....)
qtyRequested = Request.Form("itemQty")
If Not inventoryRS.EOF Then
   qtyOnHand = inventoryRS("quantity")
   If CDbl(qtyOnHand) >= CDbl(qtyRequested) Then
      reqError = reqError _
         & "<BR>Inventory reports " _
         & qtyOnHand _
```

```
            & " units of item " _
            & Request.Form("itemNum") _
            & " already in stock. Requisition denied."
        Else
            reqMessage - reqMessage _
            & "<BR>Inventory reports " _
            & qtyOnHand _
            & " units of item " _
            & Request.Form("itemNum") _
            & " already in stock. Order quantity reduced to " _
            & qtyRequested - qtyOnHand _
            & " units."
            qtyRequested = qtyRequested - qtyOnHand
        End If
    End If
    If reqError <  " Then
        Response.Write("<H1>Error</H1>")
        Response.Write(reqError)
    Else
        Response.Write(reqMessage)
        Set rsOrder = Server.CreateObject("ADODB.Recordset")
        rsOrder.CursorType = 2
        rsOrder.LockType - 2
        rsOrder.Open "PORequests", oPOConn
        rsOrder.AddNew
        rsOrder("timeStamp") - Now()
        rsOrder("requestorID") = Request.Form("requestorID")
        rsOrder("vendorID")   = Request.Form("vendorID")
        rsOrder("itemNum")     - Request.Form("itemNum")
        rsOrder("itemQty")     - qtyRequested
        rsOrder("needBefore") - Request.Form("needBefore")
        rsOrder("needAfter") = Request.Form("needAfter")
        rsOrder("itemPrice") = price
        rsOrder.Update
        rsOrder.Close
        Set rsOrder = Nothing
        oPOConn.close
        Set oPOConn = Nothing
        Response.Write("<H1>Request added!</H1>")
        Response.Write("<P>Your request has been added to the " _
```

```
        & "purchase requisition database")
    Response.Write("<TABLE>")
    Response.Write("<TR><TD>Requestor ID</TD><TD>" _
        S Request.Form("requestorID") & "</TD></TR>")
    Response.Write("<TR><TD>Vendor ID</TD><TD>" _
        & Request.Form("vendorID") & "</TD></TR>")
    Response.Write("<TR><TD>Item Number</TD><TD>" _
        & Request.Form("itemNum") & "</TD></TR>")
    Response.Write("<TR><TD>Quantity</TD><TD>" _
        & qtyRequested & "</TD></TR>")
    Response.Write("<TR><TD>Need Before</TD><TD>" _
        & Request.Form("needBefore") & "</TD></TR>")
    Response.Write("<TR><TD>Need After</TD><TD>" _
        & Request.Form("needAfter") & "</TD></TR>")
End If
%>
</BODY>
</HTML>
```

ROPE и SDL

В этом разделе мы рассмотрим программу poGen.exe, написанную на Visual Basic. Эта программа использует новую технологию Microsoft — *представительский сервис удаленных объектов* (Remote Object Proxy Engine, ROPE). В главе 8 мы написали SOAP-клиент сложным способом: мы сцепляли строки, чтобы сформировать конверт SOAP, затем пересылали конверт с содержимым серверу, который выбирал нужную информацию, формировал ответ и отправлял его. Сцепление строк помогло лучше понять структуру пакета SOAP и узнать, как он работает. но это решение соображениям не годится для серьезных программ: во-первых, программисты должны знать синтаксис SOAP для удаленною вызова процедур, а во вторых, они должны понимать правила XML, чтобы создавать документы, поддающиеся синтаксическому анализу.

Не лучше ли использовать стандартную службу, расположенную между вами и создаваемыми SOAP-сообщениями? Она могла бы транслировать запросы к серверу в нужный XML-код и возвращать требуемые данные. Иначе говоря, не лучше ли иметь шлюз для удаленного вызова методов? Такой шлюз и реализует ROPE (рис. 11-6). С точки зрения реализации, это COM-сервер (rope.dll), поставляемый в пакете разработчика SOAP Toolkit для Visual Studio 6.0; он включен в состав компакт-диск книги.

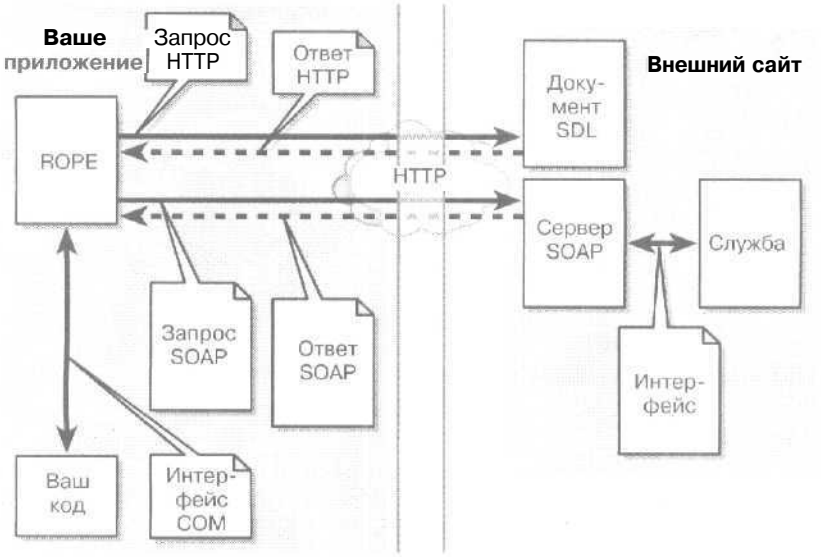


Рис. 11-6. ROPE преобразует запрос в документ SOAP и связывается с удаленными объектами.

Для работы ROPE нужен XML-файл в синтаксисе *языка описания службы* (Service Description Language, SDL). Этот файл описывает все службы, доступные запрашивающему приложению. Вы можете создать документ SDL с помощью мастера из состава SOAP Toolkit. Мастер считывает определение COM-объекта и создает соответствующий файл SDL. Документ SDL можно создать и вручную, скажем, если вы имеете дело не с COM-приложением (этот случай проиллюстрирован в листинге 11-3). Спецификация SDL включена в комплект документации SOAP Toolkit для Visual Studio 6.0. Пример документа SDL, который мы используем, приведен в листинге 11-3.

ПРИМЕЧАНИЕ Когда я писал книгу, спецификация SDL пребывала в состоянии становления. Большинство сторон согласно, что система обнаружения методов на базе XML нужна, но синтаксис мог претерпеть изменения к моменту публикации этой книги. Самую свежую информацию см. на сайте *http://architag.com/support*[1].

[1] Недавно опубликована спецификация разработанного Microsoft языка описания Web-служб (Web-services Description Language, WSDL). *http://msdn.microsoft.com/xml/general/wsdl.asp.* — *Прим. ред.*

Листинг 11-3. Документ SDL определяет сетевые службы, доступные через SOAP и ROPE.

Services.xml

```
<?xml version="1.0" ?>
<serviceDescription
   name="SoapService"
   xmlns="http://localhost/soapdemo/sdlschema.xml"
   xmlns:dt="http://www.w3.org/1999/XMLSchema"
   xmlns:ss ="#ServiceSchema">
   <import   namespace="#ServiceSchema" location ="#ServiceSchema"/>
   <soap xmlns="urn:schemas-xmlsoap-org:soap-sdl-2000-01-25">
      <interface name="BizTalkInterface"/>
      <service>
         <implements name="BizTalkInterface" />
         <!- Передача сообщения BizTalk на обработку ->
         <requestResponse name="BizTalkMessage">
            <request ref="ss:BizTalkMessageRequest"/>
            <response ref="ss:BizTalkMessageResponse"/>
            <parameterorder>BizTalkDoc</parameterorder>
         </requestResponse>
         <!- Проверка БД Inventory ->
         <requestResponse name="checkInventory">
            <request ref="ss:checkInventoryRequest"/>
            <response ref="ss:checkInventoryResponse"/>
            <parameterorder>itemNum quantity</parameterorder>
         </requestResponse>
         <!- Проверка статуса сервера ->
         <requestResponse name="RUOK">
            <request  ref="ss:RUOKRequest"/>
            <response ref="ss:RUOKResponse"/>
         </requestResponse>
         <addresses>
            <location url="http://localhost/toycarparts/BTServer.xar"/
>
         </addresses>
      </service>
   </soap>
   <ss:schema   id="ServiceSchema">
```

```
        <element name="BizTalkMessageRequest">
            <type>
                <element name="BizTalkDoc" type="dt:string"/>
            </type>
        </element>
        <element name="BizTalkMessageResponse">
            <type>
                <element name="DeliveryReceipt" type="dt:string"/>
            </type>
        </element>
        <element name="checkInventoryRequest">
            <type>
                <element name="itemNum" type="dt:string"/>
                <element name="quantity" type="dt:string"/>
            </type>
        </element>
        <element name="checkInventoryResponse">
            <type>
                <element name="itemPrice" type="dt:string"/>
                <element name="deliveryDate" type="dt:string"/>
            </type>
        </element>
        <element name="RUOKRequest">
            <type>
                <element name="Subsystem" type="dt:string"/>
            </type>
        </element>
        <element name="RUOKResponse">
            <type>
                <element name="Status" type="dt:string"/>
            </type>
        </element>
    </ss:schema>
</serviceDescription>
```

Чтобы предоставить приложению COM-интерфейс, объект ROPE считывает документ SDL и выясняет, какие службы доступны. Затем они анонсируются как Web-службы, доступные средствами SOAP. Другими словами, для программиста, работающего с ROPE, все является COM-объектом. Это очень интересное решение. В следующем примере вы увидите, как работает ROPE.

Программа генерации заказа на покупку

Программа генерации заказа poGen считывает данные из таблицы PORequests БД PurchaseOrders, сортирует заявки по поставщикам и создает заказы с привлечением адресной информации, содержащейся в БД Vendors. Она может запускаться раз в день или даже каждую минуту, если заявок много. Пользовательский интерфейс программы poGen показан на рис. 11-7. Основные элементы программы poGen описаны в следующем разделе.

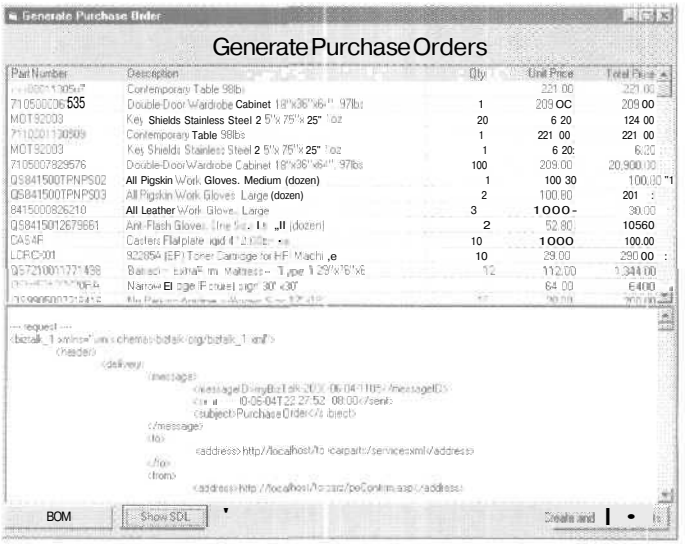


Рис. 11-7. Программа poGen создает заказы на покупку и отправляет их поставщикам в виде документов BizTalk.

Процедура Form.Load

Процедура *Form.Load* (листинг 11-4) извлекает реквизиты товара и отображает их в окне. Это делается исключительно из соображения наглядности — в принципе такая программа не предусматривает взаимодействия с пользователем.

Листинг **11-4. Процедура Form_Load** программы poGen **отображает** реквизиты **заказанного товара.**

```
Form.Load
Private Sub Form_Load()
    Randomize
```

```
    strSQL = "SELECT PORequests.*, catalog.*, vendors.* " _
        & "FROM vendors INNER JOIN (PORequests " _
        & "INNER JOIN [catalog] ON [PORequests].[itemNum]=" _
        & "[catalog].[itemNum]) ON ([PORequests].[vendorID]=" _
        & "[vendors].[vendorID]) AND ([vendors].[vendorID]=" _
        & "[catalog].[vendorID]) ORDER BY [vendors].[vendorID]"
    Set odbPO = New ADODB.Connection
    odbPO.Open ("dbTC-PurchaseOrders")
    Set orsPO = odbPO.Execute(strSQL)
    Do Until orsPO.EOF
        gridPO.AddItem orsPO("itemNum") _
            £ vbTab & orsPO("description") _
            & vbTab & orsPO("itemQty") _
            & vbTab & Format(orsPO("itemPrice"), _
                "###,##0.00") _
            & vbTab & Format(orsPO("itemPrice") _
                * orsPO("itemQty"), "###,##0.00")
        orsPO.MoveNext
    Loop
    orsPO.Close
    Set orsPO = Nothing
End Sub
```

Элемент управления *MSFlexGrid* отображает данные в виде таблицы. Для заполнения полей элемента применяется запрос, извлекающий информацию из таблиц БД PurchaseOrder.

Процедура createPO_Click

Когда пользователь нажимает кнопку Create And Send Pos (Создать и отправить запрос на покупку), процедура *createPO_Click* (листинг 11-5) в цикле по поставщикам выбирает все заявки на покупку для данного поставщика и создает единый документ-заказ.

Листинг 11-5. Процедура createPO_Click формирует заказы и отправляет их поставщикам.

```
createPO_Click
Private Sub createPO_Click()
    Dim txtBizTalk As String
    Dim txtSOAP As String
    Dim txtXML As String
    Dim oXMLDoc As New msxml2.DOMDocument
```

```
Dim oParmsDoc As New msxml2.DOMDocument
Dim orsRequisitions As New ADODB.Recordset
Dim orsPOSent As New ADODB.Recordset
Dim strSQL As String
Dim currentDate As Date
Dim strSendTo As String
Dim strReplyTo As String
Dim currentVendor As Integer
Dim poNum
Dim strParms As String
txtResponse.Text = ""
currentDate = Now()
currentVendor = 0
strSQL = "SELECT PORequests.*, vendors.*, PORequests.vendorID " _
   & "FROM vendors INNER JOIN " _
   & "PORequests ON vendors.vendorID = PORequests.vendorID " _
   & "ORDER BY PORequests.vendorID"
orsRequisitions.CursorType = adOpenKeyset
orsRequisitions.LockType = adLockOptimistic
orsRequisitions.Open strSQL, odbPO,,, adCmdText
orsPOSent.CursorType = adOpenKeyset
orsPOSent.LockType = adLockOptimistic
orsPOSent.Open "POSent", odbPO,,, adCmdTable
Do Until orsRequisitions.EOF
   Screen.MousePointer = vbHourglass
   orsPOSent.AddNew
   orsPOSent("poDate") - currentDate
   orsPOSent("timeSent") = Now()
   orsPOSent.Update
   poNum = orsPOSent("poNum")
   'Генерация номера заказа
   orsPOSent("messageID") = "uuid:" & genUUID()
   orsPOSent.Update
   txtXML = "<purchaseOrder_Chapter11_1.0" _
      & vbCrLf & " xmlns = 'urn:purchaseOrder_Chapter12_1.0.xdr'" _
      & vbCrLf & " custID = '" & CUST_ID & . . . _
      & vbCrLf & " poNumber = '" & poNum &. . . . . _
      & vbCrLf & " poDate = '" & iso85G1(currentDate) & "'>"
   currentVendor - orsRequisitions("vendorID")
```

```
nextItem:
    txtXML = txtXML & vbCrLf & "        <item" _
        & vbCrLf & " needAfter='" _
        & iso8601(orsRequisitions("needAfter")) & ..... _
        & vbCrLf & " needBefore='" _
        & iso8601(orsRequisitions("needBefore")) & ..... _
        & vbCrLf & " reqID='" & orsRequisitions("reqID") & "'" _
        & ">"
    txtXML = txtXML & vbCrLf S "        <number>" _
        & orsRequisitions("itemNum") S "</number>"
    txtXML = txtXML & vbCrLf & "        <price>" _
        & orsRequisitions("itemPrice") & "</price>"
    txtXML = txtXML & vbCrLf & "        <qty>" _
        & orsRequisitions("itemQty") & "</qty>"
    txtXML = txtXML & vbCrLf & "</item>"
    strParms - "<?xml version='1.0'?>" _
        & "<parms>" _
        & "<toAddress type='agr:department'>" _
        & orsRequisitions("vendorToAddress") & "</toAddress>" _
        & "<messageID>" & orsPOSent("messageID") _
        & "</messageID>" _
        & "<sendTo>" & orsRequisitions("vendorToAddress") _
        & "</sendTo>" _
        & "<confirmTo>" & orsRequisitions("vendorConfirmAddress") _
        & "</confirmTo>" _
        & "</parms>"
    orsRequisitions("poNum") = poNum
    orsRequisitions.Update
    orsRequisitions.MoveNext
    If orsRequisitions.EOF Then GoTo around
    If currentVendor - orsRequisitions("vendorID") Then
        GoTo nextItem
    End If
around:
    txtXML = txtXML & "</purchaseOrder_Chapter11_1.0>"
    oXMLDoc.loadXML (txtXHL)
    If oXMLDoc.parseError.errorCode <> 0 Then
        MsgBox ("XML Error: " & oXMLDoc.parseError.reason _
            & " on line " S oXMLDoc.parseError.Line)
    End If
```

```
        oParmsDoc.loadXML (strParms)
        txtBizTalk = envelopeBizTalk(oXMLDoc, oParmsDoc)
        oXMLDoc.loadXML (txtBizTalk)
        If oXMLDoc.parseError.errorCode <> 0 Then
            MsgBox ("XML Error: " & oXMLDoc.parseError.reason _
                & " on line " & oXMLDoc.parseError.Line)
        End If
        sendMsg oXMLDoc
    Loop
out:
    orsPOSent.Close
    Set orsPOSent = Nothing
    orsRequisitions.Close
    Set orsRequisitions = Nothing
    Screen.MousePointer = vbDefault
End Sub
```

Процедура создает элемент *PurchaseOrder_Chapter11_1.0* в виде XML-строки. Именно она будет содержанием документа BizTalk, пересылаемого поставщику.

Функция sendMsg

Функция *sendMsg* взаимодействует со шлюзом ROPE (листинг 11-6).

Листинг 11-6. Функция sendMsg обеспечивает доступ к Web-службе, считывая SDL-файл на удаленном сайте и создавая соответствующий COM-интерфейс.

```
sendMsg
Function sendMsg(oXMLDoc As msxml2.DOMDocument)
    Dim retryCount
    Dim sendTo As String
    ' Поиск вызова метода
    sendTo = oXMLDoc.selectSingleNode("//dlv:to/dlv:address").Text
    Dim oROPE As New ROPE.Proxy
    Dim oPackager As New ROPE.SOAPPackager
    ' Вызов ROPE-представителя
    Dim bRetVal As Boolean
    bRetVal = oROPE.LoadServicesDescription(icURI, sendTo)
    If (bRetVal = False) Then
        MsgBox "LoadServicesDescription = " _
            & bRetVal,, "LoadServicesDescription FAILED"
```

```
        End
    End If
    Dim sBuffer As String
    Dim sTransaction
    sTransaction = vbCrLf & "- - request --" & vbCrLf & oXMLDoc.xml
    On Error GoTo badSOAP
    sBuffer - oROPE.BizTalkMessage(oXMLDoc.xml)
    sTransaction = sTransaction _
        & vbCrLf & "-- response --" & vbCrLf & sBuffer
    sTransaction = sTransaction _
        & vbCrLf & vbCrLf & "------------" _
        & vbCrLf & vbCrLf
    txtResponse.Text = sTransaction & txtResponse.Text
    txtResponse.Refresh
    On Error GoTo 0
    Set oROPE = Nothing
    sendMsg - True
    Exit Function
badSOAP:
    Set oROPE = Nothing
    MsgBox ("SOAP server returned a fault")
    On Error GoTo 0
    sendMsg = False
    Exit Function
End Function
```

Главная задача функции *sendMsg* — создание экземпляра объекта ROPE и установление связей с ним. Рассмотрим соответствующий фрагмент кода:

```
bRetVal = oROPE.LoadServicesDescription(icURI, sendTo)
sBuffer = oROPE.BizTalkMessage(oXMLDoc.xml)
```

В первой строке создается соединение с внешней службой посредством загрузки документа SDL, что обеспечивает позднее связывание с объектом. Позднее связывание подразумевает доступность методов объекта только в период выполнения, но не во время разработки. Это видно по простому признаку — Microsoft IntelliSense не дает подсказок о методах внешнего объекта.

Как только объект **ROPE** извлечет данные из документа SDL, вы сможете вызывать методы объекта, используя привычный интерфейс COM. Приведенные выше две строки кода выполняют всю работу по

установлению связи с любым внешним объектом, поддерживающим SOAP и SDL. Если вы хотите посмотреть методы, описанные в документе SDL, нажмите кнопку Show SDL (Показать SDL).

Обработка заказа на покупку

У компании Toy Car Parts есть «приложение», позволяющее вводить данные о заказах, — страница PlaceOrder.htm (рис. 11-8). Оно представляет собой документ HTML, который вызывает для обработки заказа программу на OmniMark.

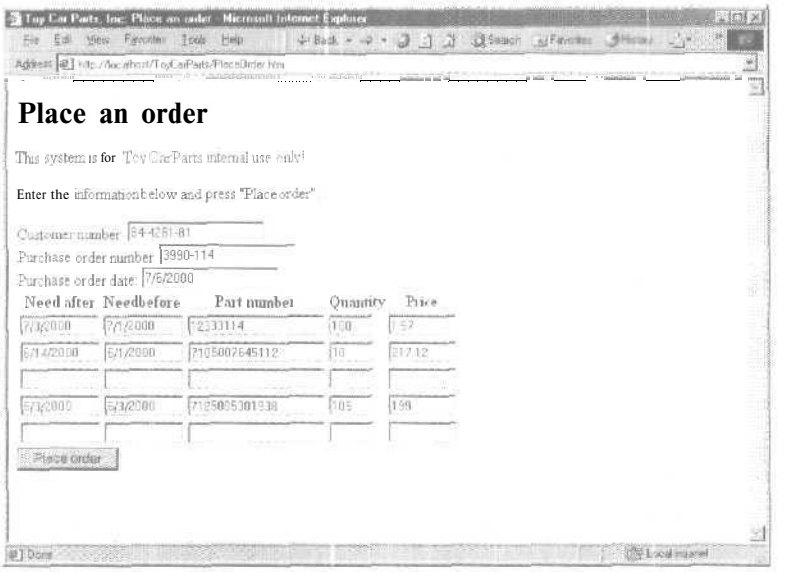


Рис. 11-8. Приложение обработки заказов PlaceOrder.htm.

PlaceOrder.htm — это простой HTML-документ с формой, позволяющей служащему ввести информацию о заказе. Данные этой формы HTML передаются OmniMark-программе для последующей обработки согласно бизнес-правилам компании, включая модификацию соответствующих БД и отправку ответа пользователю (рис. 11-9).

Наша система обработки заказов на базе BizTalk должна решать те же задачи, что и существующее приложение. Мы могли бы написать приложение для обработки документов BizTalk по тем же бизнес-правилам, но это дублировало бы уже выполненную работу. Мы по-прежнему хотим иметь возможность работать с приложением ручного ввода — скажем, на случай поступления заказа на бумаге. Если мы

напишем «интеллектуальное» приложение BizTalk, нам придется реализовывать одни и те же бизнес-правила дважды. Итак, самое разумное — как-то встроить нашу автоматизированную систему в существующий технологический процесс вместо служащего, сохранив возможность ручного ввода информадии.

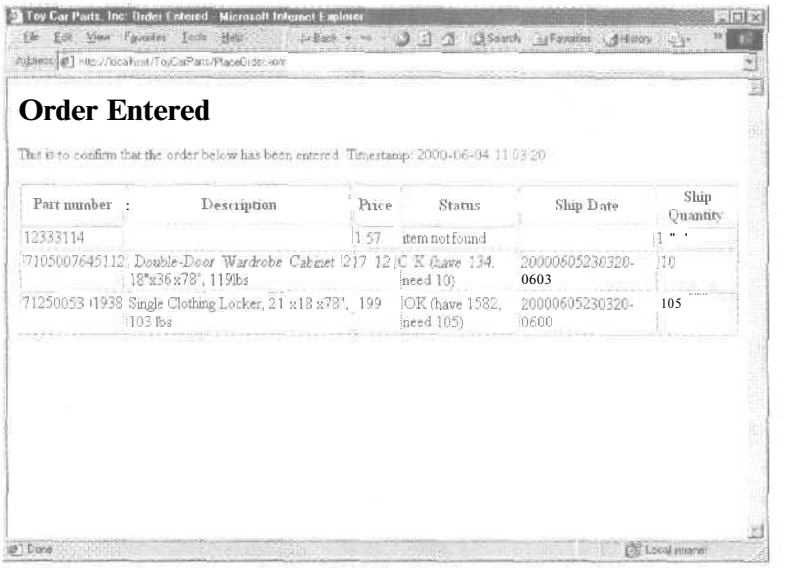


Рис. 11-9. Ответ, созданный программой PlaceOrder.xom.

Мы сделаем это так: наш сервер BizTalk должен будет создавать такие же документы, как и страница PlaceOrder.htm, и передавать их для обработки программе PlaceOrder.xom. На самом деле это не так уж сложно, особенно с OmniMark. Несколько сложнее интерпретировать ответ программы в формате HTML. Мы сделаем это с помощью сервера BizTalk.

Автоматизированная обработка заказа на покупку

Сценарий нашего примера продолжается: процесс в компании Toy Car Parts получил запрос на вызов метода объекта сервера BizTalk. Этот метод, найденный средствами SDL, называется *BizTalk Message*. Его обрабатывает приложение CGI, выступающее в качестве сервера BizTalk. Программа BTServer.xom написана на OmniMark.

ПРИМЕЧАНИЕ На самом деле программа poGen создает не совсем то сообщение SOAP, что нам нужно. В период работы над книгой технология ROPE все еще находилась в стадии разработки. Для применения вызова методов с поздним связыванием SOAP-документ должен быть четко стандартизирован. Хотя подобные вызовы работают для большинства приложений, конверт SOAP, сгенерированный при помощи ROPE, не отвечает требованиям к сообщениям BizTalk. Окончательную версию этого кода см. по адресу *http://architag.com/support*Ох уж эти передовые технологии!

У метода *BizTalkMessage* один параметр — документ BizTalk. Объект ROPE в программе *poGen* создает документ SOAP. После этого создается параметр заголовка HTTP *SOAPAction,* определенный спецификацией SOAP. Этот параметр информирует сервер SOAP о том, какой именно метод будет вызван. OmniMark получает это параметр, используя функцию *UTIL_GetEnv*:

```
set SOAPAction to UTIL_GetEnv ("HTTP_SOAPACTION")
```

Параметр *SOAPAction* имеет вид *;http://localhost/toycarparts/BTServer.xar#BizTalkMessage.* Теперь нам нужно выделить название метода из указателя ресурса. Переменная *SOAPAction* обрабатывается кодом, приведенным в листинге 11-7; этот код выделяет название метода и проверяет, поддерживается ли его вызов.

Листинг 11-7. Определение вызываемого метода.

```
CheckingSOAPAction
open SOAPResponse as buffer
do scan #main-input
   match any{formVars{"CONTENT_LENGTH"}}
      => inputData
   out Debug "%ninputData:%n%x(inputData)%n"
   set SOAPAction to UTIL_GetEnv ("HTTP_SOAPACTION")
   put Debug "%nSOAPAction (http):%g(SOAPAction)%n"
   ;http://localhost/toycarparts/BTServer.xar#BizTalkMessage
   do scan SOAPAction
      match [any except "#"]+ "#" any* => SOAPmethod
         set SOAPAction to SOAPMethod
   else
         set SOAPAction to ""
   done
```

```
put Debug "%nSOAPAction (http):%g(SOAPAction)%n"
do when SOAPAction = ""
    put SOAPResponse "%n<SOAP-ENV:Fault>"
        || "%n <faultcode>101</faultcode>"
        || "%n <faultstring>no method specified"
        || "%n</faultstring>"
        || "%n</SOAP-ENV:Fault>"
else
    do scan SOAPAction
        match "BizTalkMessage"
            using group processPO do
                do xml-parse instance
                        scan inputData
                        suppress
                done
            done
            do when errorMsg - ""
                out SOAPResponse "%n     <BizTalkMessageResponse>"
                        || "%n   <DeliveryReceipt status='OK'>"
                        || "BizTalk Message received at "
                        || format-ymdhms
                                "=W, =xD =n =xY, =h:=m:=s =a.m. =t%n"
                                with-date now-as-ymdhms
                        || " From confirmation server: "
                        || escapeXML (fromPOConf)
                        || "</DeliveryReceipt>"
                        || "%n </BizTalkMessageResponse>"
            else
                out SOAPResponse "%n     <BizTalkMessageResponse>"
                        || "%n   <DeliveryReceipt status='ERROR'>"
                        || "BizTalk Message Error: "
                        || errorMsg || ". Processed at "
                        || format-ymdhms
                                "=W, =xD =n =xY, =h:=m:=s =a.m. =t%n"
                                with-date now-as-ymdhms
                        || "</DeliveryReceipt>"
                        || "%n </BizTalkMessageResponse>"
            done
        match "checkInventory"
                out SOAPResponse "%n     <checkInventoryResponse>"
```

```
                || "%n  <DeliveryReceipt status='OK'>"
                || "This method is not supported (2nd edition...). "
                || "Processed at "
                || format-ymdhms
                        "=W. =xD =n =xY, =h:=m:=s =a.m. =t%n"
                        with-date now-as-ymdhms
                || "</DeliveryReceipt>"
                || "%n </checkInventoryResponse>"
        natch "RUOK"
            put SOAPResponse "<RUOKResponse><status>"
                || "OK at "
                || format-ymdhms
                        "=W. =xD =n =xY, =h:=m:=s =a.m. =t%n"
                        with-date now-as-ymdhms
                || "</status></RUOKResponse>"
    else
        put: SOAPResponse "%n<SOAP-ENV:Fault>"
            || "%n <faultcode>100</faultcode>"
            || "%n <faultstring>method not supported:"
            || "%n %g(SOAPAction)</faultstring>"
            || "%n</SOAP-ENV:Fault>"
    done
  done
done
close SOAPResponse
```

Важную часть кода составляют следующие шесть строк;

```
using group processPQ do
  do xml-parse instance
    scan inputData
    suppress
  done
done
```

Обработка возложена на несколько правил, объединенных в группу OmniMark. В процессе обработки вызывается XML-анализатор, которому и передается документ **XML.**

Группа processPO обрабатывает элементы документа BizTalk (листинг 11-8).

Листинг 11-8. Группа processPO, содержащая правила обработки элементов XML сообщения BizTalk.

```
processPO
;-----------------------
group processPO
;-----------------------
global stream txtSendTo
global stream txtReplyTo
global stream itemNum
global stream itemQty
global stream itemPrice
global stream poDate
global stream poNumber
global stream custID
global stream msgBody
global counter itemCounter
global stream needAfter
global stream needBefore
global stream promiseDate
global counter promiseQty
global stream rowData variable initial-size 0
element purchaseOrder_Chapter11_1.0
   local stream HTMLResponse
   Local counter stepper
   local stream payload
   set poDate to attribute poDate
   set poNumber to attribute poNumber
   set custID to attribute custID
   set itemCounter to 0
   open msgBody as buffer
   put msgBody "CustNum=%g(custID)"
   put msgBody "&PONum=%g(poNumber)"
   put msgBody "&PODate=%g(poDate)"
   output "%c"
   close msgBody
   set HTMLResponse to postMsg (msgBody,
      "http://localhost/ToyCarParts/PlaceOrder.xar")
   clear rowData
   using group parseResponse do
```

```
      submit HTMLResponse
   done

   open payload as buffer
   put payload "<purchaseOrder_Confirm_Chapter11_1.0 "
      || "%n xmlns='urn:purchaseOrder_Confirm_Chapter11_1.0'"
      || "%n CustNum='%g(custID)'"
      || "%n PONum='%g(poNumber)'>"
   repeat
      set itemNum to rowData item stepper
      increment stepper by 4
      set promiseDate to rowData item stepper
      increment stepper by 1
      set promiseQty to rowData item stepper
      do unless promiseDate - ""
         put payload "%n<item reqID='"
         || "d" % (reqIDMapper item (stepper / 6)) || "'>"
         || "%n        <datePromised>" || promiseDate
         || "</datePromised>"
         || "%n        <qtyPromised>" || "d" % promiseQty
         || "</qtyPromised>"
         || "%n </item>"
      done
      increment stepper by 1
      Dxit when stepper > number-of rowData
   again
   put payload "%n</purchaseOrder_Confirm_Chapter11_1.0>"
   close payload
   do
      local stream originalBizTalk
      set originalBizTalk to makeResponse(payload,
         txtReplyTo, txtSendTo)
      set fromPOConf to postMsg (originalBizTalk, txtReplyTo)
   done
element #implied
   suppress
element "dlv:address" when parent is "dlv:to"
   set txtSendTo to "%c"
element "dlv:address" when parent is "dlv:from"
   set txtReplyTo to "%c"
```

```
element item
   output "%c"
   increment itemCounter
   put msgBody "&After%d(itemCounter)=%v(needAfter)"
   put msgBody "&Before%d(itemCounter)=%v(needBefore)"
   out msgBody "&Part%d(itemCounter)=%g(itemNum)"
   put msgBody "&Qty%d(itemCounter)=%g(itemQty)"
   put msgBody "&Price%d(itemCounter)=%g(itemPrice)"
   set now reqIDMapper to attribute reqID
element number when parent is item
   set itemNum to "%c"
element price when parent is item
   set itemPrice to "%c"
element qty when parent is item
   set itemQty to "%c"
```

В листинге 11-8 первый обрабатываемый элемент — *purchaseOrder_Chapter11_1.0*, главная задача которого — «обмануть» приложение обработки заказов ProcessOrder.xom, сделав вид, что его вызывает страница PlaceOrder.htm. При обработке элемента *purchaseOrder_Chapter11_1.0* приложение ProcessOrder.xom создает тело HTTP-документа, который будет передан методом POST (подробнее о методе HTTP POST см. главу 8). Документ передается CGI-программе PlaceOrder.xom функцией *postMsg* (листинг 11-9).

Листинг 11-9. Функция postMsg передает документ CGI-программе методом HTTP POST.

```
postMsg
define stream function postMsg (
   value stream payload,
   value stream url
   ) as
   local HttpRequest Request
   local HttpResponse Response
   local stream request-Headers variable
   local stream response-Headers variable
   HttpRequestSetFromURL Request from (url)
   set Request key "method" to "POST"
   set Request key "entity-body" to payload
   HttpRequestSend Request into Response
   HttpObjectGetHeaders Request into request-Headers
```

```
HttpObjectGetHeaders Response into response-Headers
do when not(HttpObjectIsInError Request)
   return Response key "entity-body"
done
```

Функция *postMsg* использует библиотеку функций HTTP языка OmniMark. Функция *HttpRequestSetFromURL* инициализирует пакет HTTP. После этого мы должны изменить имя параметра на *POST* и разместить наши данные в теле документа. И, наконец, функция *HttpRequestSend* отправляет пакет и помещает ответ в другой объект. Пример тела пакета HTTP, пересылаемого методом *POST*, приведен ниже (код представляет собой непрерывную строку, разбитую на несколько лишь из-за ограниченной ширины книжной страницы).

```
CustNum=84-4281-81&PONum=1182&PODate=2000-06-04&After1=2000-05-
24&Before1=2000-06-22&Part1=LSC91023&Qty1=1&Price1=340&After2=2000-05-
21&Before2=2000-06-11&Part2=LSC91012&Qty2=12&Price2=35
```

Единственное, что нас интересует, — это тело ответного HTTP-документа, который мы получаем после отправки своего. Ответный документ представляет собой HTML-код, созданный программой CGI (листинг 11-10).

Листинг 11-10. Ответный HTML-документ, созданный программой PlaceOrder.xom.

```
Return.htm
<HTML>
 <HEAD>
   <TITLE>Toy Car Parts, Inc: Order Entered</TITLE>
 </HEAD>
 <BODY>
   <H1>Order Entered</H1>
   <P>This is to confirm that the order below has been entered,
    Timestamp: 2000-06-04 11:54:52</P>
   <TABLE BORDER='1'>
    <THEAD>
      <TR>
       <TH>Part number</TH>
       <TH>Description</TH>
       <TH>Price</TH>
       <TH>Status</TH>
       <TH>Ship Date</TH>
```

```
        <TH>Ship Quantity</TH>
      </TR>
    </THEAD>
<TR>
   <TD VALIGN='TOP'>LSC91023</TD>
   <TD VALIGN='TOP'>"Marine Corps" Locker, 2551bs </TD>
   <TD VALIGN='TOP'>340</TD>
   <TD VALIGN='TOP'>Manager approval (price too low)</TD>
   <TD VALIGN='TOP'>20000605235452-0600</TD>
   <TD VALIGN='TOP'>1</TD>
</TR>
<TR>
   <TD VALIGN='TOP'>LSC91012</TD>
   <TD VALIGN='TOP'>Single Clothing Locker, 21"x18"x78", 117lbs</TD>
   <TD VALIGN='TOP'>35</TD>
   <TD VALIGN='TOP'>Manager approval (price too low)</TD>
   <TD VALIGN='TOP'>20000605235452-0600</TD>
   <TD VALIGN='TOP'>12</TD>
</TR>
 </TABLE>
 </BODY>
</HTML>
```

Как только документ получен, начинается выполнение правил группы parseResponse:

```
using group parseResponse do
   submit HTMLResponse
done
```

Группа parseResponse (листинг 11-11) содержит набор правил *find*, которые выполняют поиск по шаблону и заполняют ассоциативный массив (полка с доступом по ключу в терминологии OmniMark) значениям из ячеек таблицы.

Листинг 11-11. Группа parseResponse содержит группу правил *find* для обработки документа HTML, возвращенного программой PlaceOrder.xom.

```
ParseResponse
;--------------
group parseResponse
;--------------
find ("<TD VALIGN='TOP'>" any** "</TD>" white-space*)+ => Cells
```

```
   repeat scan Cells
      match "<TD VALIGN='TOP'>" any** => Content "</TD>"
         set new rowData to Content
      match any
   again

find any
   ; остальное нас не интересует
```

Массив *rowData* содержит данные из всех ячеек таблицы. Теперь можно завершить создание подтверждения заказа, который будет отправлен компании Toy Carz. Он будет выглядеть примерно так:

```
<purchaseOrder_Confirm_Chapter11_1.0
   xmlns='urn:purchaseOrder_Confirm_Chapter11_1.0'
   CustNum='84-4281-81'
   PONum='1189'>
   <item reqID='62'>
      <datePromised>20000606000352-0600</datePromised>
      <qtyPromised>1</qtyPromised>
   </item>
   <item reqID='70'>
      <datePromised>20000606000352-0600</datePromised>
      <qtyPromised>12</qtyPromised>
   </item>
</purchaseOrder_Confirm_Chapter11_1.0>
```

Этот документ создан кодом, показанным в листинге 11-12.

Листинг 11-12. Создание подтверждения заказа.

```
Creating line items

put payload "<purchaseOrder_Confirm_Chapter11_1.0 "
   || "%n xmlns='urn:purchaseOrder_Confirm_Chapter11_1.0'"
   || "%n CustNum='%g(custID)'"
   || "%n PONum='%g(poNumber)'>"
repeat
   set itemNum to rowData item stepper
   increment stepper by 4
   set promiseDate to rowData item stepper
   increment stepper by 1
```

```
    set promiseQty to rowData item stepper
    do unless promiseDate = ""
        put payload "%n    <item reqID='"
        || "d" % (reqIDMapper item (stepper / 6)) || "'>"
        || "%n     <datePromised>" || promiseDate
        || "</datePromised>"
        || "V     <qtyPromised>" || "d" % promiseQty
        || "</qtyPromised>"
        || "%r </item>"
    done
    increment stepper by 1
    exit wren stepper > number-of rowData
again
put payload "%n</purchaseOrder_Confirm_Chapter11_1.0>"
close payload
```

Подтверждение **помещено в документ BizTalk функцией** *make-Response* (листинг 11-13).

Листинг 11-13. Функция *makeResponse* создает конверт BizTalk и помещает в него документ.

```
makeResponse
define stream function makeResponse(
value stream strPayload,
value stream strSendTo,
value stream strReplyTo
) as
local stream response
set response to "<?xml version='1.0'?>"
|| "%n<SOAP-ENV:Envelope"
|| "%n xmlns:SOAP-ENV='http://schemas.xmlsoap.org/soap/envelope/'"
; I "%rxmlns:SOAP-ENC='http://schemas. xmlsoap.org/soap/encoding/' "
| | "%n xmlns:xsi='http://www.w3.org/1999/XMLSchema-instance'>"
| | "%n <SOAP-ENV:Header>"
|' | "%n<dlv: delivery SOAP-ENV:mustUnderstand='1'"
|| "%n xmlns:dlv='http://schemas.biztalk.org/btf-2-0/delivery'"
|| "%n xmlns:agr='http://www.trading-agreements.org/types/'>"
|| "%n <dlv:to>"
| | "%n    <dlv:address xsi:type='agr:department'>"
|| escapeXML(strSendTo) || "</dlv:address>"
```

```
|| "%n </dlv:to>"
|| "%n <dlv:from>"
|| "%n   <dlv:address xsi:type='agr:organization'>"
|| escapeXML(strReplyTo) || "</dlv:address>"
|| "%n </dlv:from>"
|| "%n </dlv:delivery>"
|| "%n <prop: properties SOAP-ENV:mustUnderstand='1'"
|| "%n xmlns:prop='http://schemas.biztalk.org/btf-2-0/properties'>"
|| "%n <prop:identity>uuid:" || genUUID()
|| "</prop:identity>"
|| "%n <prop:sentAt>"
|| iso8601(now-as-ymdhms)    "</prop:sentAt>"
|| "%n <prop:expiresAt>"
|| iso8601(add-to-ymdhms now-as-ymdhms hours 2)
|| "</prop:expiresAt>"
|| "%n <prop:topic>" || escapeXML(strReplyTo)
|| "</prop:topic>"
|| "%n </prop:properties>"
|| "%n </SOAP-ENV:Header>"
|| "%n <SOAP-ENV:Body>"
|| "%n" || strPayload
|| "%n </SOAP-ENV:Body>"
|| "%n</SOAP-ENV:Envelope>"
return response
```

В следующем разделе мы познакомимся с программой обработки подтверждения заказа. Она представляет собой ASP-страницу, обрабатывающую HTTP-документ с сообщением BizTalk. Эта страница вызывается точно так же, как и программа обработки заказа PlaceOrder.xom.

Обработка подтверждения заказа

Вернемся в компанию Toy Carz, где ASP-приложение poConfirm.asp ожидает подтверждения заказа. Эта программа играет роль сервера BizTalk, читая документ BizTalk и обрабатывая его в соответствии с типом элемента и его содержимым (листинг 11-14).

Листинг 11-14. Программа poConfirm.asp обрабатывает подтверждение **заказа.**

```
poConfirm.asp
<%@language=vbscript%>
<%
```

```
strXML = Replace (strXML, "%20", " ")
strXML = Replace (strXML, "%2F", "/")
strXML = Replace (strXML, "%3D", "=")
strXML = Replace (strXML, "%0A", vbCrLf)
Set oXMLDoc = Server.CreateObject("Microsoft.XMLDOM")
oXMLDoc.loadXML(strXML)
If oXMLDoc.parseError.errorcode <> 0 Then
   Response.Write("Error, Reason: " & oXMLDoc.parseError.reason _
      & ", Code; " & oXMLDoc.parseError.errorcode _
      & ", Line: " & oXMLDoc.parseError.line)
End If
poNum = oXMLDoc.selectSingleNode_
("//purchaseOrder_Confirm_Chapter11_1.0").attributes.getNamedItem_
("PONum").nodeValue
Set collItem = oXMLDoc.selectNodes("//item")
For i = 0 To collItem.length - 1
   reqID = collltem.item(i).attributes.getNamedItem("reqID").nodeValue
datePromised = collItem.item(i).selectSingleNode("datePromised").text
   qtyPromised = collItem.item(i).selectSingleNode("qtyPromised").text
   Response.Write("reqID: " & reqID _
      & ", datePrornised: " & datePromised _
      & ", QtyPromised: " & qtyPromised)
   closeReq reqID, datePromised, qtyPromised
Next
closePO(poNum)
%>
```

Программа poConfirm.asp решает три задачи. Во-первых, она должна «закрыть» каждую открытую заявку на покупку, занося в таблицу PORequests дату и объем поставки, гарантированные компанией Toy Car Parts. Это делает функция *closeReq*. Во-вторых, она извещает служащих, сделавших заявку, используя библиотеку CDONTS (Collaboration Data Objects), которая, в частности, предоставляет программный доступ к электронной почте средствами Microsoft Exchange Server. В-третьих, poConfirm.asp должна «закрыть» заказ в таблице POSent. Это делает функция *closePO*.

Затем программа отправляет ответ серверу BizTalk, написанному на OmniMark, который в свою очередь отправляет сообщение программе генерации заказа на покупку, инициировавшей весь процесс.

Наша очень упрощенная система демонстрирует минимальные требования к серверу BizTalk, использующему две платформы и два языка программирования. Разумеется, реальные системы гораздо сложнее, но открытые стандарты вроде BizTalk, SOAP и XML позволяют создавать и развертывать их.

По ходу дела у нас возникла очень интересная проблема — интеграция с приложениями, долгие годы работающими в компании. Система PlaceOrder удачно вошла в новую среду — зачем же нам было менять ее? Мы вдохнули новую жизнь в традиционное приложение благодаря современным инструментам и открытым стандартам. Вот вам и главный урок: вы вовсе не обязаны отбрасывать работающие системы только потому, что они не используют новейшие технологии.

На горизонте уже видны новые инструменты, которые позволят разработчикам создавать системы, способствующие установлению связей между вами и вашими партнерами. SOAP Toolkit for Visual Studio 6.0 — один из таких инструментов, позволяющий развертывать системы, работающие не с сетью, а в сети. Другие поставщики ПО объявили о поддержке SOAP и разработке средств, аналогичных инструментам Microsoft.

Разрабатываются и другие средства для работы с серверами BizTalk. В главе 12 я расскажу о предварительной версии Microsoft BizTalk Server 2000, которая включает инструментарий для создания схем и их преобразования друг в друга. Коммерческая версия Microsoft BizTalk Server 2000 будет включать и другие инструменты, позволяющие разработчику интегрировать бизнес-транзакции и существующие системы.

Глава 12

Microsoft BizTalk Server 2000

В главе 10 были перечислены требования к приобретаемому или разрабатываемому серверу BizTalk. Фактически единственное обязательное требование к серверу — понимать тэги BizTalk и обрабатывать в соответствии с ними документы. Однако возможности Microsoft BizTalk Server 2000 по обработке бизнес-документов на базе XML далеко превосходят эти требования. Этот сервер включает все службы, необходимые для интеграции документооборота и существующих приложений. Кроме того, сюда включены новые компоненты для управления данными, обмена информацией, управления сервером и его обслуживания. Microsoft — один из авторов инициативы BizTalk, поэтому ее продукт реализует большинство средств для организации взаимодействия с партнерами.

BizTalk Server 2000 представляет собой сервер для обмена деловой информацией и интеграции приложений. BizTalk Server 2000 основан на XML, что позволяет взаимодействовать с любыми партнерами по бизнесу и любыми бизнес-приложениями независимо от используемых платформ, ОС и технологий. Сервер реализует шлюз для обмена документами через Интернет и включает ряд служб, обеспечивающих целостность, защиту и доставку данных.

BizTalk Server 2000 способен принимать входящие документы, автоматически выяснять их формат, выделять ключевые параметры, выполнять обработку согласно заданным правилам и доставлять документы получателю. В его состав также включены средства преобразования данных, обеспечения их целостности, защиты и гарантированной доставки.

BizTalk Server 2000 располагает средствами описания структуры бизнес-документов (схем) и преобразования документов из одной структуры в другую с помощью XSLT. В терминологии BizTalk Server 2000 сопоставление элементов схем для автоматического преобразования структуры данных называется *отождествлением* (mapping).

Для пересылки любых данных применяется XML, но система позволяет работать и с другими форматами, включая *формат электронного документооборота* (Electronic Data Interchange, EDI). Средства преобразования данных BizTalk Server 2000 вполне достаточны для организации взаимодействия с другими приложениями.

В этой главе мы на двух простых примерах познакомимся с важнейшими инструментами для создания и обработки XML-документов, которые предлагает BizTalk Server 2000: Microsoft BizTalk Editor и Microsoft BizTalk Mapper.

BizTalk Editor

В главе 4 я упоминал инструментарий для создания схем XML Authority. Зачем нам еще один инструмент? Все дело в том, что XML Authority — универсальный инструмент, в то время как редактор BizTalk Editor специально предназначен для создания бизнес-документов. Кроме того, создаваемые им схемы содержат дополнительную информацию (она хранится в элементах из пространств имен Microsoft и BizTalk), которая существенно облегчает преобразование документов из одной схемы в другую. Редактор BizTalk Editor позволяет создавать схемы как на основе отраслевых стандартов, так и по стандартам, принятым в организации.

Пример: создание схемы заказа

Попробуем с помощью BizTalk Editor создать схему для BizTalk Server 2000. Пускай это снова будет заказ товара. Создадим стандартную схему заказа, которая позволит обмениваться заказами с партнерами. Мы используем инструменты из состава ознакомительной версии BizTalk Server 2000, включенной в состав компакт-диска книги.

1. Запустите редактор BizTalk Editor командами Start, Programs, Microsoft BizTalk Server 2000, BizTalk Editor. После этого экран будет выглядеть так, как показано на рис. 12-1.

   На рис. 12-1 приведен простой пример, достаточный для демонстрации взаимодействия между инструментальными средствами BizTalk Server 2000. Реальные документы гораздо сложнее, однако концепции остаются теми же.

2. Выбрав в меню File команду New, создайте новый документ. В появившемся диалоговом окне New Document Specification установите переключатель Blank Specification.

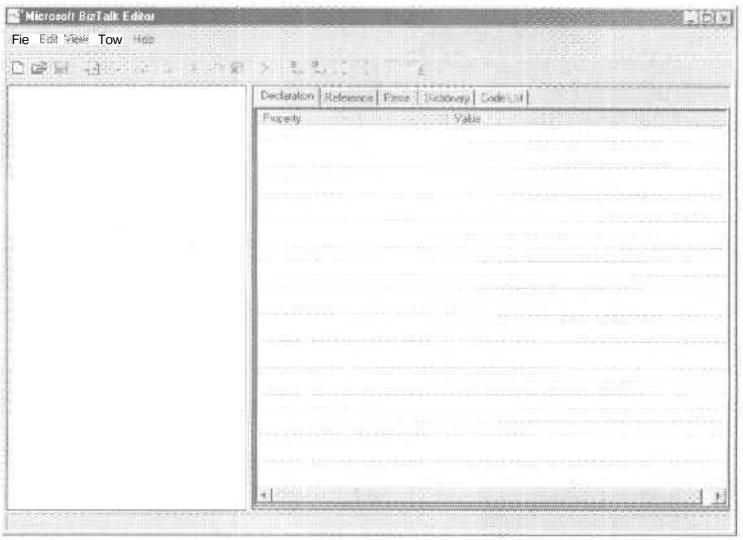


Рис 12-1. ***Редактор BizTalk Editor*** *позволяет* ***создавать*** *описания структуры бизнес-документов.*

3. В левой панели редактора появится значок с меткой Blank Specification, соответствующий корневому элементу документа. Щелкнув его правой кнопкой, выберите в меню команду Rename и переименуйте корневой элемент в *PurchaseOrder.*

   BizTalk Editor не использует терминологию XML. Вместо терминов «элемент» и «атрибут» применяются *запись* (record) и *поле* (field), которые ближе разработчикам баз данных. В BizTalk Editor запись может содержать данные и поля. Тип элемента данных (элемент или атрибут) выбирается в области объявлений. По умолчанию записи соответствуют элементам, а поля — атрибутам.
4. Щелкните правой кнопкой переименованную запись и выберите в контекстном меню команду New Field, чтобы создать новое поле. Назовите его *PODate.*
5. Повторим п. 4 для еще одного поля — *CustomerNumber.*
6. Щелкните правой кнопкой запись *PurchaseOrder* и выберите в контекстном меню команду New Record. Сейчас вы будете создавать

контейнер для остальных полей. Назовем новую запись *Item*. Дополнительные определения задаются в правой панели. Все поля должны иметь тип *Attribute,* а записи — *Element.* Щелкнув ячейку Content (ее содержимое пока пусто, о чем свидетельствует слово *Empty),* выберите в списке значение *Text Only.* Эта запись будет служить для хранения объема заказа.

7. Щелкнув правой кнопкой запись Item и выбрав в меню команду New Field, создайте новое поле *Qty.* В панели Declaration щелкните стрелку списка *Data Type* и выберите в списке *Integer (int).*
8. Повторите п, 7 и создайте новое поле *Price* с типом данных *Fixed Point (14.4).*
9. В меню File выберите команду Save. Присвойте файлу имя *PurchaseOrder.xml* и сохраните его в подходящей папке. В результате экран будет выглядеть так (рис. 12-2):

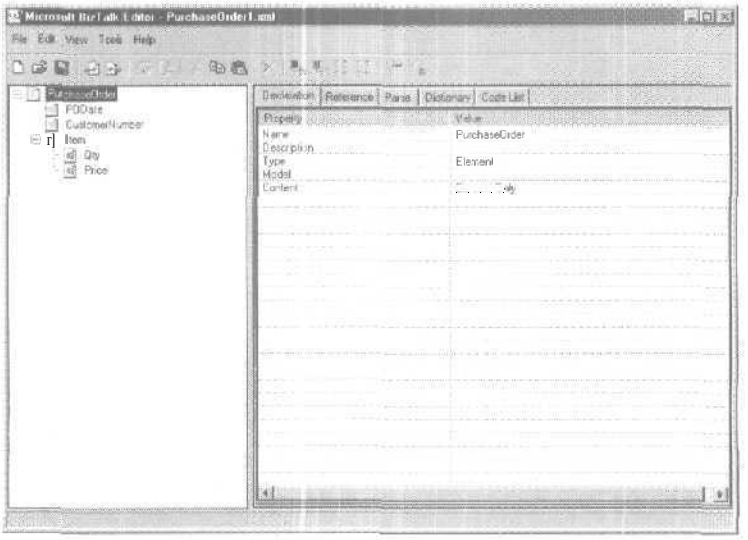


Рис 12-2. Структура заказа, созданная с помощью BizTalk Editor.

После записи документа будет создана схема (листинг 12-1). Поскольку сервер BizTalk использует дополнительную информацию, его схемы не похожи на те, что описаны в главе 4.

Наш документ PurchaseOrderl .xml следует синтаксису XDR (XML Data Reduced). Однако после принятия Консорциумом W3C окончательной редакции синтаксиса XSD компания Microsoft будет поддерживать и этот стандарт. Преобразовать документ из одного синтакси-

са в другой можно командой Save As меню File редактора BizTalk Editor или использовать XSLT.

Обратите внимание на пространство имен *b:*, определенное в открывающем тэге <Schema> (листинг 12-1). Это пространство имен содержит набор тэгов, используемых компонентами сервера BizTalk Server 2000.

Листинг 12-1. Схема, созданная редактором BizTalk Editor.

```
PurchaseOrder1.xml
<?xml version="1.0"?>
<Schema name="PurchaseOrder"
   b:root_reference="PurchaseOrder"
   b:standard="XML"
   xmlns="urn:schemas-microsoft-com:xml-data"
   xmlns:d="urn:schemas-microsoft-com:datatypes"
   xmlns:b="urn:schemas-microsoft-com:BizTalkServer">
   <b:SelectionFields/>
   <ElementType name="PurchaseOrder" content="eltOnly">
      <b:RecordInfo/>
      <AttributeType name="PODate">
         <b:FieldInfo/>
      </AttributeType>
      <AttributeType name="CustomerNumber">
         <b:FieldInfo/>
      </AttributeType>
      <attribute type="PODate"/>
      <attribute type="CustomerNumber"/>
      <element type="Item"/>
   </ElementType>
   <ElementType name="Item" content="textOnly" model="closed">
      <b:RecordInfo/>
      <AttributeType name="Qty" d:type="int">
         <b:FieldInfo/>
      </AttributeType>
      <AttributeType name="Price" d:type="fixed.14.4">
         <b:FieldInfo/>
      </AttributeType>
      <attribute type="Qty"/>
      <attribute type="Price"/>
```

```
    </ElementType>
</Schema>
```

BizTalk Mapper

В главе 6 была создана программа XSLT, преобразующая один документ XML в другой. Возможность преобразования документов имеет важное значение, когда партнеры используют разные схемы.

Думается, предпочтительнее работать именно с XSLT, но далеко не во всех кампаниях есть эксперты в этой области. BizTalk Mapper позволяет, сопоставляя записи и поля двух схем, отождествить их. BizTalk Server 2000 на основе такого отождествления может преобразовывать данные из одного формата в другой.

Пример: отождествление двух схем заказа

Чтобы понять, как работает BizTalk Mapper, воспользуемся только что созданной схемой. Сейчас мы отождествим ее с другой схемой заказа.

1. Запустите BizTalk Mapper командой Start, Programs, Microsoft BizTalk Server 2000, BizTalk Mapper.
2. Выберите две схемы. Во-первых, в меню File выберите команду New. В диалоговом окне Select Source Specification Type нажмите папку Local Files. В появившемся диалоговом окне Select Source Specification укажите местонахождение схемы PurchaseOrder1.xml из предыдущего примера.

   Диалоговое окно Select Destination Specification Type останется на экране. Повторите те же действия для второй схемы, в качестве которой мы выберем схему заказа с компакт-диска книги. Файл называется \Samples\Ch12\PurchaseOrder2.xml. После открытия второй схемы экран будет выглядеть, как на рис. 12-3,

   Отождествление схем выполняется перетаскиванием полей слева на поля справа. Помимо этого, в программу заложены некоторые дополнительные возможности.
3. В меню Tools выберите команду Options. В появившемся диалоговом окне BizTalk Mapper Options установите флажок Allow Record Links — это позволяет сопоставлять не только поля, но и содержимое записей.

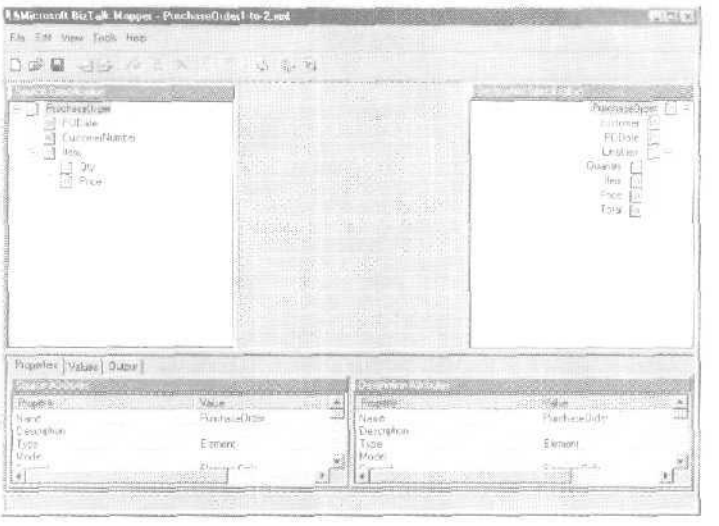


Рис. 12-3. Отождествление двух схем заказа с помощью BizTalk Mapper.

4. Перетащите поле ***Customer Number*** **панели Source** Specification на поле *customer* **панели** Destination Specification. Теперь **поля** будут **связаны линией (рис.** 12-4).

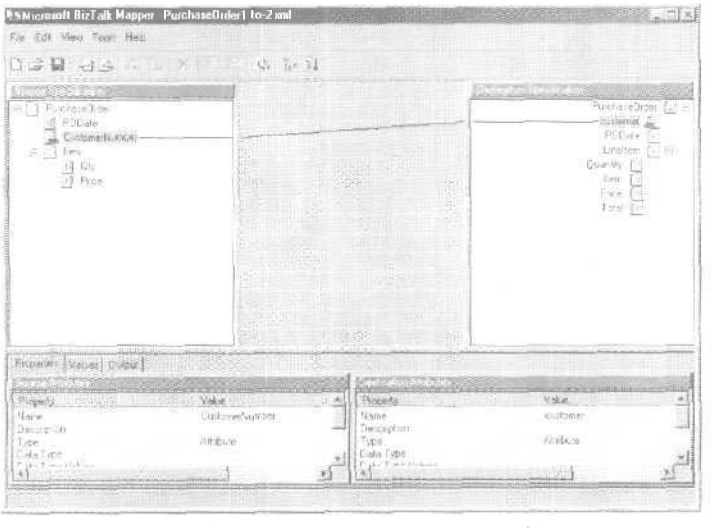


Рис. 12-4. Отождествление выполняется связыванием полей из левой и правой панелей.

5. Повторите п. 4 еше трижды для nap *Item* и *Item*, *Qty* и *Quantity, Price* и *Price*. Все просто.
6. Заметьте: на панели Destination Specification есть поле *Total,* отсутствующее на панели Source Specification (это общая сумма заказа). В этом случае нам придется заняться вычислениями. На помощь приходит *набор функций* (functoid) BizTalk Mapper. В меню View выберите команду Functoid Palette. На экране появится диалоговое окно Functoid Palette (рис. 12-5).

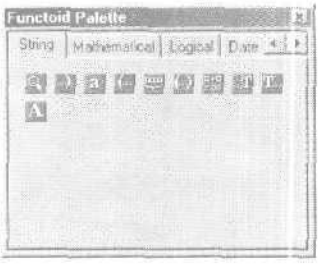


Рис. 12-5. Функции позволяют выполнять вычисления, необходимые для преобразования полей исходной схемы в поля результирующей схемы.

7. Поле *Total* схемы назначения вычисляется умножением полей *Qty* и *Price* панели Source Specification, поэтому нам придется умножать поля. Откройте вкладку Mathematical.
8. Выберите значок с изображением символа умножения и перетащите его на сетку между панелями Source Specification и Destination Specification (рис. 12-6).
9. Соедините элемент *Qty* исходной схемы со значком умножения.
10. Сделайте то же самое с элементом *Price.*
11. Соедините значок умножения с полем *Total* правой панели. Здесь будет храниться результат умножения значений полей.
12. Осталось справиться с датой. Поле *date* исходной схемы имеет формат месяц/день/год, а в схеме в правой панели дата должна быть в формате ISO 8601 (ГГГГ-ММ-ДД). Специальной функции преобразования форматов нет, поэтому придется создать сценарий. В диалоговом окне Functoid Palette откройте вкладку Advanced. Здесь только один значок — Custom Visual Basic Script. Перетащите его в область между панелями.
13. Соедините поле *PODate* исходной схемы со значком сценария, а его — с элементом элемент *PODate* схемы в правой панели.

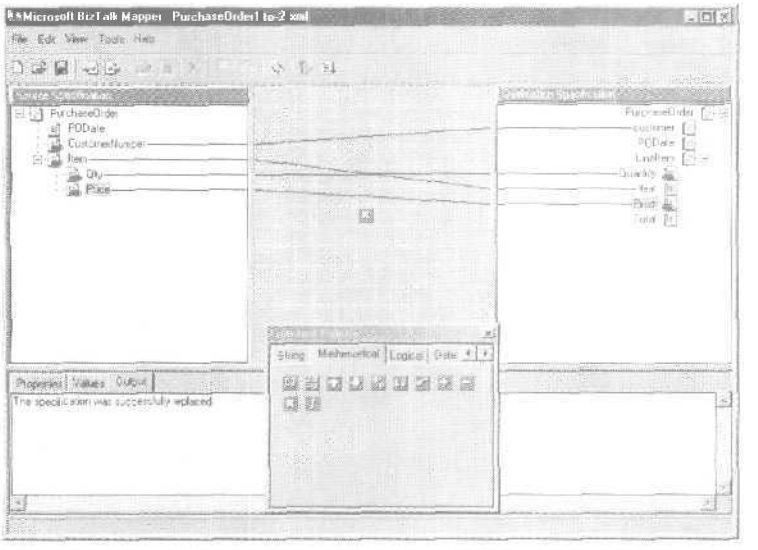


Рис. 12-6. *Функция умножения перемножает значения полей схемы слева и помещает результат в поле схемы справа.*

14. Дважды щелкнув значок сценария, откройте вкладку Script. На экране появится текст сценария (рис. 12-7).

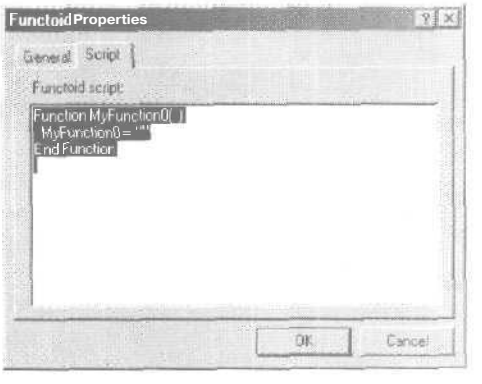


Рис. 12-7. *Значок Custom VisualBasic Script позволяет создавать сценарии преобразования документов.*

15. Вместо текста по умолчанию введите следующий сценарий:

```
Function mdy2iso8601(dateIn)
   iFirst = instr(dateIn, "/")
```

```
    iSecond = instr(iFirst + 1, dateIn, "/")
    iMonth = left(dateIn, iFirst - 1)
    iDay = mid(dateIn, iFirst + 1, iSecond - iFirst - 1)
    iYear = mid(dateIn, iSecond + 1, 4)
    mdy2iso8601 = iYear & "-" & right("0" & iMonth, 2) & _
                "-" & right("0" & iDay, 2)
End Function
```

Функция *mdy2iso8601* преобразует дату из одного формата в другой. Закончив ввод, в диалоговом окне Functoid Properties щелкните кнопку OK. Окно BizTalk Mapper теперь должно выглядеть так (рис. 12-8):

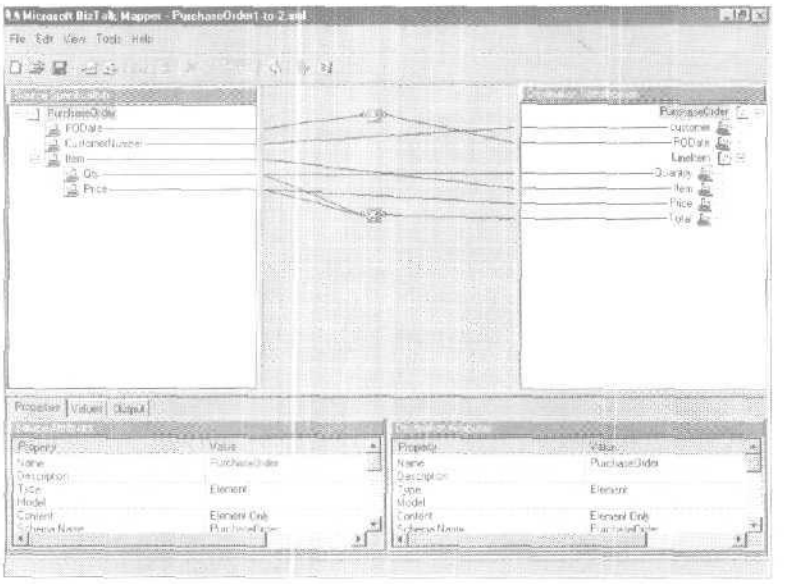


Рис. 12-8. Окно BizTalk Mapper с двумя значками преобразования данных.

16. Проверим работу сценария. Закройте диалоговое окно Functoid Properties. Откройте вкладку Values в нижней части окна BizTalk Mapper. Выберите поле *PODate* на панели Source Specification и введите любую дату в формате ММ/ДД/ГГГГ, скажем, 04/27/2000. Работу этой функции мы проверим чуть позже, а сначала введем тестовые данные для остальных полей.

17. Проверим и умножение полей. В панели Source Specification выберем поле *Qty* и введем на вкладке Values целое значение поля — например, 95.
18. В панели Source Specification выберите поле *Price* и введите цену в виде десятичной дроби — например, 12.95.
19. Все готово к тесту. В меню Tools выберите команду Test Map. Обратите внимание на появившуюся вкладку Output с результатами. Результат будет выглядеть так;

```
<PurchaseOrder customer="CustomerNumber_1"
   PODate="2000-04-10">
<LineItem>
<Quantity>95</Quantity>
<Item>Item_1</Item>
<Price>12.95</Price>
<Total>1230.25</Total>
</LineItem>
</PurchaseOrder>
```

Как и следовало ожидать, формат поля *PODate* изменился, а значение элемента <Total> равно произведению объема заказа на иену.

20. Теперь создадим шаблон XSLT, который выполняет преобразование. В меню Tools выберите команду Compile Map. В результате получится шаблон XSLT (листинг 12-2).

BizTalk Server 2000 будет использовать шаблон преобразования из листинга 12-2 всякий раз, когда от партнера поступает документ XML, отвечающей схеме PurchaseOrder1.xml.

21. Сохраните созданное преобразование командой Save Compiled Map As меню File в файле *PO1-2.xsl*. Именно этот шаблон BizTalk-сервер будет применять для преобразования. Просмотреть файл можно через обозреватель Microsoft Internet Explorer.

Листинг 12-2. Шаблон XSLT, созданный BizTalk Mapper.

P01-2.XS1

```
<xsl:stylesheet
   xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
   xmlns:msxsl='urn:schemas-microsoft-com:xslt'
   xmlns:var='urn:var' xmlns:user='urn:user' version='1.0'>
   <xsl:output method='xml' indent='yes' omit-xml-declaration='yes'/>
   <xsl:template match='/'>
```

```
      <xsl:apply-templates select='PurchaseOrder'Л
   </xsl:template>
   <xsl:template match='PurchaseOrder'>
      <PurchaseOrder>
         <xsl:attribute name='customer'>
            <xsl:value-of select='@CustomerNumber'/>
         </xsl:attribute>
         <xsl:variable name='var:v1'
            select='user:mcy2iso8601(string(@PODate))'/>
         <xsl:attribute name='PODate'>
            <xsl:value-of select='$var:v1'/>
         </xsl:attribute>
      <xsl:for-each select='Item'>
         <LineItem>
            <Quantity>
               <xsl:value-of select='@Qty'/>
            </Quantity>
            <Item>
               <xsl:value-of select='./text()'/>
            </Item>
            <Price>
               <xsl:value-of select='@Price'/>
            </Price>
            <xsl:variable name='var:v2'
select='user:fctmathmultiply2(string(@Qty), string(@Price))'/>
            <Total>
               <xsl:value-of select='$var:v2'/>
            </Total>
         </LineItem>
      </xsl:for-each>
      </PurchaseOrder>
   </xsl:template>
   <msxsl:script language='VBScript' implements-prefix='user'>
      <![CDATA[
      Function FctMathMultiply2( p_strParm0, p_strParm1 )
       If ( IsNumeric( p_strParm0 ) And _
          IsNumeric( p_strParm1 ) ) Then
            FctMathMultiply2 = CStr ( CDbl( p_strparm0 ) _
            * CDbl( p_strParm1 > )
       Else
```

```
            FctMathMultiply2 = ""
        End If
    End Function
    Function mdy2iso8601(dateIn)
        iFirst = instr(dateIn, "/")
        iSecond = instr(iFirst + 1, dateIn, "/")
        iMonth = left(dateIn, iFirst - 1)
        iDay = mid(dateIn, iFirst + 1, iSecond - iFirst - 1)
        iYear = mid(dateIn, iSecond + 1, 4)
        mdy2iso8601 = iYear & "-" & _
        right("0" & iMonth, 2) & _
        "-" & right("0" & iDay, 2)
    End Function
    ]]>
  </msxsl:script>
</xsl:stylesheet>
```

Помимо XML, BizTalk Mapper поддерживает и другие форматы, включая распространенные стандарты EDI ANSI X12 и EDIFACT. Результирующие документы могут иметь текстовый формат и XML (на базе DTD и схем XDR). И все же BizTalk Mapper в основном ориентирован на работу с документами XML в рамках спецификации BkzTalk Framework 2.0.

Другие средства BizTalk Server 2000

В состав BizTalk Server 2000 включены дополнительные возможности организации взаимодействия с партнерами.

- **BizTalk Management Desk** — используя этот графический интерфейс, можно локально и удаленно управлять обменом данными между партнерами и приложениями.
- **BizTalk Server Administration Console** — модуль консоли MMC для управления сервером или группой серверов BizTalk. Модуль снабжен подробной документацией.
- **BizTalk Document Tracking** — пользовательский интерфейс для отслеживания прохождения документов через сервер. Это средство предназначено для аналитиков и системных администраторов. Данные о прохождении документов позволяют аналитикам осуществлять контроль и вырабатывать стратегию. Извлекая информацию из обрабатываемых документов, аналитик может получить подробные сведения о всех операциях. Администратору интерфейс позволяет выбирать объекты для мониторинга и режимы прото-

колирования работы системы, настраивать параметры и устранять неисправности.

- **BizTalk Workflow Designer** — позволяет создавать процессы обработки документов с помощью интерфейса Microsoft Visio. Этот интерфейс реализует графическое представление бизнес-процессов, позволяя аналитику проектировать их с помощью технологии drag-and-drop, после чего BizTalk Server 2000 автоматически создает процесс обработки документов. Бизнес-процесс можно связать с описанием партнера, после чего любой документ, поступающий от этого партнера, будет автоматически обрабатываться созданным процессом.

Средства обработки данных BizTalk Server 2000

Сервер BizTalk Server 2000 способен не только передавать документы XML через Интернет. Его цель — служить центром управления взаимодействием с партнерами. Для описания способа взаимодействия с партнерами сервер предлагает несколько удобных инструментов.

Чтобы продемонстрировать эти возможности сервера BizTalk, рассмотрим пример, где:

- компания Toi Carz Corp., производитель игрушечных автомобилей, использует сервер Microsoft Commerce Server 2000 для связи с Интернетом и ПО компании Great Plains для бухгалтерского учета; для взаимодействия с партнерами установлен сервер BizTalk Server 2000;
- компания Toy Car Parts, Inc., импортер и поставщик комплектующих для игрушечных автомобилей, один из поставщиков Toi Carz, для управления ресурсами использует систему SAP R/3, а для взаимодействия с партнерами — сервер BizTalk Server 2000;
- компания Mehtap Enterprises, производитель моторчиков для игрушечных автомобилей, один из поставщиков Toy Car Parts, использует устаревшее ПО EDI, не модернизировавшееся с 1980-х годов; для взаимодействия с ней надо уметь работать с документами в формате EDI.

Размещение заказа

Служащий Toi Carz размещает заказ на 10 моторчиков компании Mehtap Enterprises. Компания Toy Car Parts принимает этот заказ. Процесс прохождения заказа показан на рис. 12-9 и описан в табл. 12-1.

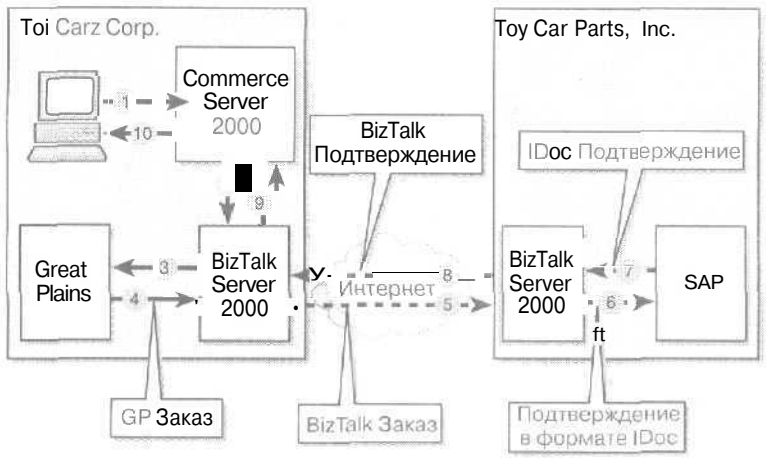


Рис. 12-9. *Размещение заказа поставщику.*

Табл. 12-1. Описание схемы на рис. 12-9, демонстрирующей **прохождение** заказа между компаниям.

Шаг	Описание
1	Служащий Toi Carz размещает заказ. Заказ обрабатывается сервером Commerce Server 2000.
2	Commerce Server передает запрос серверу BizTalk Server 2000.
3	Сервер BizTalk Server 2000 передает сообщение бухгалтерской системе Great Plains...
4	...которая создает заказ в формате Great Plains и передает его серверу BizTalk Server 2000.
5	Сервер BizTalk Server 2000 преобразует заказ в документ BizTalk и пересылает его серверу BizTalk компании Toy Car Parts через Интернет по протоколу HTTP.
6	В компании Toy Car Parts установлена система SAP R/3, работающая с документами в формате IDoc. Сервер BizTalk Server 2000 преобразует документ BizTalk в формат IDoc и пересылает его системе SAP,
7	Система SAP подтверждает серверу BizTalk получение заказа IDoc.

(продолжение)

Шаг	Описание
8	Сервер BizTalk компании Toy Car Parts преобразует подтверждение в документ BizTalk и пересылает его обратно серверу BizTalk компании Toi Carz.
9	Сервер BizTalk компании Toi Carz посылает подтверждение серверу Commerce Server 2000 по внутренней сети.
10	Сервер Commerce Server 2000 сообщает служащему, что поставщик получил заказ и приступил к его выполнению.

Проверка наличия товара и заказ у производителя

Toy Car Parts должна проверить возможность поставки заказанного товара со своего склада и в случае отсутствия заказать недостающий объем. На рис. 12-10 показано, как компания Toy Car Parts взаимодействует с поставщиками. Схема описана в табл. 12-2.

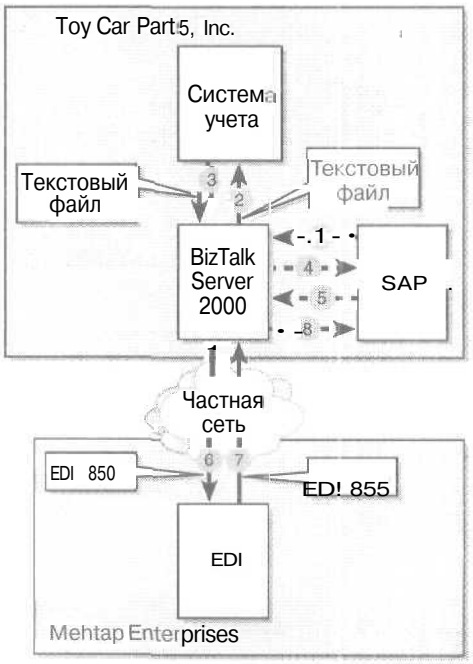


Рис. 12-Ю. Взаимодействие поставщика с производителем.

Табл. 12-2. Описание схемы на рис. 12-10.

Шаг	Описание
1	Система SAP посылает серверу BizTalk Server 2000 запрос в формате IDoc о наличии заказанного товара на складе.
2	Сервер BizTalk Server 2000 преобразует документ в формате IDoc в текстовый файл, который использует складская система. Система SAP не работает с таким форматом, поэтому приходится использовать возможности сервера BizTalk Server 2000.
3	Складская система сообщает, что на складе всего 3 запрошенных моторчика.
4	Сервер BizTalk Server 2000 преобразует ответ в формате IDoc и передает его системе SAP.
5	Система SAP знает, что нужны 10 моторчиков, а в наличии только 3, поэтому нужно создать заказ производителю на поставку еще *1* моторчиков. Этот запрос создается и пересылается серверу BizTalk Server 2000.
6	Производитель моторчиков работает с документами в формате EDI 850, поэтому сервер BizTalk Server 2000 преобразует созданный системой SAP документ в формате Шос в формат EDI 850 и передает его через внешнюю сеть поставщику — компании Mehtap Enterprises.
7	EDI-система Mehtap Enterprises обрабатывает запрос и высылает подтверждение в формате EDI 855. Теперь ей осталось лишь выполнить заказ. Компания Mehtap Enterprises даже не догадывается о существовании BizTalk, поскольку BizTalk Server 2000 выполняет все преобразования.
8	Сервер BizTalk Server 2000 преобразует подтверждение в формате EDI 850 в формат IDoc и пересылает его системе SAP.

Пересылка подтверждения заказчику

На заключительном **этапе** в **Toi** Carz **поступает подтверждение, что** заказ **принят, и** сообщается **ожидаемая** дата его **исполнения. Этот процесс проиллюстрирован на рис.** 12-11 **и описан в** табл. 12-3.

Табл. 12-3. Описание схемы на рис. 12—11.

Шаг	Описание
1	Система SAP создает уведомление в формате IDoc и посылает его серверу BizTalk Server 2000.

(продолжение)

Шаг	Описание
2	Сервер BizTalk Server 2000 в компании Toy Car Parts преобразует уведомление в документ XML и посылает его серверу BizTalk в компании Toi Carz.
3	Сервер BizTalk Server 2000 в Toi Carz преобразует документ XML в формат, понятный ПО Great Plains. Теперь результат прохождения запроса учтен, и служащие Toi Carz могут работать дальше,

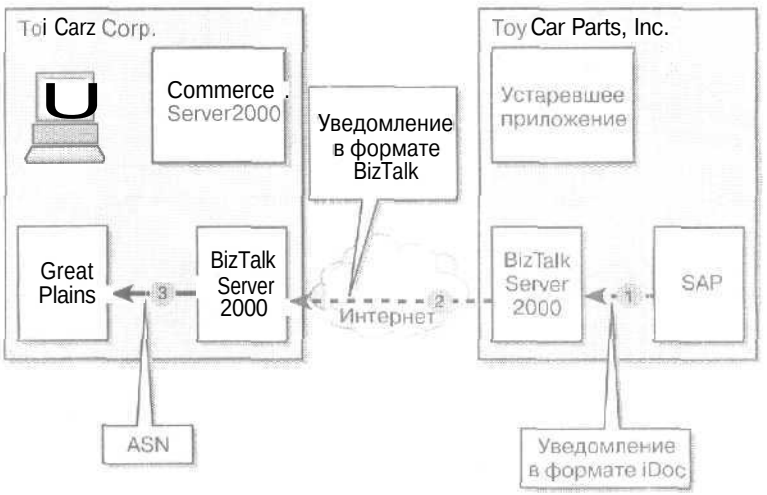


Рис. 12-31, *Передачауведомления в организацию, приславшую запрос.*

Обратите внимание на несколько важных деталей. Во-первых, нет нужды изменять существующие приложения (бухгалтерское ПО Great Plains, систему SAP, складскую систему) — сервер BizTalk выступает как универсальный переводчик, который преобразует запросы и подтверждения в форматы, описанные в соглашении между партнерами.

Во-вторых, Интернет можно использовать для осуществления транзакций по всему миру. Это значит, что вы можете сотрудничать с любой компанией, подключенной к Интернету, заключив с ней соответствующее соглашение.

В-третьих, вместо Интернета можно использовать любые существующие сети.

Часть IV

Справочные материалы

В приложениях вы найдете дополнительные материалы, которые я посчитал нужным включить в книгу, но не нашел им места среди глав. Приложение А поможет вам быстро освоиться с языком OmniMark, который играет важную роль в разработке BFC-сервера в главе 11. Приложение Б, «Спецификация инфраструктуры BizTalk», и приложение Г, «Спецификация протокола SOAP 1.1», полезны и сами по себе, и помогут вам разобраться примерах кода глав 8 и 11. В приложении В вы найдете список Web-сайтов, посвященных XML и BizTalk. Я регулярно посещаю эти сайты; думаю, у вас это тоже войдет в привычку.

Приложение А

OmniMark для нетерпеливых

OmniMark использован в этой книге как пример языка кросс-платформенной разработки с мошной поддержкой современных сетевых сред. Я использую OmniMark вот уже 10 лет и нахожу его достаточно стабильным, эффективным и хорошо поддерживаемым. OmniMark изначально предназначался для операций, связанных с обработкой текста, и с тех пор держится в группе лидирующих технологий программирования.

Сегодня OmniMark может рассматриваться как серьезная альтернатива Perl в сфере серверного программирования. Рекламные материалы утверждают, что Omni Mark «дешевле, чем Perl». Учитывая, что Perl распространяется бесплатно, это звучит довольно странно, не так ли? Хотя не совсем. Если вы *уже* видели образцы кода на Perl, то. возможно, заметили, что он довольно сложен для зрительного восприятия. Эти трудности связаны со специфическим синтаксисом выражений, массой скобок и других разделителей.

Если вы программируете на Perl, я хочу задать вам один вопрос: когда вы последний раз правили программу, написанную кем-то другим? Не просто было, да? Я знаю много Perl-программистов, предпочитающих переписать чужой кол, но не исправлять его. А как насчет программы, написанной вами полгода назад? Переработайте-ка ее сегодня. Мне кажется, что, если речь заходит об эксплуатации программ, *общая стоимость владения* (Total Cost of Ownership, TCO) программ на Perl гораздо выше.

Как и Perl, OmniMark — бесплатный продукт, но это только одно из его достоинств. OmniMark использует синтаксис сопоставления с образцом, основанный на грамматике английского языка, простого

для понимания и применения. Мой опыт показывает, что общая стоимость цикла разработки программ на OmniMark значительно ниже, чем на Perl, так как поддержка программ на OmniMark намного проще. Кроме того, в многопроцессорной многосерверной среде OmniMark работает гораздо быстрее Perl, поскольку не использует режим постоянной загрузки интерпретатора в память. Perl-программы обычно вызываются через интерфейс CGI. При вызове нужно загрузить интерпретатор Perl, который обрабатывает сценарий. По завершении работы интерпретатор выгружается из памяти, и эта процедура повторяется при каждом вызове. Фильтры 1SAPI и другие новые методы позволяют избежать такого варианта обработки, но, по моему опыту, используют их очень немногие программисты. В OmniMark же эти возможности встроены изначально.

Потоковый язык программирования

Потоковые языки программирования, такие как OmniMark, созданы для того, чтобы дать программисту возможность непосредственной обработки входного потока и немедленного создания выходного потока, минимизируя необходимость создания промежуточных структур данных. Потоковые языки особенно полезны в Интернете и интрасетях, где потоковые данные наиболее распространены.

В области сетевого программирования у потоковых языков есть три главных преимущества: эффективность, надежность и производительность. Они эффективны, так как минимизируют операции копирования и перемещения данных. Они повышают надежность программ, сокращая использование переменных и промежуточных структур данных, что снижает возможность возникновения ошибок. Потоковые языки повышают производительность, воспитывая стиль программирования, ориентированный на обработку данных. Этот стиль предусматривает ясное и четкое описание процессов обработки данных в программе.

Как и в любом потоковом языке, в OmniMark обработка входных и выходных потоков реализована на уровне базовых средств языка. В OmniMark применяется концепция абстрактных источников данных и потоков вывода, так что программист работает только с универсальными источниками данных и потоками вывода OmniMark. Программист связывает источники и потоки с файлами и портами, используя базовые средства языка или *компоненты расширения OmniMark* (OmniMark extension, OMX). Обработка и вывод данных, таким образом, полностью независимы от конкретного типа источника и адресата данных.

OmniMark использует понятия текущего устройства ввода и вывода. Для обработки источника данных нужно объявить его текущим устройством ввода; для вывода данных достаточно объявить соответствующий поток текущим устройством вывода. Эта модель ввода-вывода заметно упрощает обработку, поскольку нет нужды указывать. какие именно данные обрабатываются и куда именно адресуется исходящий поток. Вы просто определяете соответствующие потоки и обрабатываете данные в процессе их поступления. Вы можете обрабатывать данные при помощи операций сканирования или синтаксического анализа. Сканирование использует методы поиска по шаблону OmniMark, а синтаксический анализ — встроенные анализаторы XML и SGML.

OmniMark-программа состоит из так называемых *правил.* Правила обработки применяются для выбора текущих устройств ввода и вывода и для запуска синтаксического анализа или сканирования. Для сканирования служат правила поиска; такое правило содержит шаблон, с которым сравниваются данные, и набор инструкций, выполняемые в случае соответствия. Другие инструменты сканирования применяются для локального сканирования, манипулирования переменными и для работы с различными аспектами разметки. Правила разметки включают правила для обработки элементов, которые срабатывают каждый раз, когда анализатор находит элемент в данных XML или **SGML,** правила для обработки данных, содержащихся в элементе, и правила обработки ошибок разметки, которые служат для перехвата и обработки ошибок.

Приведенная ниже простая программа на OmniMark считает слова в тексте *«Заходит гусь в бар»*:

```
global integer word-count initial {0}
process
   submit "Заходит гусь Б бар"
   output "d" % word-count || "%n"
find letter+
   increment word-count
```

Программа содержит два правила и одно объявление глобальной переменной. Правило обработки срабатывает при запуске программы. Оператор *submit* пересылает в текущий поток ввода текст *«Заходит гусь в бар»* и инициирует сканирование.

Правило *find* активизируется оператором *submit.* Оно использует шаблон *letter+* для поиска любой последовательности букв. Слово *letter* в этом шаблоне — символьный класс, представляющий все бук-

вы алфавита в верхнем и нижнем регистре. Знак «плюс» (+) — индикатор повтора. Вместе они означают «одна или более букв».

Каждый раз, когда срабатывает правило *find*, глобальная переменная *word-count* увеличивается на 1. Правило *find* срабатывает один раз для каждого слова (т. е. непрерывной последовательности букв) в текущем потоке ввода. По окончании сканирования выполняется следующий оператор правила обработки — *output*: он выводит значение счетчика слов. Если текущий вывод не был явно определен, по умолчанию подразумевается стандартный поток вывода. В интегрированной среде разработки (Integrated Development Environment, IDE) Omni Mark вывод направляется в окно журнала, а при работе с утилитами командной строки — на экран терминала.

Оператор *output* использует оператор форматирования (%) для преобразования значения целой (integer) переменной *word-count* в строку. Оператор конкатенации (||) служит для добавления символа новой строки *«%n»*.

Чтобы попробовать выполнить эту программу, введите ее в окне IDE OmniMark и щелкните кнопку «Run». Порядок следования правил в этой программе значения не имеет, так как каждое правило срабатывает только при возникновении определенного события. Поэтому мы без проблем могли бы написать ее так:

```
global integer wordcount initial {0}
find letter+
   increment wordcount
process
   submit "Заходит гусь з бар"
   output "d" format wordcount || "%n"
```

Эта программа работает точно так же, как и предыдущая. Сказанное вовсе не означает, что порядок следования правил в OmniMark-программе никогда не имеет значения. Если одно событие вызывает срабатывание более чем одного *правила find*, правило, встретившееся первым, срабатывает, а все следующие — нет. Это позволяет размещать более конкретные правила перед более общими и добиваться срабатывания общих правил, только если не сработали конкретные. Рассмотрим пример:

```
global integer wordcount initial {0}
   process
      submit "Заходит гусь в баз"
      output "d" format wordcount || "%n"
```

```
find "гусь"
   output "*"
find letter+
   increment wordcount
find any
```

Эта программа выведет значение *3. В следующей программе порядок следования правил *find* изменен и, соответственно, меняется результат:

```
global integer wordcount initial {0}
process
   submit "Заходит гусь в бар"
   output "d" format wordcount || "%n"
find letter+
   increment wordcount
find "гусь"
   output "*"
findany
```

Программа выводит значение *4*.

Зачем я добавил новое правило *find any?* Затем, что оно исправляет ошибку, допущенную в первой версии программы. Правило *find leiter+* ищет слова. Но как быть с пробелами, разделяющими слова? Если вы запускали первую версию программы, то заметили, что результат был напечатан с отступом в три пробела. Это и были те самые неучтенные три пробела. Любой ввод, для которого правилами *find* не найдено соответствия, перенаправляется прямо в поток вывода. Правило *find any* в конце набора правил *find* исполняет роль своего рода губки, «впитывающей» весь нераспознанный ввод. Разумеется, конструкция *find any*, если вы ее используете, всегда должна быть последней в наборе правил *find*.

Обработка XML

OmniMark обрабатывает XML (и SGML) потоковым способом. Данные XML из источника поступают синтаксическому анализатору. Структурные детали, такие как элементы, вызывают срабатывание правил разметки. Если поток данных не перехватывается или не перенаправляется вашей программой, он направляется в текущее устройство вывода. Инструкции OmniMark *do xml-parse* и *do sgml-parse* перенаправляют ввод соответствующему процессору разметки. К примеру, следующий фрагмент инициирует обработку файла Myfile.xml процессором разметки синтаксического анализатора XML:

```
do xml-parse instance
   scan file "Myfile.xml"
   output "%c"
done
```

В процессе обработки разметки вы должны учитывать все уровни вложенности — ведь элемент может включать другие элементы. Любой элемент в потоке данных имеет три аспекта: начало, содержимое и коней. Чтобы дать вам возможность определить порядок действий при обработке содержимого, процесс обработки разметки останавливается на каждом элементе. **Вы** должны явно указать продолжение работы.

Как же продолжить анализ? Для этого OmniMark использует управляющую последовательность *%c*. Считайте это эквивалентом инструкции *continue xml-parse*. Поскольку для генерации вывода, соответствующего содержанию текущего элемента, нужно будет продолжить анализ, этот оператор имеет вид простой управляющей последовательности *%c*, которую вы можете вывести в строку текста. Инструкция *do xml-parse* просто устанавливает анализатор в некое начальное состояние для обработки ввода. Чтобы процесс начался, нужно вывести оператор продолжения анализа *%c*:

```
do xml-parse document
   scan file "daffy.xml"
   output "Начало %c Конец"
done
```

Каждое правило разметки обязано использовать оператор продолжения анализа *%c* или его альтернативу — оператор *suppress,* который продолжает анализ, подавляя вывод анализируемого содержимого в поток вывода. По очевидным причинам в каждом правиле разметки может быть не более одного оператора *%c* или *suppress.* Не делайте частой ошибки — не думайте, что *%c* служит для подстановки значения содержимого элемента. Это абсолютно неверно. Любой вывод, получаемый с помощью *%c*, — либо поток данных, напрямую направленный в текущий вывод, либо результат выполнения правил синтаксического анализа содержимого элемента.

Создание областей ввода

Для обработки любых данных в программе на OmniMark нужно определить область ввода, в которой находятся данные, и затем сканировать или анализировать эти данные. Можно создать текущую область ввода, используя оператор *using input as* и имя источника данных:

```
process
   using input as file "daffy.txt"
         submit #current-input
```

Встроенная переменная *#current-input* принимает значения источника для текущей области ввода. Вы можете упростить этот процесс, указав имя источника данных прямо в операторе *submit.*

```
Process
   submit file "daffy.txt"
```

В предыдущем примере оператор *submit* создает текущую область ввода и инициирует процесс сканирования. Однако встроенная переменная *#current-input* полезна, если вы хотите продолжить обработку текущей области ввода в другой операции сканирования. Ниже оператор *do scan* в правиле *find* берет на себя выполнение сканирования в текущей точке, выводит некие данные, а затем позволяет исходному процессу продолжить обработку:

```
Process
   submit file "sample.htm"
find ul "<table>"
   do   scan  flcurrent-ir.put
         match any++ => table-body ul "</table>"
            output table-body
   done
```

Текущая область ввода устанавливается и оператором *do xml-parse.* Параметр *scan* описывает источник данных.

```
do xml-parse document
   scan file "ducks.xml"
   output "%c"
done
```

Создание областей вывода

Весь вывод программ на OmniMark по умолчанию направляется на текущее устройство вывода. Вы можете выбрать другое устройство вывода внутри правила.

```
element duck
   local stream bar
   open bar as file "Bar.txt"
```

```
using output as bar do
        output "кря кря кря %c"
done
```

Правило временно назначает текущим устройством вывода файл Bar.txt. Любой вывод внутри блока операторов *do...done* согласно оператору *using outputas* будет направляться в файл. По окончании работы блока текущим вновь станет устройство по умолчанию. Заметьте, однако, что оператор *output* содержит оператор продолжения *анализа* (*%c*), так что новое устройство вывода будет действовать для всех операций обработки элемента *duck*. Оно становится текущей областью вывода для всех операторов *output,* выполняющихся в любом правиле разметки, которое срабатывает в результате анализа элемента *duck*.

Чтобы понять, как работает процесс создания области вывода, рассмотрим файл XML Duck.xml. Он содержит схему и следующую разметку:

```
<line>
А <player>Утка</player> подходит к <player>бармену</player>
</line>
```

Теперь рассмотрим такую программу на OmniMark:

```
global stream words
global stream players

process
    open words as file "words.txt"
    open players as file "players.txt"
    using output as words
    do xml-parse instance
            scan file "duck.xml"
            output"%c"
    done
element line
    output "%c"
element player
    using output as players do
            output "%c "
    done
```

Выполнив эту программу, мы получим два файла: words.txt будет содержать фразу *« подходит к »*, *а* people.txt — *« Утка бармену »*. Хоро-

шенько рассмотрите этот код, чтобы убедиться, что вы действительно понимаете, какое устройство вывода будет текущим в правиле *element line*. Вам ясно, почему вывод в файлы выполнен именно так? Если да, то вы знаете достаточно, чтобы понимать, как OmniMark управляет выводом. Нет? Не переживайте. Такая иерархическая программная модель — новинка для большинства программистов. Так как XML-документ — тоже иерархический объект, эта модель имеет важное значение для обработки подобных документов. Просто помните, что модель вывода имеет иерархическую структуру. Поработав с OmniMark или с другим иерархическим языком программирования, вы начнете понимать, как это происходит.

Последнее правило предыдущей программы можно значительно упростить, заменив оператор *using output as* в блоке оператором *put:*

```
element player
   put players "%c"
```

Здесь оператор *put* устанавливает поток *players* в качестве текущей области вывода и выводит в нее строку *«%c»*. Эта область вывода будет действительной для всех правил, срабатывающих в результате выполнения *«%c»*.

Текущий вывод OmniMark — мощный механизм, который упрощает код, устраняя необходимость постоянного объявления вывода. Как только вы устанавливаете место назначения текущего вывода, весь выходной поток направляется в него, если только вы явно не объявляете о перенаправлении в другом месте кода. Заметьте: такое положение подразумевает, что оператор *output* может выполняться в разных областях текущего вывода в зависимости от контекста, в котором он встречается. Способность отделять процесс создания выходного потока от процесса объявления места назначения вывода — мощная и гибкая особенность программирования на OmniMark.

Текущий вывод имеет дополнительную возможность поддержки более чем одного адресата одновременно. Этот фрагмент кода поместит строку «Заходит утка в Меня не пускают!» и в файл outfile.txt, и в переменную *my-buffer*:

```
global stream my-file
global stream my-buffer
process
   open my-file as file "outfile.txt"
   output-to my-file
   submit "Заходит гусь в бар"
```

```
find "бар"
   open my-buffer as buffer
   using output as my-buffer and #current-output
   do
         output "Меня не пускают!"
   done
```

Ключевое слово *#current-output* действует для любого адресата текущего вывода. Вы можете использовать его, чтобы добавить новый адресат к текущему активному.

Компоненты OMX

OmniMark поддерживает использование файлов и констант в выражениях языка. Компоненты OMX обеспечивают ввод данных из других источников и обеспечивают расширение функциональности программ OmniMark. Компоненты применяются для подключения к внешним источникам данных, для создания и управления соединениями по соответствующим протоколам, а также обеспечивают интерфейсы для подключения OmniMark-программ, представляющих собой либо исходный код, либо откомпилированные модули. Вы можете обрабатывать источник, либо связывать потоки данных с модулем и объявлять потоковую часть текущей области вывода.

Две следующие программы создают клиент-серверное приложение, осуществляющее ввод и вывод данных. Первая, server.xom, создает TCP/IP-соединение и ожидает событий. Вторая, client.xom, также создает TCP/IP-соединение и связывает с ним источник и поток, чтобы читать и отправлять данные через это соединение:

```
include "omtcp.xin"
process
      ;запуск службы
      local TCPService service initial {TCPServiceOpen at 5436}
      ;цикл сервера
      repeat
            ;получение данных о соединении от клиента
            local TCPConnection connection initial
                        {TCPServiceAcceptConnection service}
            ;получение строки запроса и вывод
            put tterror TCPConnectionGetSource connection
                        protocol IOProtocolMultiPacket || "%n"
```

```
            :send a response
            set TCPConnectionGetOutput connection
                        protocol IOProtocolMultiPacket
                        to "<foo><bar>text</bar><baz bat='gruznatz'/
></foo>"

            catch #program-error
      again
```

А это код приложения-клиента:

```
include "omtcp.xin"
include "builtins.xin"
global stream server-host initial {"localhost"}
global integer server-port initial {5436}
declare catch connection-error
define switch function TCPconnection-is-working
            read-only TCPconnection connection
      as
      do when TCPConnectionIsInError connection
            local stream error-report variable
            TCPConnectionGetStatusReport connection into error-report
            log-message "TCP connection error:"
            repeat over error-report
                        log-message error-report
            again
            returr false
      else
            return true
      done
process
      local TCPConnection connection
      local stream output-file
      local stream request
      :открытие соединения
      set connection to TCPConnectionOpen
            on server-host
            at server-port
      throw connection-error
            unless TCPconnection-is-working connection
      :запрос
```

```
set TCPConnectionGetOutput connection
        protocol IOProtocolMultiPacket
        to "<request><typelist/></request>"
:обработка ответа
do xml-parse instance
        scan TCPConnectionGetSource connection
                    protocol IOProtocolMultiPacket
        put #error "%c"
done
catch connection-error
        put tterror "Unable to connect%n"
element ffimplied
    put fie-ror "%n[%q]%c[/%q]"
```

Запустив приложение-сервер в одном командном окне, а приложение-клиент — в другом, понаблюдайте за их взаимодействием. В этих программах функция *TCPConnectionOpen*, которую вы найдете в библиотеке OmniMark omtcp, создает соединение с сервером TCP/IP и возвращает переменную OMX (соединение) типа *TCPConnection*— дескриптор объекта OMX *TCPConnection,* который управляет соединением.

Функция *TCPConnectionGetOutpu* связывает поток OmniMark (запрос) с соединением. Вы можете использовать этот поток для отправки данных серверу. Программа генерирует соответствующий запрос, сканируя первый аргумент командной строки (*args[1]*) Вывод, созданный правилами *find,* которые вызываются оператором *submit*, будет направляться серверу.

Функция *TCPConnectionGetSourc* возвращает источник OmniMark, связанный с соединением TCP/IP. Это источник, определенный в параметре *scan* инструкции *do xml -parse*, подразумевает, что данные, полученные от сервера, будут сразу направляться анализатору XML.

Переменные и типы данных

OmniMark поддерживает встроенные и внешние типы данных (последние — с помощью компонентов OMX). Типы данных *stream*, *switch* (логический) и *integer* являются встроенными. В OmniMark нет строкового типа — для обработки таких данных служит переменная *stream.* Дополнительные числовые типы *BCD* — десятичное число в двоичном виде (Binary Coded Decimal) и *float* — реализуются компонентами OMX- Другие типы переменных OMX представляют множество различных внешних источников данных.

Переменные могут быть глобальными или локальными. Первые действуют на протяжении всего времени выполнения и доступны в любом месте программы. Вторые доступны только в том правиле или функции, где они объявлены.

Поскольку мы не можем использовать переменные без предварительного объявления, глобальные переменные обычно объявляются в самом начале программы OmniMark, а локальные — в начале правила или функции, в которых планируется их использовать. При объявлении переменной надо указать область ее действия.

Объявление глобальной целой числовой переменной *word-count* выглядит так:

```
global integer word-count
```

а локальная потоковая переменная *quotation* — так:

```
local stream quotation
```

Для записи строки в потоковую переменную служит ключевое слово *set,* например:

```
set quotation to "У вас есть виноград?"
```

Для присвоения значения числовой переменной можно использовать оператор *set*, как и для потоковых переменных, но, кроме того, вы можете использовать операторы *increment* и *decrement*. Например, для увеличения значения переменной *count1* на 1, достаточно применить оператор:

```
increment count1
```

Вы можете увеличивать или уменьшать значение числовой переменной на значение другой числовой переменной. Вот как, к примеру, уменьшить значение переменной *count1* на величину значения переменной *count2*:

```
decrement count1 by count2
```

Следующая программа использует глобальную переменную типа *switch* для определения, какая именно инструкция *output* должна быть выполнена.

```
global switch question
process
   set question to true
   do when question      ;проверка если question содержит true
```

```
        output "быть"
    else
        output "иль не быть"
    done
```

Обратите внимание, что программа всегда выводит строку «быть».

При объявлении переменной можно присвоить начальное значение:

```
global integer count2 initial {3}
global stream quotation2 initial {"Кряква!"}
global switch status2 initial {true}
```

Вы можете присваивать переменной значение другой переменной. Так, правило *process* в следующей программе установит значение глобальной переменной *varl* равным значению локальной переменной *var2* и выведет число *8*.

```
global integer var1
process
    local integer var2
    set var2 to 8
    set var1 to var2
process
    output "%d(var1)"
```

Наконец, вы можете инициализировать переменную динамическим выражением. К примеру, вместо того, чтобы писать:

```
local TCPConnection connection
set connection to TCPConnectionOpen
    on server-host
    at server-port
```

переменную можно инициализировать динамически:

```
local TCPConnection connection initial
 {TCPConnectionOpen
      on server-host
      at server-port
 }
```

Ввод-вывод и переменные

В OmniMark есть три ключевых слова для вывода данных: *set, output* и *put*. Ключевое слово *set* — единственное, которое можно использовать

с числовыми типами данных. С потоковыми переменными допустимы все три. Потоковые переменные обслуживают как сами потоки, так и строковые переменные. При работе со строковыми переменными потоковая переменная фактически имитирует потоки, работающие с буфером, который OmniMark обрабатывает как источник данных или адресата, подобно остальным потокам.

Ключевое слово *output* направляет данные в поток, открытый оператором *open* и объявленный частью текущей области вывода инструкцией *using output as.*

Ключевое слово *put* работает точно так же, как и оператор управления *using output as,* позволяя изменять текущий вывод и произвести одиночную операцию вывода.

Ключевое слово *set* выполняет функции как оператора *open,* так и оператора *using output as:* оно открывает поток, объявляет его текущим и выполняет одиночную операцию вывода. Это означает, коль уж мы говорим о потоках, что для работы с переменными и внешними источниками данных нет различных операторов. Напротив, есть набор операторов, которые вы можете соответствующим образом использовать как для работы с переменными, так и с внешними источниками данных. Например, вы можете применить оператор *set* для вывода значения в файл:

```
set file "duckbar.txt" to "Заходит гусь Б бар."
```

Точно так же операторы *open, put и close* позволяют присвоить значения переменной:

```
local stream Duck
open Duck as buffer
put Duck "Заходит гусь в бар."
close Duck
```

Преимущество этого синтаксиса в том. что он позволяет выводить данные в поток много раз перед тем, как его закрыть. Это более простой и эффективный путь, чем создание строки при помощи конкатенации. Например, вы можете заменить код:

```
set Duck to "Заходит гусь в бар"
set Duck to Duck || "Бармен говорит ему: 'Гусям не подаем'"
set Duck to Duck || "Гусь: 'А я не за милостыней, а за пивом.'"
```

таким:

```
open Duck as buffer
```

```
using output as Duck do
        output "Заходит гусь в бар"
        output "Бармен говорит ему: 'Гусям не подаем"
        output "Гусь: 'А я не за милостыней, а за пивом.'"
done
```

Эта чрезвычайно мощное свойство OmniMark позволяет выбирать механизм присвоения значений, соответствующий масштабу ваших операций. Вы можете применять оператор *set для* единичных операций присваивания. Для крупномасштабных операций используйте оператор *output с* любыми файлами или потоковыми переменными и выполняйте нужные действия, не беспокоясь об указании адресата или его типа. Выбор механизма, соответствующего объему операций, значительно упростит ваш код.

Прежде чем поток будет прочитан или выведен, его нужно закрыть:

```
open Duck as buffer
put Duck "Заходит гусь в бар."
close Duck
output Duck
```

Вы можете использовать оператор *reopen* для повторного открытия закрытого потока с оригинальным содержанием. Однако, если вы используете оператор *open* для нового открытия потока, вы потеряете его содержимое:

```
open Duck1 as buffer
open Duck2 as buffer
put Duck1 "Заходит гусь в бар."
put Duck2 "Заходит гусь в бар,"
close Duck1
close Duck2
reopen Duckl
open Duck2
put Duck1 "Подходит она к бармену"
put Duck2 "Подходит она к бармену"
```

После выполнения приведенного кода поток Duckl будет содержать обе строки, а Duck2 — только строку «Подходит она к бармену».

Структуры данных

OmniMark использует структуры данных, называемые *полками*. Вы можете ссылаться на данные на полке, используя их индекс или текстовый ключ. Полки могут иметь фиксированный или переменный размер. Все переменные OmniMark — фактически полки, но, так как обычная переменная является полкой с фиксированным размером и с одним элементом на ней, для нее это не имеет особого значения.

Объявление глобальной потоковой полки переменного размера под названием *quotations* выглядит так:

```
global stream quotations variable
```

По умолчанию полка переменного размера содержит 0 элементов. Объявление локальной пелочисленной полки, содержащей 3 элемента, выглядит так:

```
local integer totals size 3
```

Чтобы создать полку с начальным значением, отличающимся от значения по умолчанию, добавьте ключевое слово *initial* в объявление, а за ним — значения в скобках. Например;

```
global integer sizes size 4 initial {1, 2, 3, 4}
```

Это объявление создает глобальную целочисленную полку с названием *sizes*, которая может содержать 4 элемента с начальными значениями *1, 2, 3* и *4*. Вы также можете создать полку переменного размера, содержащую ряд начальных значений:

```
global integer sizes variable initial {1, 2, 3. 4}
```

Разница между этими полками лишь в том, что первая имеет фиксированный размер и может содержать только 4 элемента, а вторая начинается с четырех элементов, имеющих начальные значения, и может быть расширена или уменьшена до необходимого числа элементов.

Кроме того, можно создавать полки специфического размера без указания начальных значений элементов. Для этого используется ключевое слово *initial-size*:

```
global integer sizes variable initial-size 4
```

Это объявление создает целочисленную полку из 4 элементов, которая при необходимости может быть увеличена или уменьшена.

Вот как присвоить потоковой полке переменной величины *quotations* значение строки «Зима тревоги нашей»:

```
set new quotation to "Зима тревоги нашей"
```

Для доступа к элементам на полке вы должны выбрать метод индексации: либо по номерам элементов, либо по ключу. Номера элементов берутся в квадратные скобки, ключи — в фигурные.

```
set quotation[3] to "Слова, слова, слова."
output quotation{"Гамлет"}
```

Вы можете назначить ключ для существующего элемента:

```
set key of quotation[2] to "Ричард III"
```

Вы можете установить ключ при создании нового элемента. Например, для создания нового элемента на полке *quotes,* имеющего значение «Увы, бедный Йорик.», с ключом «Гамлет» можно поступить так:

```
set new quotes{"Гамлет"} to "Увы, бедный Йорик."
```

По умолчанию новый элемент добавляется в конец полки. Разместить новый элемент в нужном месте можно с помощью ключевых слов *before* или *after.* Например, для создания нового элемента прямо перед вторым сделайте следующее:

```
set new quotes before [2] to "Коня!"
```

Это действие создает новый элемент, имеющий значение «Коня!» и расположенный между первым и вторым элементами полки *quotes.* Поскольку номера элементов основываются на их позиции на полке. новый элемент приобретает значение индекса 2, а бывший элемент с этим индексом — значение индекса 3.

Вот как создать новый элемент, расположенный сразу после элемента с ключом «Макбет»:

```
set new quotes{"Ричард III"} after {"Макбет"} to "Коня!"
```

Следующая программа, иллюстрирующая концепцию полок OmniMark, создает потоковую полку и присваивает значение первому элементу этой полки. Затем программа устанавливает ключ для этого элемента. После этого создаются еще три элемента: один в конце полки (по умолчанию), второй — перед вторым элементом и третий — после элемента с установленным ключом.

```
Process
   local stream quotes variable
```

```
set new quotes{"Гамлет"} to "Быть иль не быть?"
set new quotes{"Макбет"} to "Кто ты, кинжал?"
sot new quotes{"Ричард III"} before [2] to "Коня!"
set new quotes{"Ромео"} after {"Ричард III"}
        to "Но тише! Что за свет пробился сквозь окно?"

repeat over quotes
        output key of quotes
                || " - "
                || quotes || "%n"
again
```

Программа создает следующий вывод:

```
Гамлет - Быть иль не быть?
Ричард III - Коня!
Ромео - Но тише! Что за свет пробился сквозь окно?
Макбет - К~о ты, кинжал?
```

Ссылки

Ссылочными называются переменные, которые могут быть выведены до того, как им будет присвоено значение. Ссылочные переменные позволяют поместить «подстановочные» переменные в выходной поток и позже присвоить им значения или изменить их. Следующий фрагмент иллюстрирует использование ссылочных и подстановочных переменных:

```
process
        output "Прощай, " || referent "world-type" || " мир!"
        set referent "world-type" to "жестокий"
```

Результатом работы этой программы будет строка вывода «Прощай, жестокий мир!».

Следующий пример чуть сложнее:

```
process
    local stream fowl
    set fowl tc "Утка%n"
    set referent "water" to "Утка%n"
    output fowl
    output referent "water"
    set fowl to "бар%n"
    set referent "water" to "бар%n"
```

В результате работы программы будут выведены строки:

```
Утка
Баб
```

Результат вывода потоковой переменной *fowl* не изменился после изменения ее значения; изменение значения ссылочной переменной, напротив, изменило результат вывода.

Заметьте: в то время как потоковая переменная *fowl* была объявлена до ее использования, ссылочная переменная *water* не была объявлена. Для создания и применения ссылочной переменной нужно лишь дать ей имя и присвоить значение.

Еще один простой пример использования ссылочных переменных — вывод номеров страниц документа, содержащего *n* страниц, например. *страница 1 из 8.* Пока документ не обработан полностью, вы не знаете, сколько страниц он содержит. Однако, используя ссылочную переменную, вы можете просто поместить подстановочную переменную в поток вывода и выводить текущий номер страницы при необходимости. Когда документ будет обработан окончательно и количество страниц определено, ссылочной переменной будет присвоено конечное значение номера страницы.

Следующая программа изменяет нумерацию в списке пьес, содержашемся в текстовом файле. Каждый раз, найдя число в начале строки, программа модифицирует ссылочную переменную *play-count* и выводит текущий номер пьесы и ссылочную переменную. Хотя ссылочная переменная модифицируется и вывод происходит каждый раз, когда срабатывает правило, в потоке вывода присутствует только конечное значение ссылочной переменной,

```
find line-start digit+ => play-number
   set referent "play-count" to play-number
   output "Пьеса "
          || play-number
          || " из "
          || referent "play-count"
process
   submit file #args[1]
```

Для входного файла:

```
1 Гамлет
2 Ричард III
3 Макбет
```

```
4 Ромео и Джульетта
5 Король Лир
```

программа выдаст следующий результат:

```
Пьеса 1 из 5     Гамлет
Пьеса 2 из 5     Ричард III
Пьеса 3 из 5     Макбет
Пьеса 4 из 5     Ромео и Джульетта
Пьеса 5 из 5     Король Лир
```

По умолчанию ссылочные переменные уничтожаются по окончании работы программы. Однако вы можете сделать это раньше, используя инструкцию *using nested-referents*:

```
find line-start digit+ => play-number
   set referent "play-count" to play-number
   output "Пьеса "
          || play-number
          || " из "
          || referent "play-count"
process
   using nested-referents
      submit file #args[1]
```

В этом примере ссылочная переменная *play-count* создается только для операции *submit*. Она будет уничтожена, как только завершится процесс сканирования, инициированный оператором *submit*, т. е. раньше, чем завершится сама программа.

Условные операторы

Практически любой оператор OmniMark можно сделать условным, просто добавив после него ключевые слова *when* или *unless*. Например, любое правило может иметь условие выполнения. Приведенное ниже правило будет выполняться, только когда значение *count1* будет равно *4*:

```
find "утка" when count1 = 4
   output "Я нашел утку%n"
```

Основной условной конструкцией является блок *do when...done*.

```
do when foo = 4
   output "Да, значение переменной 'foo' равно четырем%n"
done
```

Можно использовать дополнительную ветку *else*:

```
do when output-in-upper-case = true
    output "ug" % words j | "%n"
else
    output words || "%n"
done
```

В случае нескольких вариантов подойдет конструкция *else when*:

```
do when foo = 4
    output "Да, значение переменной 'foo' равно четырем%n"
else when count1 = 5
    output "Значение переменной 'foo' равно пяти%n"
else when count1 = 6
    output "Значение переменной 'foo' равно шести%n"
else
    output "Значение переменной 'foo' не равно 4, 5, или 6%n"
done
```

Операторы & (И) и | (ИЛИ) позволяют конструировать сложные условия:

```
do when (foo > 12 | foo < 56) & bar > 3
```

Еще одна форма условной конструкции — оператор *do select*.

```
do select foo
    case 1 to 5
        output "переменная 'foo' маленькая%n"
    case 50 to 100
        output "переменная 'foo' большая%n"
    else
        output "переменная 'foo' средняя%n"
done
```

Кроме того, логические конструкции можно применять в инструкции *do scan*, которая использует сканирование для разбора данных. Например, возможны условия, базирующиеся на сопоставлении с образцом:

```
Process
    do scan telephone-number
        match "(" digit{3} ")" digit{3} "-" digit{4}
```

```
            output "междугородный звонок%n"
        match digit{3} "-" digit{4}
            output "честный звонок%n"
        else
            output "неизвестный формат%n"
    done
```

Операторы цикла

Ornni Mark поддерживает четыре типа операторов цикла: *repeat, repeat for, repeat over* и *repeat scan.*

Цикл *repeat* предусматривает повторное выполнение содержащихся в нем операций до тех пор, пока не встретится явно заданное выражения выхода из цикла *exit.*

```
process
    local integer foo
    repeat
        output "переменная 'foo' равна " || "d" % foo || "%n"
        increment foo
        exit when foo - 4
    again
```

Цикл *repeat for* повторяется для каждого значения переменной цикла.

```
Process
    Local integer foo
    repeat for integer foo from 1 to 7 by 2
            output "переменная 'foo' равна "      || "d" % foo || "%n"
    again
```

Цикл *repeat over* повторяет выполнение для каждого элемента на полке:

```
Process
        local stream duckType variable
set new duckType key "Sarkidiornis melanotos" to "Гребешковая утка"
        set new duckType key "Anas crecca" to "Зеленокрылый чирок"
        set new duckType key "Melanitta perspicillata" to "Нырок турпан"
        repeat over duckType
            output key of duckType || "=" || duckType || "%n"
        again
```

В цикле *repeat over* ссылка на текущий элемент осуществляется по имени полки без индекса.

Оператор цикла *repeat scan* можно применять для сканирования данных точно так же, как и оператор *submit* и набор правил. Единственная разница: оператор *repeat scan* локален для отдельного правила или функции; не отобранные им данные не выводятся в выходной поток. Вместо этого цикл *repeat scan* прерывается при нахождении данных, не соответствующих шаблону:

```
process
   local integer word-count
   repeat scan "У Мэри был ягненок"
         match letter+
               increment word-count
         match any
   again
   output "d" % word-count
```

Арифметические операторы и операторы сравнения

OmniMark располагает полным комплектом арифметических операторов и операторов сравнения. Это арифметические операторы + (сложение), — (вычитание), * (умножение), / (деление) и *modulo* (остаток от деления). Дополнительные операторы для некоторых числовых типов данных поддерживаются компонентами OMX. OmniMark также поддерживает полный набор операторов сравнения. Взгляните на примеры:

```
process
   do when x = y
         output "равно%n"
   else when x > y
         output "больше%n"
   else when x < y
         output "меньше%n"
   done
```

Кроме того, в языке есть операторы сравнения != (не равно). >= (больше или равно) и <= (меньше или равно).

Сопоставление с образцом

OmniMark позволяет выполнять сканирование для поиска по шаблону во входных данных. Например, приведенное ниже правило *find* сработает, когда во входном потоке будет найдена строка *«Гамлет:»*:

```
find "Гамлет:"
   output "<b>Гамлет</b>: "
```

Однако такой подход требует написания отдельных правил *find* для каждого имени персонажа, которое мы хотели бы заключить в тэги HTML <b>, как показано ниже:

```
find "Гамлет:"
   output "<b>Гамлет</b>: "
find "Горацио:"
   output "<b>Горацио</b>: "
find "Бернардо:"
   outout "<b>Бернардо</b>: "
```

Вот здесь-то и вступают в игру шаблоны OmniMark. Богатые встроенные возможности поиска по шаблону OmniMark предоставляют возможность поиска строк, используя более абстрактную модель, чем поиск по заданной строке. Это иллюстрирует следующий пример — правило *find* будет искать любую строку с любым количество символов и непосредственно предшествующую двоеточию:

```
find letter+ ":"
```

К сожалению, шаблон, описанный в этом правиле *find*, недостаточен для безошибочного поиска только имен персонажей. При поиске будут найдены любые последовательности символом, предшествующие многоточию и расположенные в любом месте текста, включая слова в середине предложений.

Но в середине предложения слова редко начинаются с прописной буквы — это характерно для имен собственных. Зная это, мы можем внести дополнительные детали в наше правило *find*. Теперь оно будет искать любую строку, начинающуюся с прописной буквы (*uc*), содержащую минимум одну букву (*letter+*) и завершающуюся двоеточием (*:*):

```
find uc letter+ ":"
```

Однако, если бы перед нами стояла фактическая задача разметки ASCII копии «Гамлета», мы бы столкнулись с тем, что наше правило *find* находит только имена персонажей, состоящие из одного слова,

например, Гамлет. Офелия или Горацио. Если же имена персонажей состоят из двух частей, обнаруживается только вторая часть. Значит, такие имена, как Королева Гертруда и Лорд Полоний, были бы размечены неверно.

Чтобы найти эти составные имена так же, как и однословные, усовершенствуем правило *find,*

```
find uc letter+ (white-space+ uc letter+)? ":"
```

В этой версии правило *find* будет исследовать и второе слово перед двоеточием. Шаблон *(white-space+ uc letter+)?* находит один или более пробелов, следующую за ними прописную букву и одну или несколько любых букв. Эти изменения позволяют правилу *find* находить имена персонажей, состоящие как из одного, так и нескольких слов.

Если вы хотите найти последовательность из трех цифр, примените следующий шаблон:

```
find digit {3}
```

а чтобы найти 3- или 4-значное число, необходим такой:

```
find digit {3 to 4!
```

Найти дату в формате гг/мм/дд позволяет шаблон:

```
find digit {2} "/" digit {2} "/" digit {2}
```

Для поиска канадского почтового индекса пригодится шаблон:

```
find letter digit letter space digit letter digit
```

Ключевые слова *letter* и *uc*, использованные для создания предыдущих шаблонов, называются классами символов. Вот некоторые классы символов OmniMark:

- **letter** — одиночная буква в верхнем или нижнем регистре;
- uc — одиночный символ в верхнем регистре;
- **lc** — одиночная буква в нижнем регистре;
- **digit** — одиночная цифра (от 0 до 9);
- **space** — одиночный пробел;
- **blank** — одиночный пробел или символ табуляции;
- **white-space** — одиночный пробел, символ табуляции или символ новой строки;
- **any-text** — любой символ кроме символа новой строки;
- any — любой одиночный символ.

Вы можете определить собственный класс символов. Например, приведенное ниже правило *find* сработает, если во входном потоке будет обнаружен любой из четырех арифметических операторов:

```
find ["+-*/"]
   output "найден арифметический оператор%n"
```

Можно определить класс «все, кроме указанных», используя оператор исключения (\). Следующее правило *find* будет искать любые символы, кроме закрывающей фигурной скобки:

```
find [\ " Г]
```

Оператор исключения можно применять к встроенным классам символов. Такое правило *find* будет находить любые согласные буквы:

```
find [letter \ "aeiouAEIOU"]
```

Строки и встроенные классы символов можно объединять в одном правиле для создания новых классов символов, используя оператор *or*. Следующее правило поиска будет искать любой арифметический оператор или одиночную цифру:

```
find ["+-*/" j digit]
```

Приведенное ниже правило поиска ищет любой арифметический оператор или одиночную цифру, кроме символа *«0»*:

```
find ["+-*/" I digit \ "0"]
```

В любом шаблоне можно использовать операторы определения частоты вхождения символов:

- + — 1 или более;
- * — 0 или более;
- ? — 0 или один;
- ** — от 0 до...
- ++ — от 1 до...

Вы помните, что шаблон *letter+* ищет одну или более букв, *letter** — О или более, а *uc?* означает (I или одну прописную букву.

Поиск по шаблону в OmniMark требует аккуратности. Следующее правило никогда не сработает:

```
find "<table>" any* "</table>"
```

Дело в том, что конструкция *any** будет применяться для всего ввода, включая строку *«</table>»*. Поскольку часть шаблона *«</table>»*

никогда не будет найдена, поиск по шаблону всегда будет завершаться неудачей.

Для записи шаблона, который ищет символы до специфического разделителя, используйте операторы определения частоты вхождения символов ** или ++:

```
find "<table>" any** "</table>"
```

Здесь выражение *any*** находит 0 или более символов до строки *«</table>»*.

Следующее правило будет работать только при наличии минимум одного символа между строками *«<table>»* и *«</table>»* во входном потоке:

```
find "<table>" any++ "</table>"
```

Оператор *lookahead* можно применять для определения наличия шаблона во входных данных без их обработки:

```
find any++ lookahead "</table>"
```

Этот шаблон будет находить любое количество символов от текущей позиции до вхождения строки *«</table>»*, но не будет обрабатывать строку *«</table>»*, оставив ее во входных данных, где она может быть обработана другими правилами. Заметьте: строке *«</table>»* должен предшествовать хотя бы один символ, иначе либо правило завершится неудачей, либо поиск будет продолжен до вхождения следующей строки *«</table>»* в данных (разумеется, если она там есть).

Следующее правило решает эту проблему:

```
find any** lookahead "</table>"
```

Но это правило сработает дважды, если сработает вообще: первый раз, когда найдет все символы до *«</table>»*, и второй — когда будет считать символы, идущие за *«</table>»* (за его положением в источнике после первого вхождения). В принципе это правило должно было бы впасть в бесконечный цикл, находя 0 символов, за которыми следует строка *«</table>»*, без всякого продвижения вперед. Но OmniMark не допускает учета нулевого числа символов двум последовательными правилами, так что это правило не будет выполняться, пока не появятся новые данные.

Оператора частоты вхождения * и ** позволяют создавать шаблоны, находящие соответствия нулевой длины. OmniMark никогда не разрешит нахождение двух последовательных соответствий, не содержащих данных, но если вы заметите, что правило против ожидания

срабатывает дважды, это, возможно, происходит потому, что второй раз обнаружено соответствие нулевой длины, разрешенное при использовании операторов * и **.

Переменные шаблона

При применения шаблонов для поиска во входных данных бывает нужно сохранить результат в переменной для последующего использования. Такие переменные шаблона определяются символом =>, после чего на них можно ссылаться. Например, в первом правиле приведенного ниже примера найденные данные сохраняются в переменной шаблона *found-text*.

```
process
   submit "Утка [кряква] заходит в бар."
find ("[" letter+ "]") => found-text
   output round-text
findany
```

Вывод программы представляет собой строку *[кряква]*.

А что, если вы захотите вывести только слово, находящееся в квадратных скобках, а не сами скобки? Сделайте так:

```
Process
   submit "Утка [кряква] заходит в бар."
find "[" letter+ => found-text "]"
   output found-text
find any
```

Программа выведет строку *кряква*. В данном фрагменте переменная шаблона связана только с частью шаблона, предшествующей присвоению значения переменной. Таков режим работы переменной шаблона по умолчанию. Поэтому предыдущий пример будет работать некорректно, если только мы не заключим три элемента шаблона в круглые скобки, чтобы гарантировать нахождение текста, соответствующего шаблону в целом.

Шаблон может содержать более одной переменной шаблона. Переменные даже могут быть вложенными. Рассмотрим пример:

```
Process
   submit "Утка [кряква] заходит в бар,"

find        ("[" => first-bracket
   letter+ => found-word
```

```
    "]" => second-bracket) => found-text

    output  first-bracket
    output  found-word
    output second-bracket
    output  found-text
find any
```

Вывод этой программы представляет собой строку *[кряква][кряк-ва]*. Первое слово *«[кряква]»*является результатом работы первых трех операций вывода; второе — результатом работы четвертой операции.

Функции

Подобно большинству языков программирования, OmniMark поддерживает функции, но в отличие от других программы на OmniMark не являются иерархией функций. Правила — вот основные структурные элементы OmniMark-программ, а функции — вспомогательные. Функции не могут содержать правила (хотя могут вызывать их с помощью операторов *submit, do xml-parse* и *do sgml-parse*). Вы можете применять функции для локализации кода, совместно используемого различными правилами. Кроме того, функции можно задействовать для поиска по шаблону или внутри шаблонов для динамического определения строки поиска.

Функции локализуют фрагменты кода, но не изолируют вас от текущей среды, в частности, от текущей области вывода. Вывод функции направляется в текущую область вывода. Если функция возвращает значение, оно возвращается вызывающей операции. Если функция изменяет текущую область вывода (оператором *output-to*), это изменение распространяется на текущую среду.

Функция, возвращающая значение, определяется так:

```
define integer function add
    (value integer x,
        value integer y)
    as
    return x + y
```

Тип значения, возвращаемого функцией, указывается сразу после ключевого слова *define*. Возвращаемое значение может иметь любой допустимый тип переменной OmniMark или компонента OMX. Значение возвращается при помощи оператора *return,* который также определяет точку выхода из функции.

Так вы можете вызвать функцию *add*:

```
Process
   output "d" format add(2,3)
```

Функции могут генерировать вывод. Следующая функция выводит сгенерированное ею значение, прежде чем вернуть значение вызывающей операции. Заметьте: у функции нет описания типа возвращаемого значения; оператор *return* не требуется.

```
define function output-sum
   (value integer x,
      value integer y
   ) as
   output "d" format (x + y)
```

Эта функция может быть вызвана, как обычный оператор OmniMark:

```
Process
   output-sum(2, 3)
```

Вы можете создавать функции, одновременно возвращающие значение и осуществляющие вывод.

```
define integer function add
   (value integer x.
      value integer y
   } as
   output "Я буду складывать %d(x) А %d(y)%n"
   return x + y
process
   local integer z
   set z to add(2,3)
   output "%d(z)%n"
```

Однако я бы рекомендовал избегать написания функций, одновременно возвращающих значение и осуществляющих вывод. Дело не только в том, что это усложняет ваш код, но и в том, что это может привести к непредвиденным результатам. Так, если возвращаемое значение направляется в текущий вывод, вы рискуете не получить ни возвращаемое значение, ни вывод в ожидаемом виде.

Кроме того, вы можете создавать функции, не возвращающие значений и не осуществляющие вывода. Следующая функция сбрасыва-

ет значения всех переменных-переключателей на полке *switch*, передаваемой ей в качестве изменяемого *(modifiable)* аргумента:

```
define function clear-flags
    (modifiable switch the-flags
    ) as
    repeat over the-flags
            set tne-flags to false
    again
```

Все действия, которые вы можете выполнять в правиле, вызывающем функцию, допустимы и в самой функции. Скажем, функция, вызванная из правила разметки, может использовать оператор продолжения синтаксического анализа *%c*. Самое частое применение этого — возможность написания функции, обрабатывающей класс элементов разметки.

Текущие области ввода или вывода используются кодом внутри функции так же, как и в вызвавшем ее правиле (или функции).

Генерация и обработка исключений

Генерация и обработка исключений служат для управления выполнением OmniMark-программ. Вероятно, проще всего представить перехват и порождение исключений как механизм обработки исключительных ситуаций. Что же такое исключительная ситуация? В принципе это любое прерывание нормального хода выполнения программы. Некоторые исключительные ситуации возникают независимо от вашего желания при выполнении программы или при попытке установить связь с *внешними* объектами. Исключительные ситуации применяются и для решения других задач.

Простой фрагмент кода обычно обрабатывает 90% всех исключений. Оставшиеся 10% требуют существенно другой обработки. Разделение ситуаций на обычные и исключительные — удобный метод упрощения и упорядочения кода. Такой подход вовсе не подразумевает, что исключения обязательно являются ошибками или возникают неожиданно. Часто исключительные ситуации — это именно то, в чем вы наиболее заинтересованы. Программирование с использованием исключений — это просто техника создания алгоритмов для решения специфических задач.

Вы можете использовать порождение и перехват исключений в программе с многоуровневой вложенностью для быстрого выхода с любого уровня вложенности. OmniMark не поддерживает оператор «goto» (к счастью), поэтому выход из вложенного цикла или из дру-

гой иерархической области может быть слишком сложным. Есть простое решение — создать перехват исключения и затем породить исключение где-нибудь во вложенной области. OmniMark достаточно умен, чтобы самостоятельно предпринять необходимые действия по выходу из цикла.

Перехват программных ошибок

Самое простое использование перехвата и порождения исключений позволяет программе восстанавливаться, когда что-либо происходит не так, как должно бы. Рассмотрим код простой серверной программы, содержащий цикл:

```
include "omioprot.xin"
include "omtcp.xin"
process
    local TCPService my-service
    set my-service to TCPServiceOpen at 5432
    repeat
        local TCPConnection my-connection
        local stream my-response
        set my-connection to TCPServiceAcceptConnection my-service
        open my-response as TCPConnectionGetOutput my-connection
        using output as my-response
        do
            submit TCPConnectionGetLine my-connection
        done
    again
```

Любая ошибка в этой программе или в соединении TCP приведет к завершению, так как обработчиков ошибок нет. Однако серверные программы лучше писать так, чтобы они работали в любой ситуации, поэтому мы изменим код, чтобы программа могла справиться с ошибками:

```
include "omioprot.xin"
include "omtcp.xin"
process
    local TCPService my-service
    set my-service to TCPServiceOpen at 5432
    repeat
        local TCPConnection my-connection
        local stream my-response
```

```
        set my-connection to TCPServiceAcceptConnection my-service
        open my-response as TCPConnectionGetOutput my-connection
        using output as my-response
        do
            submit TCPConnectionGetLine my-connection
        done
        catch ftp год ram-error
    again
```

Мы добавили только одну строку, но эта версия программы гораздо устойчивее. В случае возникновения любой ошибки выполнения правил *find*, вызванных оператором *submit* в цикле *repeat*, управление передается в строку *catch #program-error*. Попутно OmniMark очистит занятую память. Локальные области и захваченные ресурсы будут освобождены.

Правда, мы не предприняли попыток сохранить результаты работы, выполнявшейся в момент возникновения ошибки. Эта работа будет прервана, а все связанные с ней ресурсы освобождены. Но серверная программа по-прежнему останется в рабочем состоянии и будет готова принять следующий запрос.

Разумеется, мы бы хотели узнать, какая именно ошибка произошла и почему. Константа *ttprogram-error* делает эту информацию доступной, так что мы можем исправить ошибку или хотя бы проанализировать ее. Чтобы быть уверенными, что все ошибки фиксируются, перепишем программу:

```
include "builtins.xin"
include "omioprot.xin"
include "omtcp.xin"

process
    local TCPService my-service
    set my-service to TCPServiceOpen at 5432
    repeat
        local TCPConnection my-connection
        local stream   my-response
        set my-connection to TCPServiceAcceptConnection my-service
        open my-response as TCPConnectionGetOutput my-connection
        using output as my-response
        do
            submit TCPConnectionGetLine my-connection
```

```
        done
        catch #program-error code c message m location l
        log-message ( "Ошибка "
        || "d" format c
        || " "
        || m
        || " "
        || l
        || " время "
        || date "=Y/=M/=D =h:=m:=s"
                                                    )
    again
```

Заметьте: перехват исключения относится ко всему исполняемому коду. Код внутри блока перехвата исполняется, только если при выполнении программы произойдет ошибка.

Пока мы не увидели ничего относящегося к «порождающей» части порождения и перехвата исключений. Дело в том, что OmniMark сам переходит к группе *#program-error*. OmniMark может также переходить к обработке внешних исключений *#external-exception* при возникновении проблем соединения с внешними по отношению к программе объектами (таких, как невозможность открыть файл или установить соединение с компонентом OMX). Мы не беспокоились о перехвате внешних исключений *#external-exception* в предыдущем фрагменте, так как ошибка в программе, перехватываемая обработчиком внешних исключений, одновременно является обыкновенной программной ошибкой, с которой справляется оператор *catch #program-error*. В иных обстоятельствах мы могли бы применить *#external-exception* в явном виде. Предположим, одно из правил *find* в нашей серверной программе хочет, но не может открыть файл:

```
find "open file " letter+ => fowl
    output file fowl
    catch #external-exception
    output "Не могу открыть файл " || fowl || "."
```

Если попытка открыть файл окажется неудачной, возникает внешнее исключение в первой операции *output*. Мы перехватываем исключение, и программа продолжает выполнение, будто ничего не случилось.

Теперь наша серверная программа более устойчива: ошибка открытия файла не будет прерывать обработку текущего запроса. Вмес-

то этого клиент получит сообщение об ошибке, которое мы выводим наряду с любой другой информацией, генерируемой запросом.

Обработка исключений

Когда дело доходит до обработки исключений, OmniMark предоставляет вам полную свободу действий. Вы можете создавать исключения, определять точки перехода и блоки перехвата исключений. Следующая программа использует специальное исключение для дистанционного завершения работы сервера:

```
include "builtins.xin"
include "omioprot.xin"
include "omtcp.xin"
declare eaten nap-time
process
   local TCPService my-service
   set my-service to TCPServiceOpen at 5432
   repeat
         local TCPConnection my-connection
         local stream   my-response
         set my-connection to TCPServiceAcceptConnection my-service
         open my-response as TCPConnectionGetOutput my-connection
         using output as my-response
         do
            submit TCPConnectionGetLine my-connection
         done
         catch  ttprog'am-errcr
   agair
   catch nap-time
find "sleep"
   throw nap-time
```

В этом фрагменте описан классический случай исключения. Каждый запрос, адресованный нашему серверу, является запросом информации — каждый, кроме одного. Запрос *«sleep»* — это команда для сервера завершить работу. Это исключительный случай, и мы обрабатываем его с помощью исключения.

В данном исключительном случае, когда мы завершаем работу сервера, надо выйти из цикла установления соединения. Мы проделываем это оператором перехвата *catch*, размещенного вне цикла. Этот оператор имеет метку *nap-time*. Имена нужны операторам перехвата, чтобы мы могли задействовать несколько таких операторов и с раз-

ными целями. Как и любые другие имена в OmniMark-программах, имя оператора перехвата должно быть объявлено, что мы и сделали в начале программы. Мы помешаем оператор вне цикла обработки запроса. Так как OmniMark очищает память при выполнении перехода, все локальные области внутри цикла должным образом закрываются. Затем, поскольку мы уже вне цикла и в конце обработки правил, программа завершает работу, и сервер останавливается.

Сам по себе переход прост. При возникновении исключительной ситуации (запроса *sleep)* мы просто переходим к соответствующему блоку перехвата. OmniMark управляет всем остальным.

Передача дополнительной информации

При переходе к блоку *#program-error* или *#external-exception* OmniMark передает дополнительные данные в виде трех параметров: *code* (код, идентифицирующий исключение), *message* (сообщение об ошибке) и *location* (фрагмент кода, вызвавший ошибку). Вы также можете предать дополнительную информацию вашему обработчику исключений при помоши дополнительных параметров, используя объявление функции. Следующая программа добавляет возможность протоколирования причины выключения сервера:

```
include "builtins.xin"
include "omioprot.xin"
include "omtcp.xin"
declare catch nap-time because value stream reason

process
   local TCPService my-service
   set my-service to TCPServiceOpen at 5432
   repeat
         local TCPConnection my-connection
         local stream   my-response
         set my-connection to TCPServiceAcceptConnection my-service
         open my-response as TCPConnectionGetOutput my-connection
         using output as my-response
         do
            submit TCPConnectionGetLine my-connection
         done
         catch #program-error
   again
   catch nap-time because r
```

```
        log-message ( "Выключить по причине "
                      || r
                      || " Время: "
                      || дата "=Y/=M/=D =h:=m:=s"
                                                )
find "sleep" white-space* any* => the-reason
   throw nap-time because the-reason
```

Правило *find* получает оставшуюся часть сообщения *«sleep»* и использует ее как параметр при переходе. Перехватчик исключений получает эти данные и использует их для создания соответствующей записи в протоколе.

Очистка памяти после завершения работы

Я говорил, что OmniMark очищает память при завершении работы. Это значит, что он очищает все, о чем знает, но это не относится к результатам вашей работы. С другой стороны, вам, возможно, нужно обязательно выполнить некие инструкции, даже если произойдет исключение. Для этого служит ключевое слово OmniMark *always*. Допустим, наш сервер ведет собственный протокол соединения и использует возможности OmniMark для перехвата ошибок. Мы хотели бы, чтобы файл протокола соединения закрывался между запросами, чтобы облегчить запись в файлы. Поэтому мы открываем и закрываем файлы для каждого соединения:

```
declare catch nap-time because value stream reason
include "builtins.xin"
include "omioprot.xin"
include "omtcp.xin"
global stream log-file-name initial {"nap-time.log"}
process
   local TCPService my-service
   local stream my-log
   set my-service to TCPServiceOpen at 5432
   repeat
         local TCPConnection my-connection
         local stream   my-response
         reopen my-log as file log-file-name
         set my-connection to TCPServiceAcceptConnection my-service
         put my-log TCPConnectionGetPeerIP my-connection
                                  || date "=Y/=M/=D =h:=m:=s"
```

```
        open my-response as TCPConnectionGetOutput my-connection
        using output as my-response
        do
            submit TCPConnectionGetLine my-connection
        done
        close my-log
        catch ttprogram-error code c message m location 1
            log-message ( "Ошибка "
            || I "d" format c
            || " "
            || m
            || " в "
            || l
            || " время "
            || date "=Y/=M/=D =h:=m:=s"
                                                    )
again
catch nap--_ime because reason
        log-message ( "Выключить го причине "
                    || reason
                    || " Время: "
                    || date "=Y/=M/=D =h:=m:=s"
                                                )
```

В этом фрагменте ошибка при обработке запроса вызовет переход к блоку *#program-error*.Это может означать, что строка *close my-log* не была выполнена и файл протокола остался открытым. (В этом случае OmniMark не может очистить память, так как поток *my-log* принадлежит к более широкой области, которая не была закрыта.) Мы хотим, чтобы строка *close my-log* выполнялась всегда, независимо от наличия или отсутствия ошибки. Чтобы быть уверенными в этом, используем инструкцию *always:*

```
declare catch nap-time because value stream reason
include "builtins.xin"
include "omioprot.xin"
include "omtcp.xin"
global stream log-file-name iritial {"nap-time.log"}
process
   local TCPService my-service
   local stream my-log
```

```
set my-service to TCPServiceOpen at 5432
repeat
        local TCPConnection my-connection
        local stream   my-response
        reopen my-log as file log-file-name
        set my-connection to TCPServiceAcceptConnection my-service
        put my-log TCPConnectionGetPeerIP my-connection
                                || date "=Y/=M/=D =h:=m:=s"
        open my-response as TCPConnectionGetOutput my-connection
        using output as my-response
        do
           submit TCPConnectionGetLine my-connection
        done
        catch #program-error code c message m location l
           log-message ( "Ошибка "

           || "d" format c
           || " "
           || m
           || " в "
           || l
           || " время "
           || date "=Y/=M/=D =h:=m:=s"
                                                        )
        always
           close my-log
again
catch nap-time because reason
        log-message ( "Выключить по причине "
                      || reason
                      || " Время: "
                      || date "=Y/=M/=D =h:=m:=s"
                                                  )
find "sleep" white-space* any* => the-reason
   throw nap-time because the-reason
```

В момент перехода OmniMark закрывает области одну за другой, пока не обнаружит ту, что содержит блок перехвата для этого исключения. Поэтому OmniMark выполняет любой код в блоке *always* для каждой такой области, включая ту, что содержит блок перехвата. В

этом примере строка *close* будет выполнена перед выполнением блока *catch* независимо от того, где или почему произошла ошибка.

Программирование с применением исключений — мощная техника, позволяющая сделать программы более надежными и простыми для чтения и написания. Взгляните на полный текст серверной программы со всеми средствами обработки и протоколирования исключений (но без правил *find*, выполняющих оставшуюся часть работы):

```
declare catch nap-time because value stream reason
include "builtins.xin"
include "omioprot.xin"
include "omtcp.xin"
global stream log-file-name initial {"nap-time.log"}
process
   local TCPService my-service
   local stream my-log
   set my-service to TCPServiceOpen at 5432
   repeat
         local TCPConnection my-connection
         local stream   my-response
         reopen my-log as file log-file-name
         set my-connection to TCPServiceAcceptConnection my-service
         out my-log TCPConnectionGetPeerIP my-connection
                          || date "=Y/=M/=D =h:=m:=s"
         open my-response as TCPConnectionGetOutput my-connection
         using output as my-response
         do
            submit TCPConnectionGetLine my-connection
         done
         catch #program-error code c message m location l
            log-message ( "Ошибка "
                        || "d" format c
                        || " "
                        || m
                        || " в "
                        || l
                        || " время "
                        || date "=Y/=M/=D =h:=m:=s"
                                                  )
```

```
        always
           close my-log
   again
   catch пар-time because reason
        log-message ( "Выключить по причине "
                      || reason
                      || " Время: "
                      | I date "=Y/=M/=D =h:=m:=s"
                                          )
find "sleep" white-space* any- => the-reason
   throw пар-time because the-reason
find "open file " letter+ => foo
   output file foo
   catch #external-exception identity i message m location 1
   output "Не могу открыть файл " || foo || "."
   log-message ( "Ошибка "
                 |I i
                 || " "
                 || m
                 || " в "
                 || 1
                 || " время "
                 || date "=Y/=M/=D =h:=m:=s"
                          )
```

Группы правил

По умолчанию все правила OmniMark активны в течение выполнения программы. Вы можете изменить это, объединив правила в группы:

```
group duck
   find "подходит"
        output "#walks#"
   find"вразвалку"
        output "#waddles#"
group bar
   find "бармен"
        output "#bartender#"
   find "пиво"
        output "#beer#"
```

```
group  ttimplied
process-start
   using group duck
   do
         submit "Заходит гусь в бар."
   done
   jsing group bar
   do
         submit "Она говорит бармену, 'плесни-ка пива'."
   Done
```

В этой программе только правила, содержащиеся в группе *duck*, используются для обработки строки *«Заходит гусь в бар.»*, а для обработки строки *«Она говорит бармену, 'плесни-ка пива'»* — только правила группы *bar*.

Почему группа *group #implied* расположена перед оператором *process-start?* Правило *process-start* — такое же, как и все остальные, поэтому включается в группы так же, как и другие. Группа *group #implied* замещает группу по умолчанию. (В программе без явно указанных групп все правила относятся к группе по умолчанию.) Только заданная по умолчанию группа активна при запуске программы. Поэтому в заданной по умолчанию группе должно быть хотя бы одно правило, чтобы активизировать другие группы правил. Если мы не поместим правило запуска процесса обработки в группу по умолчанию, ни одно правило в программе никогда не будет активизировано.

Любое правило, встречающееся в программе перед первым оператором группы, автоматически относится к группе по умолчанию. Однако, используя группы правил, вы должны явно поместить глобальные правила в группу *group #implied.* (Подумайте, что произошло бы, если бы вы подключили файл, содержащий операторы групп в начале вашей программы и не поместили бы в явном виде глобальные правила в группу *group #implied.*)

Все правила в группе по умолчанию являются глобальными. Вы не можете отключить эту группу, поэтому правила, включенные в нее, всегда активны. Исходя из этого, имеет смысл свести количество правил в группе по умолчанию к минимуму. (Но помните, что вы должны поместить в нее хотя бы одно правило.)

Можете ли вы иметь одновременно более чем одну активную группу правил? Конечно.

```
using group duck & bar & dick & harry
```

Кроме того, вы можете добавить группу к текущему набору активных групп, используя оператор *ttgroup* для представления всех текущих активных групп.

```
using group duck & bar & #group
```

Константы и макросы

Вы можете определять макросы, используя ключевые слова *macro* и *macro-end*. Макросы полезны для:

- определение констант;
- определение комплексных шаблонов;
- локализация кода.

Следующий макрос определяет константу:

```
macro pi is
   3.14159265358979323846264
macro-end
```

Вот пример использования этого макроса:

```
include "ombcd.xin"
macro pi is
 3.1415926535897932
macro-end
process
   local bcd area
   local bcd r initial {5.9}
   set area to pi - r * r
   output "d" % area
```

Следующий макрос определяет шаблон десятичного числа:

```
macro decimal-number is
   ("-"? digit+ (","  digit+)?)
macro-end
```

Вы можете использовать этот макрос в правиле *find*:

```
find "[" decimal-number => number "]"
```

ПРИМЕЧАНИЕ Разумно заключать любой шаблон, определяемый в макросе, в круглые скобки. Это гарантирует, что значение шаблона присваивается переменной шаблона целиком (если в шаблоне использованы переменные).

Следующий макрос локализует блок кода — в данном случае блок перехвата исключения. Его нельзя локализовать в функции, поскольку функция создала бы новую область, а блок перехвата исключения должен работать в той области, где вызван.

```
macro catch-and-report-program-errors is
    catch tfprogram-error
          code c
          message m
          location l
    log-message ( "Ошибка "
                  || "d" % c
                  || "%n"
                  || m || "%n"
                  || l || "%n"
                                )
macro-end
```

Вы можете применить его в том месте программы, где надо разместить перехватчик исключений.

```
include "ombcd.xin"
include "builtins.xin"
macro pi is
 3.1415926535897932
macro-end
macro catch-and-report-program-errors is
    catch #program-error
          code c
          message m
          location l
    log-message ( "Ошибка "
                  || "d" % c
                  || "%n"
                  || m || "%n"
                  || l || "%n"
                                )
macro-end
process
    local bcd area
    local bcd r initial {"5.9"}
    set area to pi * r * r
```

```
output "d" % area
catch-and-report-program-errors
```

Финансовые вычисления

Финансовые вычисления выполняются в OmniMark с помощью типа данных BCD. Он служит для представления денежных значений и дробей, таких как налоговые ставки. В отличие от чисел с плавающей точкой, данные типа BCD обеспечивают точное значение дробей для финансовых расчетов.

Ниже приведен пример простых финансовых расчетов с использованием числовых переменных типа BCD. Заметьте: ключевое слово *bcd* должно предшествовать имени переменной этого типа.

```
include "ombcd.xin"
process
    local bcd price
    local bcd tax-rate
    local integer quantity
    local bcd total
    set price to 19.95
    set tax-rate to 0.07
    set quantity to 28
    set total to price * quantity * (1 + tax-rate)
    output "<$,NNZ.ZZ>" format total
```

Форматирующая строка *«<$,NNZ.ZZ>»* использует язык шаблонов BCD для создания выходной строки с символом $ в начале, запятой в качестве разделителя групп разрядов и двумя знаками после десятичной точки.

Следующая CGI-программа читает десятичное число из формы HTML и присваивает его значение переменной типа BCD:

```
include "ombcd.xin"
include "omcgi.xin"
declare #process-input has unbuffered
declare flprocess-output has binary-mode
process
    local stream form-data variable
    local bcd price
    local bcd tax-rate
    local integer quantity
```

```
local bcd total
CGIGetQuery into form-data
set price to form-data{"price"}
set quantity to form-data{"qty"}
set tax-rate to   0.07
set total to price * quantity - (1 + tax-rate)
output "Content-type: text/html"
       || crlf
       |  crlf
       || "<html><body>"
       || "<H1>Подтверждение заказа</H1>"
       || "<p>Всего заказано на сумму: "
       || "<$,NNZ.ZZ>" format total
```

Следующий документ HTML может быть использован для тестирования предыдущей CGI-программы:

```
<HTML>
      <HEAD>
            <TITLE>Размещение заказа</TITLE>
      </HEAD>
      <BODY>
            <H1>Размещение за<аза</H1>
            <FORM METHOD="POST" ACTION="bcd.xom">
                        Цена: <INPUT TYPE="TEXT" NAME="price"></
INPUT><BR>
                        Количество: <INPUT TYPE="TEXT" NAME="qty"></
INPUT><BR>

<INPUT TYPE="SUBMIT" VALUE="Рассчитать сумму с учетом налогов">
            </FORM>
      </BODY>
</HTML>
```

Сетевое программирование

Сетевое программирование относится к тем операциям, что происходят «за сценой» и включают взаимодействие приложений, работающих на разных компьютерах сети. Хотя пользователь никогда не видит этих программ, они выполняют реальную работу в Интернете и других сетевых средах.

Сервер — «рабочая лошадка» сетевого программирования, предоставляющая услуги другим сетевым программам. Сервер обычно ожидает запроса от других программ, выполняет его и отправляет ответ. Поскольку он работает непрерывно в течение многих дней и недель, сервер обязан быть устойчивым.

Программа, которая обращается к услугам сервера, называется клиентом. Клиент отправляет запрос и ждет ответа от сервера.

Приложения промежуточного уровня играют роль посредника между двумя программами. Обычно такие программы выполняют транзакции. Сервер, получающий запрос от Web-сервера и запрашивающий БД для заполнения полей запроса. — типичный пример приложения промежуточного уровня. Он управляет соединением Web-сервера и сервера БД и создает выходные данные, добавляя разметку HTML к результатам запроса к БД.

Приложение промежуточного уровня действует и как клиент, и как сервер. Такие приложения — непременный атрибут многоуровневой архитектуры. OmniMark позволяет создавать серверные программы и приложения промежуточного уровня, используя соответствующие библиотеки.

Для разработки проектов, связанных с сетевым программированием, надо понимать сетевые протоколы. Протокол — это соглашение о связи между клиентом и сервером. Используя общедоступный протокол или предлагая для общего использования собственный, вы можете предоставить доступ к своему серверу любому числу клиентов. С другой стороны, засекретив протокол (или даже применяя шифрование), можно защитить свой сервер от несанкционированного доступа.

Протоколы бывают транспортные и прикладные. Транспортные служат для передачи сообщений по сети от одного компьютера к другому. TCP/IP, транспортный протокол Интернета, поддерживается сетевыми библиотеками OmniMark и компонентами OMX TCPService и TCPConnection.

Прикладным протоколом называют соглашение, описывающее, что представляет собой сообщение и что оно означает. В то время как дисковые файлы содержат маркеры конца файла, сетевое сообщение - это просто поток байтов, пересылаемых через открытое соединение. **Вы** должны сами анализировать данные, чтобы определить целостность сообщения. Библиотека протокола ввода-вывода OmniMark поддерживает все распространенные методы определения границ сообщения.

Получив законченное сообщение, вы должны декодировать его, *а* затем создать и отправить соответствующий ответ. OmniMark — идеальный язык для декодирования сообщений сетевых протоколов. Его потоковые особенности позволяют легко интерпретировать сообщение и быстро создавать ответ.

Простое приложение-сервер

Следующая программа — это простое приложение-сервер, которое возвращает первую строку детского стихотворения, получая сообщение, содержащее имя главного персонажа этого стихотворения.

```
declare catch server-die
include "omioprot.xin"
include "omtcp.xin"
include "builtins.xin"
define switch function TCPservice-is-working
        read-only TCPService trie-service
        as
do when TCPServiceIsInError trie-service
        local stream errorReport variable initial-size 0
        TCPServiceGetStatusReport the-service into errorReport
        log-message "TCPService Error:"
        repeat over errorReport
            log-message errorReport
        again
        return false
  else
        return true
  done
process
  local TCPService my-service
  set my-service to TCPServiceOpen at 5432
  throw server-die unless TCPService-is-working my-service
  repeat
        local TCPConnection my-connection
        local stream my-response
        set my-connection to TCPServiceAcceptConnection my-service
        throw server-die unless TCPService-is-working my-service
        open my-response as TCPConnectionGetOutput my-connection
        using output as my-response
            submit TCPConnectionGetLine ny-connection
```

```
        catch #program-error
    again
    catch server-die
find "Мэри%n"
    output "У Мэри был ягненок%13#%10#"
find "Том%n"
    output "Том. Том, волынщика сын.%13#%10#"
find "Джек" | "Джилл" "%n"
    output "Идут на горку Джек и Джилл.%13#%10#"
find "умереть%n"
    output "Ах! Шлеп!%13#%10#"
    throw server-die
```

Работу сервера можно сравнить с работой телефона. Сначала мы подключаем его, назначая телефонный номер. Затем он должен ждать вызова. Дождавшись, сервер отвечает на вызов, выслушивает сообщение и соответствующим образом отвечает. Сеанс связи может состоять из обмена одним или несколькими сообщениями. Когда сеанс связи завершается, сервер «вешает трубку» и переходит в режим ожидания следующего вызова.

Работу сервера можно разбить на три фазы:

- **запуск** — начало работы;
- **цикл запроса** — ожидание вызова, ответ, повторение цикла;
- **останов** — завершение работы.

Поскольку сервер работает в течение длительного времени и обрабатывает много запросов, к нему предъявляются два основных требования.

- Независимо от того, что происходит при выполнении запроса, сервер не должен завершаться аварийно — он должен продолжать работу.
- По завершении выполнения запроса (независимо от того, что при этом происходит) сервер должен переходить в состояние готовности к приему. (Если сервер находится в различных состояниях при приеме разных запросов, его ответы не могут быть признаны достоверными.)

Посмотрим, соответствует ли этим требованиям сервер из нашего примера.

```
process
    local TCPService my-service
    set my-service to TCPServiceOpen at 5432
```

Этот фрагмент описывает начало работы сервера. Компонент OMX TCPService служит для инициирования обслуживания порта 5432 компьютера, на котором запушен сервер. Адресом сервера будет сетевой адрес компьютера в сочетании с номером порта. Использование разных портов позволяет нескольким серверам работать одновременно на одном компьютере.

```
repeat
   local TCPConnection my-connection
   ...
   set my-connection to TCPServiceAcceptConnection my-service
again
```

Этот код обрабатывает входящие вызовы. Компонент OMX *TCP-ServiceAcceptConnection* ожидает соединения с клиентом. Устанавливая соединение, сервер возвращает переменную компонента OMX TCPConnection, которая содержит описание соединения с клиентом. Переменная *my-connection* типа *TCPConnection* объявлена внутри цикла *repeat,* чтобы при выходе из цикла на завершающей стадии обеспечить автоматическое закрытие и очистку соединения.

```
repeat
   local TCPConnection my-connection
   ...
   set my-connection to TCPServiceAcceptConnection my-service
   ...
   submit TCPConnectionGetLine my-Conn
again
```

Компонент OMX *TCPConnection* реализует источник данных, чтобы программа могла получать информацию от клиента. Чтение данных из сетевого соединения отличается от чтения данных из файла. В отличие от работы с файлом, когда вы можете либо читать или записывать данные, сетевое соединение является дуплексным. OmniMark не может выявить коней сообщения, переданного через сетевое соединение, как в случае с файлом. Пока соединение открыто, всегда есть возможность поступления новых данных. Поэтому для передачи данных через сеть требуется специальный протокол прикладного уровня для определения конца сообщения. OmniMark поддерживает все распространенные прикладные протоколы, предназначенные для этих целей, с помощью библиотеки протоколов ввода-вывода. Библиотека TCP использует библиотеку протоколов ввода-вывода для поддержки протоколов Интернета. В нашем примере применяется

строчный (line-based) протокол. В запросе для обозначения конца сообщения используется комбинация символов «коней строки», «возврат каретки» (ASCII 13, 10). Функция *TCPConnectionGetLine* читает данные из источника и закрывает его, когда обнаруживается такое сочетание символов. Мы пересылаем данные из источника нашим правилам *find*, которые генерируют соответствующий ответ.

```
repeat
   ...
   local stream my-response
   ...
   open my-response as TCPConnectionGetOutput my-connection
   using output as my-response
         submit TCPConnectionGetLine my-connection
   ...
again
```

Элемент OMX *TCPConnection* представляет собой дуплексное сетевое соединение. Мы должны использовать его не только как источник для чтения данных, но и направить в него выходной поток, чтобы пересылать данные клиенту. Это делает функция *TCPConnectionGetOutput*.

Как только мы присоединяем к нашему оператору *submit* оператор *using output as my-response*, наши правила *find* получают возможность читать данные из сетевого соединения и записывать их в него.

```
find "Мэри%n"
   output "У Мэри был ягненок %13#%10#"
find "Том%n"
   output "Том, Том, волынщика сын.%13#%10#"
```

Мы используем строчный протокол, но символ конца строки на разных платформах кодируется разными символами ASCII (13, 10 для Windows, 10 для UNIX). Конструкция *%n* OmniMark *%n* является стандартным обозначением конца строки и удовлетворяет обоим требованиям. Если вы выводите в выходной поток символы *%n*, OmniMark выведет комбинацию, соответствующую платформе, на которой выполняется программа. Для передачи данных через сеть, объединяющую разные платформы, мы выбираем соответствующую комбинацию самостоятельно. Наш протокол требует комбинации символов 13, 10. Мы же используем *%n*, так что даже если клиент забудет отправить последовательность, означающую конец строки, мы по-прежнему сможем прочитать сообщение. Но при передаче данных надо

отправлять именно последовательность *«%13#%10#»*, *а* не *«%n»*. Это важный принцип сетевого программирования: «Относитесь либерально к входным данным и консервативно — к исходящим».

Это правило *find* ищет во входном потоке команду завершения работы сервера:

```
find "умереть"
   output "Ах! Шлеп!%13#%10#"
   throw server-die
```

Чтобы быть уверенными в корректном завершении работы, мы обеспечиваем возможность остановки сервера, посылая ему сообщение об окончании работы.

Завершение работы является исключением по отношению к нормальным операциям сервера. Мы инициируем его обработку путем перехода к блоку перехвата исключения под названием *server-die*.

```
Process
   ...
   repeat
         ...
   again
   catch server-die
```

Обработчик исключений *server-die* выполняется по завершении серверного цикла. OmniMark попутно очищает локальные области, обеспечивая корректное и организованное завершение работы. Поскольку мы находимся в конце процесса обработки правил, программа завершается нормально.

Ошибки и восстановление

Сервер должен работать, несмотря на любые ошибки, возможные при обработке запроса. Однако программа должны быть остановлена, если не может нормально выполнять свои функции. Решение этих задач обеспечивает код:

```
define switch function TCPservice-is-working
         read-only TCPService the-service
         as
   do when TCPServiceIsInError the-service
         local stream errorReport variable initial-size 0
         TCPServiceGetStatusReport the-service into errorReport
         log-message "TCPService Error:"
```

```
            repeat over errorReport
                log-message errorReport
            again
            return false
    else
            return  true
    done
process
    set my-service to TCPServiceOpen at 5432
    throw server-die unless TCPService-is-working my-service
    repeat
            ...
            set my-connection to TCPServiceAcceptConnection my-service
            throw server-die unless TCPService-is-working my-service
            ...
            catch #program-error
    again
```

При возникновении ошибки в процессе обработки запроса OmniMark инициирует событие *#program-error*. Мы перехватываем обработку события в конце серверного цикла, чтобы автоматически очистить все ресурсы, использованные в процессе обработки текущего запроса, и возвратить сервер в устойчивое состояние готовности. (В этом примере мы не делали попыток сохранить результаты выполнения запроса, в котором встретилась ошибка. Вы можете создать подобную обработку ошибок в рабочей программе, однако убедитесь, что вы предусмотрели переход в аварийный режим, который прерывает обработку текущего запроса и возвращает сервер в устойчивое состояние.)

В том маловероятном случае, когда происходит сбой в работе компонента TCPService, вы не сможете сделать ничего, кроме остановки сервера. Текущая версия библиотеки TCP не поддерживает перехват и обработку ошибок, поэтому полностью тестируйте программу каждый раз, когда используете эту библиотеку. При обнаружении ошибки просто запротоколируйте ее, а затем переходите к обработчику *server-die* для останова сервера.

Эта простая программа-сервер содержит все, что надо для создания устойчивого и работоспособного промышленного сервера. Вам надо адаптировать ее код под конкретный протокол, но, кроме этого, все остальное — это чистый код OmniMark, связывающий ввод и вывод с соединением.

Простая программа-клиент

Любая клиентская программа, написанная на любом языке, может работать с нашим сервером, если поддерживает нужный протокол. Вот программа-клиент, написанная на OmniMark:

```
Process
   local TCPConnection my-connection
   local stream my-request
   set my-connection to TCPConnectionOpen
                                        on "localhost" at 5432
   open my-request as TCPConnectionGetOutput my-connection
   using output as my-request
         output #args[1] | "%13#%10#"
   close my-request
repeat
   output TCPConnectionGetCharacters my-Conn
   exit unless TCPConnectionIsConnected my-Conn
again
```

Этот клиент принимает в качестве параметра командной строки имя персонажа детского стихотворения и выводит строку, полученную от сервера. Рассмотрим его подробнее.

```
set my-connection to TCPConnectionOpen
                                  on "localhost" at 5432
```

Как и сервер, для создания соединения клиент использует OMX-компонент TCPConnection. В отличие от сервера клиенту не нужен компонент TCPService, так как клиент не запускает службу, а лишь устанавливает соединение с внешней службой. Однако клиент играет более активную роль, чем сервер. Сервер ожидает вызова — клиент должен взять инициативу на себя и организовать вызов. Для этого предназначена функция *TCPConnectionOpen:* она получает сетевой адрес и номер порта сервера и, когда соединение установлено, возвращает объект OMX *TCPConnection,* который мы можем читать и записывать так же, как и в серверной программе.

```
repeat
   output TCPConnectionGetCharacters my-Conn
   exit when TCPConnectionIsInError my-Conn
again
```

Чтение данных, полученных от сервера, можно осуществить двумя путями. Поскольку наш протокол является строчным, для чтения ответа можем вызвать функцию *TCPConnectionGetLine*. Но мы знаем и то, что, закончив передавать данные, сервер разорвет соединение. (Это особенность нашего протокола.) Так что можно выбрать вариант чтения данных до разрыва соединения. Это позволит получить минимально возможное количество неполных данных, даже если что-то пойдет не так и сервер не будет посылать символов-разделителей конца строки. Вспомните: «Относитесь либерально к входным данным и консервативно — к исходящим».

Клиенты для распространенных серверов

Возможно, большинство из написанных вами на OmniMark клиентских программ будут предназначены для работы с наиболее распространенными серверами HTTP, FTP или SMTP/POP. Библиотеки соединений OmniMark поддерживают эти и другие протоколы, значительно упрощая задачу получения данных от этих серверов.

Программирование CGI

Программы на OmniMark запускаются из командной строки так (здесь Program.xom — имя запускаемой программы):

```
Omnimark -sb Program.xom
```

Сравните его с интерфейсом командной строки Perl. Для запуска программы на Perl вы используете следующий синтаксис (program.pl — имя запускаемой программы):

```
perl program.pl
```

OmniMark C/VM (компилятор/виртуальная машина) — аналог интерпретатора Perl или Python.

Вы можете передать OmniMark C/VM файл параметров или файл проекта, созданный при помощи интегрированной среды разработки. Я расскажу о файлах параметров OmniMark позже в этом приложении.

Псевдокомментарий

Псевдокомментарий #! позволяет определить в исходном коде командную строку для его выполнения, Эта нотация используется обычно в UNIX. Microsoft Internet Information Services (IIS), например, ее не применяет. IIS выполняет программы CGI, используя ассоциирование интерпретатора с расширениями имени файла. Впрочем, Web-

сервер Apache для Windows-платформ тоже применяет эту нотацию для запуска программ, работающих как программы CGI.

Данная нотация нужна лля выполнения сценариев, работающих под управлением Web-сервера Apache на любой платформе. Приступая к обработке сценария, Apache читает первую строку программы. Если первые два символа в этой строке — #!, Web-сервер использует остаток символов в первой строке (все, что следует за #!) в качестве командной строки для запуска этого сценария, размещая имя сценария в конце получившейся командной строки.

Например, для программы Hello.xom:

```
#!C:\OmniMark\omnimark.exe -sb
process
   output "Content-Type: text/html%13#%10#%13#%10#"
          || "Привет, мир OmniMark!"
```

Web-сервер выполнит командную **строку:**

```
C:\OmniMark\omnimark.exe -sb Hello.xom
```

Для аналогичной программы на Perl Hello.pl :

```
#!C:\Perl\bin\Perl.exe
print "Content-Type: text/html\r\n\r\n";
print "Привет, мир Perl!";
```

Web-сервер выполнит командную строку:

```
C:\Perl\bin\Perl.exe Hello.pl
```

Во многих языках сценариев, используемых на UNIX-платформах (например, в Perl), символ # обозначает комментарий. Следовательно, строка, начинающаяся с символов #!, будет проигнорирована при выполнении программы. OmniMark не исключение. Хотя комментарий в OmniMark обозначает символ «точка с запятой» (;), использование #! тоже допускается (правда, только в первой строке программы).

В ОС семейства UNIX программы, использующие эту нотацию, должны иметь установленный бит «исполняемый» (executable), чтобы сообщить Web-серверу, что файл является исполняемым — иначе Web-сервер вернет сообщение об ошибке при попытке выполнить программу CGI. Этот бит устанавливается командой *chmod*. Так, чтобы разрешить Web-серверу, работающему под UNIX, выполнение описанной выше программы Hello.xom, надо объявить программу исполняемой, выполнив команду:

```
chmod a+x Hello.xom
```

Конфигурирование Web-серверов для выполнения программ CGI, написанных на OmniMark

Этот раздел описывает конфигурирование Web-серверов I1S и Apache для запуска CGI-программ, написанных на OmniMark.

ПРИМЕЧАНИЕ Для выполнения настройки необходимы полномочия администратора.

Internet Information Services

Чтобы сконфигурировать IIS, работающий под Microsoft Windows 2000, для запуска CGI-программ на OmniMark выполните следующие действия.

1. Зарегистрируйтесь с правами администратора.
2. Запустите консоль управления (Microsoft Management Console), щелкнув правой кнопкой значок «Мой компьютер» и выбрав пункт меню «Управление».
3. Раскройте пункты «Службы и приложения» и «Информационные службы Интернет» в окне консоли «Структура». Щелкните правой кнопкой Web-сайт, для которого вы хотите создать каталог CGI, и выберите в меню «Новый» команду «Виртуальный каталог». Запустится мастер установки виртуальных каталогов. Создайте новый виртуальный каталог и убедитесь, что установлен флажок «Исполняемый». Этот каталог будет служить для хранения исходных файлов OmniMark.
4. Щелкните правой кнопкой новый каталог CGI и выберите в контекстном меню команду «Свойства». Откроется диалоговое окно свойств.
5. Откройте вкладку «Виртуальный каталог» и щелкните кнопку «Конфигурация». Откроется диалоговое окно «Конфигурация приложения».
6. Откройте вкладку «Выбор программы» и щелкните кнопку «Добавить». Откроется диалоговое окно «Добавить/изменить расширение».
7. В поле «Программа» введите командную строку для запуска OmniMark C/VM, подставив *%s* вместо имени вашего сценария. Обычно это выглядит так:

```
C:\OmniMark\omnimark.exe -noexpand -sb %s
```

8. Введите *.xom* в поле «Расширение».
9. Убедитесь, что установлен флажок «Запуск сценариев», и щелкните OK.

10. Повторите пп. 7–9 для расширения .хор, используя командную строку:

```
C:\OmniMark\omnimark.exe -noexpand -f %s
```

И. Повторите пп. 7-9 для расширения .xar, используя командную строку:

```
C:\OmniMark\omnimark.exe -noexpand -f %s
```

12. Повторите пп. 7–9 для расширения .pro, используя командную строку:

```
C:\OmniMark\omnimark.exe -noexpand -s pmo.xom %s
```

13. Щелкните OK.
14. В диалоговом окне «Свойства» убедитесь, что установлены флажки запуска программ и выполнения сценариев, и щелкните OK.

Personal Web Server для Windows 95/98

Для конфигурирования сервера Microsoft Personal Web Server (PWS) под управлением Microsoft Windows 95/98 для запуска CGI-программ на OmniMark сделайте следующее.

1. На всякий случай сделайте копию реестра.
2. В меню «Пуск» выберите команду «Выполнить», чтобы открыть одноименное диалоговое окно. Наберите *regedit* и щелкните OK.
3. В Редакторе реестра откройте раздел HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\w3svc\parameters\ ScriptMap.
4. Из меню «Правка» выберите команду «Создать», «Строковый параметр». Будет создан новый строковый параметр, и вы сможете задать его имя.
5. Наберите *.xom* в качестве имени параметра.
6. Дважды щелкните новый параметр. Откроется окно «Изменение строкового параметра»,
7. В окошке «Значение» наберите командную строку для запуска файлов с расширением .xom:

```
C:\OmniMark\OmniMark.exe -noexpand -sb %s
```

8. Щелкните OK.
9. Повторите пп. 4–8 для расширения .хор, используя командную строку:

```
C:\OmniMark\OmniMark.exe -noexpand -f %s
```

10. Повторите пп. 4–8 для расширения .хаг, используя командную строку:

```
C:\OmniMark\OmniMark.exe -noexpand -f %s
```

11. Повторите пп. 4–8 для расширения .рто, используя командную строку:

```
C:\OmniMark\OmniMark.exe -noexpand -s omo.xom %s
```

12. Закройте Редактор реестра.

Убедитесь, что папки вашей конфигурации сервера позволяют запускать CGI-программ. Подробности см. в документации PWS.

Web-сервер Apache

Проше всего сконфигурировать запуск сценариев CGI в любой версии Apache, используя директиву ScriptAlias в конфигурационном файле сервера. Эта директива связывает виртуальный каталог с физическим и сообщает Web-серверу, что все файлы в этом каталоге могут выполняться как сценарии CGI.

Эта директива только создает связь между виртуальным и физическим каталогами; она не устанавливает связь между расширением файла и интерпретатором. Для формирования соответствующей командной строки Apache будет использовать псевдокомментарий в первой строке кода сценария.

Apache для Windows 2000

Добавьте в ваш конфигурационный файл Apache (он называется httpd.conf и находится в каталоге, куда вы установили Apache) строку:

```
ScriptAlias /virtdir/ "C:/www/root/cgi-bin/"
```

Здесь *virtdir* — имя виртуального каталога, который вы хотите создать, а *C:/www/root/cgi-bin/* — имя физического каталога, где находятся ваши сценарии CGI. Заметьте: в имени физического каталога используется символ «/».

Все наши примеры предполагают использование в качестве каталога *virtdir* каталога *cgi-bin*:

```
ScriptAlias /cgi-bin/ "C:/www/root/cgi-bin/"
```

Перезапустите Web-сервер командой «Пуск», «Настройка», «Панель управления», «Службы», чтобы новые параметры из конфигурационного файла вступили в силу.

Теперь запрос *http://hostname/cgi-bin/Hello.xom* приведет к выполнению файла C:\www\root\cgi-bin\Hello.xom.

Apache для UNIX/Linux

Для конфигурирования Apache для UNIX или Linux для запуска сценариев CGI на OmniMark добавьте в конфигурационный файл строку:

```
ScriptAlias /virtdir/ "/www/root/cgi-bin/"
```

Здесь *virtdir* — имя виртуального каталога, который вы хотите создать, а */www/root/cgi-bin/* — имя физического каталога, хранящего ваши сценарии CGI. Обратите внимание на символ «/» в имени физического каталога.

Все наши примеры предполагают использование в качестве */virtdir* каталога /www/root/*cgi-bin*:

```
ScriptAlias /cgi-bin/ "/www/root/cgi-bin/"
```

Чтобы изменения в конфигурационном файле вступили в силу, перезапустите ваш Web-сервер командой:

```
apachectl restart
```

Теперь запрос *http://hostname/cgi-bin/Hello.xom* приведет к запуску интерпретатора OmniMark командной строкой, заданной в первой строке файла Hello.xom с названием исходного файла в качестве параметра:

```
/usr/local/bin/omnimark -sb /www/root/cgi-bin/Hello.xom
```

Местонахождение системных библиотек OmniMark

OmniMark использует переменные среды для поиска системных библиотек и подключаемых файлов. На Windows-компьютерах программа установки OmniMark устанавливает значение переменных среды. Для UNIX-компьютеров это нужно сделать самостоятельно. Если, скажем, вы установили OmniMark в каталог /usr/local/bin/omnimark, должны быть созданы переменные:

```
OMNIMARK_INCLUDE /usr/local/bin/omnimark/xin/
OMNIMARK_XFLPATH /usr/local/bin/omnimark/lib/=L.so
```

Каждая ОС семейства UNIX имеет собственные правила установки переменных среды.

ПРИМЕЧАНИЕ Переменные среды OmniMark не имеют ничего общего с переменными среды CGI.

Создание CGI-программ на OmniMark

В этом разделе описаны основы написания сценариев CGI на OmniMark, включая работу с формами и переменными среды **CGI.**

Библиотека CGI OmniMark

Библиотека omcgi.xin реализует функции, необходимые для написания CGI-программ. Чтобы воспользоваться ими, добавьте в программу CGI на OmniMark строку:

```
include "omcgi.xin"
```

Получения данных от Web-сервера

Программы CGI получают данные от Web-сервера — поля и значения HTML-форм, имя сценария и IP-адрес клиента — как через переменные среды, так и через стандартный ввод.

Методы GET и POST

HTTP определяет несколько типов клиентских запросов, называемых методами. Чаще всего используются методы *GET* и *POST*. В HTML-коде это выглядит так:

```
<form action="Commit.xom" method="post">...
```

Когда обозреватель отправляет запрос *GET*, вся информация из формы закодирована в указателе ресурса. Например, подача формы с двумя полями *user* и *age* выглядит так:

```
http://localhost/Commit.xom?user=Randall+McMurtry&age=29
```

Часть указателя ресурса, расположенная после вопросительного знака, называется строкой запроса. Она содержит информацию, которую клиент передает Web-серверу для обработки.

Если клиент отправляет запрос *POST*, строка запроса не добавляется к указателю ресурса — она передается в теле HTTP-запроса,

Какой бы метод — *GET* или *POST* — вы ни использовали, можете вызвать функцию *cgiGetQuery* для доступа к данным формы.

Доступ к информации методов GET и POST

Функция *cgiGetQuery* из библиотеки omcgi.xin определяет, какой метод (*GET* или *POST*) использован для пересылки данных, декодирует данные и копирует пары «ключ — значение» в заданную вами потоковую полку.

Например, следующая программа выводит названия и значения каждого поля формы в запросе:

```
tf!C:\Omnimark\omnifnark-sb
declare #process-input has unbuffered
declare ttprocess-output has binary-mode
include "omcgi.xin"
process
   local stream query variable
   cgiGetQuery into query
   output "Content-Type: text/html" || CRLF i| CRLF
   repeat over query
         output key of query || ": " || query || "<br>"
   again
```

Допустим, эта программа запускается запросом:

```
http://localhost/cgi-bin/Commit.xom?user=Randall+McMurtry&age=29
```

Обозреватель выведет следующую информацию:

```
user: Randall McMurtry
age: 29
```

Такой тип кода — выполняющий итерации по значениям, переданным в строке запроса, — удобен для отладки CGI.

Доступ к переменным среды CGI

Функция *cgiGetEnv* из библиотеки omcgi.xin копирует пары «имя - значение» предопределенного набора переменных среды CGI в заданную вами потоковую полку. Набор включает в себя все переменные среды, используемые Web-сервером для передачи информации программам CGI.

Так, вы можете модифицировать программу Commit.xom, чтобы она показывала непроанализированную строку запроса перед началом ее анализа. Строка запроса передается программе CGI через переменную среды *QUERY_STRING*:

```
#!/usr/local/bin/omnimark -sb
declare tfprocess-input has unbuffered
declare «process-output has binary-mode
include "omcgi.xin"
process
   local stream query variable
   local stream env variable
   cgiGetQuery into query
```

```
cgiGetEnv into env
output "Content-Type: text/html" || CRLF || CRLF
output "Исходный вариант строки: "
       || env key "QUERY_STRING" || "<br>"
                                 when env has key "QUERY_STRING"
repeat over query
       output key of query || ": " || query || "<br>"
again
```

Допустим, эта программа запускается запросом:

```
http://localhost/cgi-bin/Commit2.xom?user=Randall+McMurtry&age=29
```

Обозреватель отобразит следующую информацию:

```
Черновой вариант строки: user=Randall+McMurtry&age=29
user: Randall McMurtry
age: 29
```

Обратите внимание наличие проверки строки запроса в операторе вывода. Web-сервер создаст переменную среды только при наличии данных. Следующий запрос не создаст переменную среды *QUERY_STRING*:

```
http://localhost/cgi-bin/Commit2.xom
```

Передача результатов Web-серверу

Передача информации Web-серверу довольно проста: CGI-программа выводит простой заголовок и следующую за ним содержательную информацию в стандартный поток вывода. Оператор вывода OmniMark записывает данные в стандартный поток вывода (*stream ttprocess-output*) по умолчанию.

Формирование заголовка HTTP

Заголовок документа **HTTP** должен быть выведен перед выводом содержимого документа. Эти заголовки обрабатываются Web-сервером и помогают определить тип содержимого, передаваемого обозревателю.

Большинство программ CGI пересылают HTML обозревателю в стандартном виде:

```
output
   "Content-Type: text/html"          ; <- заголовок
   || CRLF || CRLF CGI                ; <- разделитель
   || "Привет, мир OmniMark!"         ; <- тело
```

Заметьте: вы должны использовать макрос CRLF дважды для создания пустой строки между заголовком и телом документа. Не применяйте для этого конструкцию OmniMark *«%n»*.

Хотя заголовков много, обычно используется всего несколько. Например, если вы хотите перенаправить обозреватель на другой указатель ресурса, вместо заголовка Content-Type примените такой:

```
output
   "Location: http://www.omnimark.com"
   || CRLF || CRLF                          ; тело документа не требуется
```

Web-сервер, получив этот ответ, переласт обозревателю соответствующий заголовок (например, HTTP Status 302 — Страница перемешена), после чего обозреватель будет перенаправлен на нужную страницу.

Передача двоичных файлов

Часто бывает нужно передать обозревателю двоичные файлы (например, GIF-изображений). Чтобы быть уверенным, что OmniMark не изменит внутреннюю структуру файла, используйте модификатор *binary-mode*:

```
#!/usr/local/bin/omnimark -sb
declare tfprocsss-input has unbuffered
declare ftprocsss-output has binary-mode
include "omcgi.xin"
   process
         output "Content-Type: image/gif"
                || CRLF || CRLF
                || binary-mode file "C:\TravisBook\AppxA\Duck.gif "
```

Заметьте, что нужно объявить двоичным и поток вывода, и файл.

Небуферизированный стандартный ввод и вывод

Потоки OmniMark, включая *#process-input* и *#process-output*, по умолчанию буферизируются для повышения эффективности. Однако буферизированный ввод может создать проблемы при работе CGI-программ — программа может бесконечно долго ждать оставшуюся часть данных из-за буферизации потока OmniMark. Поэтому в начале всех программ CGI на OmniMark, которые получают ввод от обозревателя, должна присутствовать строка:

```
declare #process-input has unbuffered
```

При желании можно также отменить буферизацию потока *#process-output*:

```
declare #process-output has unbuffered
```

Это объявление повысит скорость передачи ответа обозревателем, особенно для программ CGI, которые сочетают вывод с продолжительной обработкой данных.

Повышение надежности CGI-программ

OmniMark предоставляет несколько методов для написания надежных программ. В среде CGI, возможно, наиболее важны методы перехвата ошибок, способных прервать обработку.

Перехват программных ошибок

При попытке выполнить некорректное действие OmniMark-программа будет прервана с выдачей сообщения об ошибке интерпретатора. Примером некорректных действий является установка ключевого значения при отсутствующем ключе, деление на 0. попытка доступа к несуществующему файлу или отсутствие дискового пространства.

Чтобы быть уверенным, что OmniMark не вернет обозревателю сообщение об ошибке интерпретатора, воспользуйтесь перехватом исключения *#program-error*:

```
#!/usr/local/bin/omnimark -sb
declare ttprocess-output has unbuffered
declare #process-output has binary-mode
include "omcgi.xin"
process
    local stream query variable
    local stream env variable
    cgiGetQuery into query
    cgiGetEnv into env
    output "Content-Type: text/html" || CRLF || CRLF
    output "Исходный вариант строки запроса: "
          || env key "QUERY_STRING" || "<br>"
    repeat over query
          output key of query || ": " || query || "<br>"
    again
    catch #program-error
          put #error "<br><b>Ошибка при выполнении программы CGI.</
b><br>"
```

Обратите внимание на отсутствие проверки при использовании полки *envf*. Если эта программа (пусть она называется Commit3.xom) вызвана запросом */localhost/cgi-bin/Commit3.xom*, обозреватель выдаст ответ:

```
Черновой вариант строки запроса: Ошибка при выполнении программы CGI.
```

Предотвращение просмотра исходного кода

Любой макрос OmniMark можно развернуть ключом командной строки *-expand*. Следовательно, любой человек, вызывающий CGI-программу на OmniMark, может, используя этот ключ, посмотреть текст программы в обозревателе (даже если в программе нет макросов). Это нежелательно со многих точек зрения, так что всегда конфигурируйте свой Web-сервер для вызова программ OmniMark с ключом *-noexpand*, запрещающим попытку применения ключа *-expand*.

Соберем все вместе

Теперь мы знаем достаточно для написания CGI-программы на OmniMark. В этот раздел я включил две программы, иллюстрирующие концепции, обсуждавшиеся в этой главе.

Использование простых форм

Первый пример отображает простую форму, которая запрашивает ваше имя и возвращает число гласных в нем. Для вызова программы просто поместите ее в ваш каталог *cgi-bin* и наберите в обозревателе запрос *localhost/cgi-bin/Vowel.xom*:

```
#!/usr/local/bin/omnimark -noexpand -sb
;
; URL localhost/cgi-bin/Vowel.xom
;
include "omcgi.xin"
declare #process-input has unbuffered
declare #process-output has unbuffered
declare tfprocess-output has binary-mode
global stream query variable
global stream env variable
define stream function get-script-name
   as
   return env^"SCRIPT_NAME" when env has key "SCRIPT_NAME"
   return ""
process
```

```
cgiGetQuery into query
cgiGetEnv into env
output "Content-type: text/html" || CRLF || CRLF
        | j "<html><head><title>Тестовая
CGI-программа на OmniMark</title></head><body>%n"
        || "<h1>OmniMark Счетчик</h1>%n"
do when query has key "name"
   local counter vowels initial {0}
   do when length of query^"name" > 0
          repeat scan query^"name"
                  natch unanchored Л ["aeiou"]
                                  increment vowels
          again
output "<p>" || query^"name" || ", у зас %d(vowels) гласных"
          output "s" unless vowels = 1
          output " в вашем имени.</p>%n"
   else
          output "<p><b>Вы не ввели ваше имя.</b></p>%n"
   done
done
output "<form action='"
        | I get-script-name
        || "' method='get'>Ваше имя: %n<input type='text'
        name='name'>%n"
        || "<input type='submit'
        value='Посчитать гласные'></form>"
        | | "</body></html>%n"
```

Доступ к внешним источникам данных

Второй пример CGI-программы — это программа на OmniMark, которая запрашивает точное время по атомным часам Национального института стандартов и технологий (г. Боулдер, шт. Колорадо). Время обновления по умолчанию составляет 60 секунд; но вы можете изменить его прямо в указателе ресурса. Например, запрос *localhost/cgi-bin/atomclock?refresh=10* установит десятисекундное время обновления,

Эта программа использует локальную проверку установленного значения времени обновления в строке запроса. Если значение не установлено или оно не цифровое, применяется значение по умолчанию. Блок *#program-error* перехватывает все ошибки, которые могут произойти

И, наконец, блок *always* гарантирует, что независимо от любых событий, которые могут произойти в программе, будет сформирован корректный вывод HTML.

```
#! /usr/local/bin/omnimark -noexpand -sb
;
; URL localhost/cgi-bin/atomclock?refresh=NN
;
declare tfmain-output has binary-mode
declare amain-output has unbuffered
include "omioprot.xin"
include "omtcp.xin"
include "omdate.xin"
include "omcgi.xin"
process
   local counter refresh-rate initial {60}
   local stream query variable
   output "Content-type: text/html" || CRLF ||*2
          It "<html><head><title>Точное местное время"
          || "</title></head>%n"
          || '<body bgcolor="ffffff" text="000000">%n'
   cgiGetQuery into query
   do
; установить проверку существования и типа времени обновления
          set refresh-rate to query^"refresh"
          catch tfprogram-error
                         ; использовать значение по умолчанию
   done
   do scan TCPConnectionGetCharacters
                TCPConnectionOpen on "132.163.135.130" at 14
          match white-space- cigit+ blank+ digit{2} => y "-"
                         digit{2} => rc "-
" digit{2} => d blank+ digit{2} => h
                         ":" digit{2} => mm ":" digit{2} => s
             output      '<meta http-equiv="refresh" content="'
                         || "d" % refresh-rate || '">%n'
                         || '<font face="Comic Sans MS">%n'
                         || '<p align="center">Местное время %n'
                         || "согласно атомным часам в%n"
                         || "<br>Боулдер, Колорадо<br><br><b>%n"
```

```
                        || format-
ymdhms "=xh:=m:=s =a.m. on =W, =n =xD, =xY"
                                                        with-
date ymdhms-adjust-time-zone
"20%x(y)%x(m)%x(d)%x(h)%x(min)%x(s)+0000"
                                                        to-
be date "=t"
                        || "%n</b></font><br><br></p>"
        else
            output "<h1>Недопустимый формат даты!</h1>%n"
        done
    catch #program-error message the-problem
        output "<p>" || the-problem || "</p>%n"
              || "<p><b>Ошибка при выполнении  программы CGI.</b></
p>%n"
    always
        output "</body></html>"
```

Файлы параметров и проектов OmniMark

OmniMark C/VM поддерживает ряд параметров командной строки, с которыми вы можете ознакомиться с помощью ключа командной строки *-help*:

```
omnimark -help
```

Часто Web-серверы позволяют использовать подстановку только одного параметра — имени программы. Вам может потребоваться большее число параметров, самым важным из которых является имя файла протокола для записи ошибок компиляции и интерпретации. Вы можете решить эту проблему, применив файлы параметров или проектов. Оба типа файлов выполняют одинаковые задачи, однако файл проекта более соответствует современным требованиям.

Файл параметров OmniMark

По соглашению файл параметров OmniMark имеет расширение .xar, так что файл параметров программы Hello.xom будет называться Hello.xar

Файлом параметров называется файл, содержащий параметры командной строки, которые должны быть переданы программе на OmniMark. Для передачи файла параметров OmniMark служит ключ *-f*:

```
C:\omnimark\omnimark -noexpand -f Hello.xar
```

Допустим файл hello.xar содержит строки:

```
-sb hello.xom
-alog Hello.log
```

Это равносильно команде запуска программы:

```
C:\omnimark\omnimark -noexpand -sb Hello.xom -alog Hello.log
```

OmniMark читает файл параметров, игнорируя комментарии и дополнительные пробелы, и рассматривает содержимое исключительно как параметры командной строки.

Использование псевдокомментариев в файле параметров

ОС семейства UNIX не имеют механизма связывания программ-обработчиков с расширениями файлов. Они целиком полагаются на псевдокомментарий в первой строке исходного файла. Его можно разместить и в файле параметров:

```
#!/usr/local/bin/omnimark -f
-sb Hello.xom
-alog /www/root/log/Hello.log
```

ПРИМЕЧАНИЕ Для ОС семейства UNIX файл с расширением .xar должен быть объявлен как исполняемый.

Допустим, файл Hello.xar расположен в каталоге /www/root/cgi-bin/. Тогда обращение к приведенному ниже ресурсу вызовет программу hello.xom, а все сообщения будут записаны в файл протокола Hello.log:

```
/localhost/cgi-bin/hello.xar
```

Файл проекта OmniMark

OmniMark поддерживает более современный тип файлов параметров — файл проекта. По соглашению эти файлы имеют расширение .xop, так что для программы OmniMark Hello.xom файл проекта будет называться Hello.xop.

Файл проекта имеет формат XML и выполняет те же функции, что и файл параметров; он вызывается аналогичным способом. Он также может содержать псевдокомментарий в первой строке (хотя XML не поддерживает эту нотацию).

Интегрированная среда разработки OmniMark создает файл проекта автоматически, а его возможности в ближайшем будущем будут значительно расширены,

ПРИМЕЧАНИЕ Если вы разрабатываете CGI-программы, используя IDE, убедитесь, что добавили ключ *-brief* на вкладке «Внешние» окна «Параметры проекта». Если вы не сделаете этого, OmniMark выведет свое сообщение прежде, чем ваша программа начнет вывод заголовков HTTP, что приведет к путанице, и работа вашей программы CG1 будет прервана.

Компиляция CGI-программ

Компания OmniMark предлагает разработчикам компилятор байт-кода (его могут получить все подписчики сети разработчиков OmniMark). Подробную информацию см. на сайте *http://www.omnimark.com.* Компиляция приложений CGI имеет ряд преимуществ.

- **Эффективность выполнения** — большие приложения CGI будут работать гораздо быстрее, поскольку отпадает необходимость в их компиляции при каждом обращении.
- **Зашита кода приложений** — байт-код OmniMark не может быть декомпилирован.
- **Зашита бизнес-приложений** — бизнес-логика приложения не пострадает, даже если посторонний получит доступ на Web-сервер.
- **Защита интеллектуальной собственности** — коммерческие CGI-приложения можно распространять в виде байт-кода, что надежно защитит вашу интеллектуальную собственность.

По соглашению файлы, содержащие байт-код OmniMark, имеют расширение .xvc. Программы в виде байт-кода выполняются так же, как и исходный код. Ассоциация «расширение — обработчик», которую вы должны задать на Web-сервере, выглядит так:

```
.xvc = C:\OmniMark\omnivm -load %s
```

Поскольку исходный код отсутствует, в ключе *-noexpand* нет необходимости.

При запуске байт-кода можно использовать файлы параметров. Файл параметров VM имеет расширение .xva. Как и в случае с исходным кодом, первая строка файла параметров VM может содержать команду запуска виртуальной машины. Ассоциация «расширение — обработчик», которую вы должны задать на Web-сервере, выглядит так:

```
.xva = C:\OmniMark\omnivm -f %s
```

Дополнительная информация об OmniMark

Это приложение представляет собой краткое введение в язык программирования OmniMark. В качестве дополнительной информации о языке рекомендую мою книгу «Работа с OmniMark, Том 1: Приступая к работе» *(OmniMark at Work, Volume 1: Getting Started)*, включенную в состав компакт-диска книги. Хотя книга выпущена несколько лет тому назад и не содержит детального изложения концепций сетевого программирования, она все равно полезна для изучения базовых конструкций языка, включая типы данных, управляющие структуры, ссылочные переменные, структурирование обработки текстов и т. д.

Web-сайт компании OmniMark (*http://www.omnimark.com*)также содержит богатую коллекцию статей, советов и обеспечивает поддержку разработчиков по электронной почте и через списки рассылки.

Если вы работаете в области сетевого программирования, вам стоит изучить возможности OmniMark по созданию кросс-платформенных, высокопроизводительных, масштабируемых Web-приложений,

ПРИМЕЧАНИЕ Этот материал создан на основе статей, написанных сотрудниками компании OmniMark © OmniMark Technologies Corporation, 1988–2000. За поддержкой обращайтесь по адресу http://www.omnimark.com. (Однако все похвалы в адрес этого великолепного продукта принадлежат мне!)

Приложение Б

BizTalk™ Framework 2.0: спецификация документов и сообщений

Дата публикации: декабрь 2000 г.

Настоящая спецификация содержит описание концептуальной архитектуры BizTalk Framework 2.0, включая структуру документа и сообщения BizTalk. Спецификация подробно описывает создание документов и сообщений BizTalk и их безопасную передачу по существующим транспортным протоколам Интернета.

1. Введение

Широкое распространение и всесторонняя поддержка защищенных транспортных протоколов Интернет открывает новые возможности для эффективного автоматизированного взаимодействия компаний в различных сферах бизнеса. Такое взаимодействие может осуществляться как между внутренними приложениями и системами в пределах информационной инфраструктуры отдельной компании, так и с приложениями, используемыми клиентами и партнерами, и со службами, предоставляемыми коммерческими и корпоративными провайдерами.

Проблемы организации эффективного автоматизированного взаимодействия приложений разных компаний подобны тем, что возни-

кают при решении тех же задач в рамках компании или подразделения, но не во всем. Межкорпоративное взаимодействие должно отличаться особой надежностью и защищенностью информации.

Для организации межкорпоративного взаимодействия, в частности, требуются:

- достаточно гибкий универсальный язык, позволяющий описывать, упаковывать, публиковать и пересылать структурированную и неструктурированную информацию между приложениями и компаниями;
- гибкий многофункциональный язык для описания и преобразования информации из одного формата в другой при передаче между приложениями и компаниями;
- кросс-платформенные коммуникационные протоколы прикладного уровня, поддерживающие автоматизированное взаимодействие локальных и удаленных приложений;
- кросс-плат форменные механизмы обеспечения целостности, защиты и идентификации сообщений.

Расширяемый язык разметки (Extensible Markup Language, XML) и основанные на нем языки описания схем образуют открытый технологический фундамент для решения этих задач. Эти языки позволяют описывать структуру информации и организовать кросс-платформенный и нейтральный относительно ПО промежуточного уровня обмен такой информацией между приложениями и бизнес-партнерами.

Как следствие, стандартизирующие органы и инициативные группы в разных отраслях бизнеса начали активно применять XML и схемы на основе XML для описания словарей и моделей документов своей предметной области. Схемы публикуются и становятся доступными, что значительно расширяет возможности взаимодействия приложений и компаний. Популярность XML привела к появлению значительного числа решений для обмена информацией на базе XML с существующими приложениями. Было предпринято несколько попыток решения проблемы отсутствия нейтральных коммуникационных протоколов прикладного уровня, однако вследствие ориентации на конкретную платформу или решение они не смогли решить всех проблем развертывания в сложной гетерогенной среде.

Частные попытки решения этой проблемы, как правило, кончались перекладыванием ответственности за обеспечение совместимости на плечи клиентов. Инициатива BizTalk™ Framework призвана обеспечить кросс-платформенное и технологически нейтральное решение. В рамках этой инициативы создана спецификация для проек-

тирования и разработки XML-решений для обмена данными между приложениями и организациями. Спецификация основана на существующих и появляющихся стандартах Интернета, таких как протокол передачи гипертекстовой информации (Hypertext Transfer Protocol, HTTP), многоцелевые расширения электронной почты Интернета (Multipurpose Internet Mail Extensions, MIME), расширяемый язык разметки (Extensible Markup Language, XML) и упрощенный протокол доступа к объектам (Simple Object Access Protocol, SOAP). Следующие версии инфраструктуры BizTalk будут расширены за счет включения новых стандартов в области XML и Интернет-коммуникаций.

Инициатива BizTalk™ не ставит своей целью решение всех проблем электронной коммерции в секторе «бизнес — бизнес». Скажем, спецификация не касается юридических аспектов, вопросов арбитража и устранения последствий катастрофических событий, так же как и организации конкретных бизнес-процессов. Инфраструктура BizTalk™ предлагает набор базовых механизмов для решения большинства проблем электронной коммерции «бизнес — бизнес». Ожидается, что для решения прикладных проблем будут разработаны другие спецификации и стандарты, совместимые с этой инфраструктурой.

2. Область действия и структура спецификации

Спецификация начинается с обзора концептуальной архитектуры BizTalk, включая фундаментальные понятия документа и сообщения BizTalk. Затем подробно описаны структура документа и сообщения BizTalk и их защищенная передача по различным транспортным и коммуникационным протоколам Интернета.

Широкие функциональные возможности и предсказуемую семантику обеспечивает соответствие документа BizTalk нескольким правилам. Настоящая спецификация описывает следующие аспекты документов BizTalk и их семантики:

- структура документа BizTalk;
- заголовки документа BizTalk, предназначенные для его маршрутизации, описания свойств, каталогизации и управления процессами обработки;
- структура и обработка документов BizTalk, требующих гарантированной доставки.

При разработке решений на основе инфраструктуры BizTalk необходимо использовать средства соответствующих транспортных протоколов, чтобы обеспечить доставку сообщений и защиту их содер-

жимого. В настоящей спецификации описаны следующие механизмы и аспекты кодирования и транспортировки сообщений BizTalk:

- привязки к конкретным транспортным протоколам Интернета;
- кодирование сообщений при передаче и упаковка вложений средствами MIME;
- применение цифровой подписи и шифрования средствами S/MIME.

Цель настоящей спецификации — описать обмен сообщениями между серверами, отвечающими спецификации BizTalk Framework 2.0. В тексте спецификации они называются *BFC-совместимыми серверами*.

2.1 Взаимосвязь со спецификацией BizTalk Framework 1.0

Спецификация BizTalk Framework 2.0 значительно пересмотрена по сравнению с версией 1.0. В настоящей версии описаны и средства, отсутствовавшие в предыдущей версии, а именно:

- взаимосвязь с транспортными протоколами;
- гарантированная доставки сообщений;
- применение MIME для передачи вложений;
- средства защиты на базе S/MIME.

При разработке спецификации BizTalk Framework 2.0 учтены недавно появившиеся стандарты, включая:

- SOAP 1.1 [*SOAP*];
- XML-схемы. Часть 1: структуры [*XSD1*];
- XML-схемы. Часть 2: типы данных [*XSD2*].

Наиболее важным для данной спецификации является стандарт SOAP 1.1, поскольку инфраструктура BizTalk Framework 2.0 является расширением SOAP 1.1, тогда как спецификация BizTalk Framework 1.0 была выпущена до появления SOAP. Кроме того, в рамках пересмотра спецификации с учетом накопленного опыта и необходимости поддержки новых возможностей была рационализирована семантика, принципы именования и структура тэгов BizTalk. Применение SOAP 1.1 в качестве базового стандарта включает инфраструктуру BizTalk Framework 2.0 в рамки общего направления развития протоколов на основе XML в соответствии с рекомендациями консорциума W3C [*XP*].

Одна из целей инфраструктуры BizTalk Framework 2.0 — дать достаточно подробное описание обмена сообщениями как основы взаимодействия BFC-совместимых серверов. Как следствие, семантика многих тэгов значительно уточнена в сравнении с версией 1.0. Были приложены все усилия по сохранению структуры и содержания доку-

ментов BizTalk, описанных в спецификации BizTalk Framework 1.0, однако в настоящей спецификации приоритет отдан ясности семантики и соответствию стандартам.

2.2 Модель версий

Спецификация BizTalk Framework 2.0, как и SOAP 1.1. не использует традиционную модель нумерации версий. Версия определяется универсальным идентификатором пространства имен заголовков BizTalk, описанных в настоящей спецификации. По правилам SOAP 1.1 в отношении атрибута **SOAP-ENV:mustUnderstand** BFC-сервер должен реагировать сообщением об ошибке на наличие в документе обязательных заголовков некорректной формы или с неверно заданным пространством имен.

В контексте транспорта HTTP (раздел 10 настоящей спецификации) сообщение об ошибке может быть возвращено в HTTP-ответе. Однако если сообщение обрабатывается асинхронно, необходимо использовать код статуса **HTTP** 202 **accepted** и по возможности вернуть сообщение об ошибке. Подробнее о привязке функций SOAP к различным транспортным протоколам см. раздел 10.

3. Использование типов данных XML-схем

В настоящей спецификации используется атрибут описания типа **xsi:type** и несколько других типов данных из спецификации XML-схем *[XSD1, XSD2]*. Все они с комментариями перечислены ниже. Тем не менее, настоящая спецификация не полагается ни на какой конкретный синтаксис XML-схем.

Атрибут **xsi:type** позволяет зафиксировать тип элемента конкретного XML-документа, что бывает полезно при проверке структуры элемента.

Тип данных **timeInstant** применяется для хранения даты и времени. Значения элементов типа **timelnstant** представляют собой дату и время согласно разделу 5.4 стандарта ISO 8601 [*ISO8601*].

Тип данных **uriReference** представляет универсальный идентификатор ресурса согласно определению в разделе 4 документа [*URI*]. Идентификатор может быть абсолютным или относительным и содержать дополнительный идентификатор фрагмента.

Тип данных **complexType** применяется для элементов, содержание которых не описывается базовыми типами данных (как-то: строка или десятичное число); обычно такой элемент содержит вложенные элементы и/или атрибуты со своим содержанием.

Из соображений краткости ссылки на типы данных XML-схем приводятся без префикса **xsd:,** описывающего их принадлежность к пространству имен *http://www.w3.org/2000/08/XMLSchem*В примерах и фрагментах кода префиксы пространства имен приводятся в обязательном порядке.

4. Концепции

4.1 Терминология

Данный раздел содержит определения терминов BizTalk, используемых в настоящем документе.

- **BizTalk-совместимый сервер** [BizTalk Framework Compliant (BFC) Server] — набор служб, реализующих функции обработки сообщений, описанные в спецификации BizTalk Framework.
- **Приложение** (Application) — любое приложение, реализующее и исполняющее бизнес-логику. В приложение могут быть включены дополнительные компоненты-адаптеры, необходимые для передачи и приема бизнес-документов (см. ниже) и взаимодействия с BFC-сервером.
- **Бизнес-документ** (Business Document) — правильный XML-документ, содержащий бизнес-данные. В качестве данных (наполнения документа) может выступать заказ, счет, прогноз продаж и любая другая информация делового характера. Один или более бизнес-документов составляют тело документа BizTalk (см. ниже).

  Спецификация BizTalk Framework не накладывает ограничений на содержание или структуру (схему) бизнес-документов. Эти аспекты разрабатываются и согласовываются компаниями-участниками электронного документооборота.
- **Схемы** (Schema) — метаданные, описывающие структуру класса XML-документов и, в частности, бизнес-документов. Такое описание позволяет разработчикам создавать системы для обработки бизнес-документов, а синтаксическим анализаторам — проверять соответствие конкретного бизнес-документа схеме.

  Организации могут публиковать свои схемы в библиотеке схем BizTalk (*http://www.biztalk.org/library/library.asp*) или другими методами.

ПРИМЕЧАНИЕ Схема бизнес-документа не содержит Biz-тэгов, описываемых в настоящей спецификации. Схема должна содержать только тэги, необходимые для описания транзакции в соответствии с соглашением сторон, участвующих в обмене документами. Общие требования к схемам и рекомендации по их подготовке см. в «Руководстве по подготовке схем BizTalk» (*http://www.biztalk.org/resources/schemasguide.asp*).

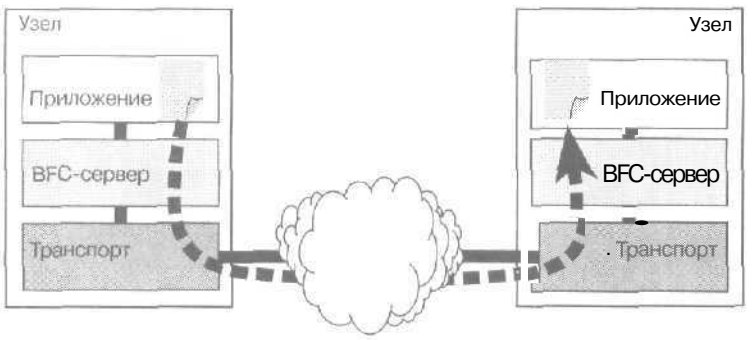
- **Документ BizTalk** (BizTalk Document) — сообщение в формате SOAP 1.1, тело которого содержит бизнес-документы, а заголовок — специальные тэги BizTalk, описывающие обработку сообщения. К документам BizTalk применимы следующие понятия.
  - **Время жизни** (Lifetime) — срок, в течение которого документ сохраняет свое значение. По окончании этого срока документ не следует отправлять или принимать, так же как и уведомлять о его поступлении.
  - **Идентификатор** (Identity) — уникальный идентификатор документа.
  - **Прием** (Acceptance) — согласие принимающей стороны выполнить необходимые действия по доставке документа. Документ принимается, когда принимающая сторона распознала документ как предназначенный для получателя, которого она обслуживает. Прием сообщения не означает просмотра и верификации всех заголовков и тела сообщения.
  - **Идемпотентность** (Idempotence) — способность многократно передавать и принимать документ с теми же последствиями, что и при однократной передаче или приеме.
  - **Уведомление** (Receipt) — спецификация BizTalk Framework предусматривает обмен уведомлениями между сторонами при необходимости уточнения семантики доставки документа. В данном контексте уведомление — это документ BizTalk одного из двух типов:
    - **уведомление** о доставке (Delivery Receipt) — сообщение о приеме документа;
    - **подтверждение** (Commitment) — помимо приема документа, уведомляет о проверке документа получателем, корректности всех элементов заголовка, имеющих атрибут **mustUnderstand=»1".** и их содержимого, корректности содержимого документа, а также о намерении обработать документ.

- **Бизиес-тэги** (BizTag) — набор обязательных и дополнительных XML-тэгов, описывающих пересылку и обработку бизнес-документов. Более точно, бизнес-тэги — это описанные в настоящей спецификации элементы и атрибуты, служащие для создания специфических для BizTalk элементов заголовка SOAP документа BizTalk. Эти тэги обрабатываются BFC-сервером и другими приложениями, участвующими в обмене документами.
- **Сообщение BizTalk** (BizTalk Message) — квант обмена информацией между BFC-серверами. Сообщения BizTalk служат для передачи документов BizTalk и любых связанных файлов между BFC-серверами. Сообщение BizTalk обязано содержать основной документ BizTalk, описывающий семантику сообщения в рамках спецификации BizTalk. Кроме того, сообщение может содержать произвольное число вложений (см. ниже), включая правильные XML-документы, могущие (но не обязанные) быть документами BizTalk. Документы BizTalk, пересылаемые как вложения, трактуются как обычные документы XML и не влияют на семантику сообщения BizTalk. Структура сообщения BizTalk зависит от транспорта, используемого для передачи сообщения, и может включать заголовки, присущие конкретному транспортному протоколу.
- **Транспорт** (Transport) — обмен сообщениями BizTalk между BFC-серверами предполагает наличие механизма, обеспечивающего физическую транспортировку сообщений между отправителем и адресатом. В отношении этого механизма мы используем термин *транспорт*. В данном контексте в качестве транспорта может выступать простейший механизм передачи дейтаграмм или файлов, полнофункциональные коммуникационные протоколы HTTP и SMTP и специализированное ПО промежуточного слоя. В настоящей спецификации не делается предположений о возможностях транспорта. Характеристики транспорта влияют только на транспортные привязки, описанные в разделе 10.
- **Пункт** (Endpoint) — BizTalk-совместимый отправитель или адресат сообщения BizTalk.
- **Адрес** (Address) — местонахождение пункта, описанное с достаточной степенью подробности для транспортировки и доставки сообщения.
- **Вложение** (Attachment) — любая информация, связанная с бизнес-документом, но не включенная в тело документа BizTalk. В этом качестве может выступать изображение, архив или любой другой файл, формат которого не соответствует требованиям к бизнес-документу. Вложения могут пересылаться в теле сообщения Biz-

Talk, объемлющего соответствующий документ BizTalk, или быть внешними по отношению к сообщению.

4.2 Логическая структура

Логическая реализация инфраструктуры BizTalk, описанная в настоящей спецификации, состоит из трех уровней. Поскольку спецификация BizTalk описывает лишь структуру сообщений BizTalk и протокол их гарантированной доставки, в реализации может использоваться любая другая логическая организация, обеспечивающая необходимые функциональные возможности, причем такое решение не скажется на соответствии спецификации. Логические уровни модели включают приложение (и соответствующие адаптеры), BFC-сервер и транспорт. Приложение является единственно возможным источником и адресатом сообщений BizTalk, и взаимодействует с другими приложениями, передавая бизнес-документы через BFC-серверы. BFC-серверы общаются по разным протоколам, включая HTTP, SMTP и очередь сообщений (Microsoft Message Queue, MSMQ). Спецификация BizTalk не предъявляет требований к транспортным протоколам и не зависит от особенностей их реализации.



Приложение отвечает за генерацию бизнес-документов и вложений, которые будут передаваться партнерам через BFC-серверы. Ответственность за преобразование бизнес-документа в документ BizTalk может быть возложена как на приложение, так и на BFC-сервер (решение этого вопроса зависит от реализации BFC-сервера). Сервер обрабатывает документ и вложения и создает сообщение BizTalk с учетом особенностей выбранного транспорта. Кроме того, с помощью бизнес-тэгов сервер определяет адрес пункта назначения в контексте выбранного транспорта. После этого сервер передает сообщение транспортному уровню для передачи BFC-серверу — адресату сооб-

шения. Интерфейсы между бизнес-приложением, BFC-сервером и транспортным уровнен определяются при реализации.

5. Структура документа BizTalk

Ниже приведен пример простого документа BizTalk.

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
   <SOAP-ENV:Header>
      <eps:endpoints SOAP-ENV:mustUnderstand="1"
         xmlns:eps="http://schemas.biztalk.org/btf-2-0/endpoints"
         xmlns:agr="http://www.trading-agreements.org/types/">
         <eps:to>
            <eps:address xsi:type="agr:department">Book Orders
            </eps:address>
         </eps:to>
         <eps:from>
            <eps:address xsi:type="agr:organization">Book Lovers</
            eps:address>
      </eps:from>
      </eps:endpoints>
      <prop:properties SOAP-ENV:mustUnderstand="1"
         xmlns:prop="http://schemas.biztalk.org/btf-2-0/properties">
            <prop:identity>uuid:74b9f5d0-33fb-4a81-b02b-5b760641c1d6</
            prop:identity>
            <prop:sentAt>2000-05-14T03:00:00+08:00</prop:sentAt>
            <prop:expiresAt>2000-05-15T04:00:00+08:00</prop:expiresAt>
            <prop:topic>http://electrocommerce.org/purchase_order/</
            prop:topic>
      </prop:properties>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <po:PurchaseOrder xmlns:po="http://electrocommerce.org/
      purchase_order/">
         <po:Title>Essential BizTalk</po:Title>
      </po:PurchaseOrder>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Этот документ представляет собой стандартное сообщение SOAP 1.1, включающее:

- прикладной бизнес-документ (в данном случае — заказ книги) со своим пространством имен, передаваемый в теле сообщения SOAP;
- специфичные для BizTalk элементы заголовка SOAP <endpoints> и <properties>, созданные с помошью бизнес-тэгов пространства имен BizTag, схема и семантика которых описана в настоящей спецификации.

В принципе тело сообщения SOAP, образующего документ BizTalk, может содержать несколько бизнес-документов, а его заголовок — несколько элементов BizTalk (и, возможно, элементов из других пространств имен). Атрибут **SOAP-ENV:mustUnderstand** со значением «1» в соответствии со спецификацией SOAP 1.1 требует от пункта назначения документа понимания и обработки элементов заголовка с таким атрибутом. В противном случае обработка документа должна быть прервана, а отправителю документа должно быть возвращено сообщение об ошибке.

Все бизнес-тэги определены в стандартном пространстве имен BizTag с базовым идентификатором ресурса *http://schemas.biztalk.org/btf-2-O/*. Область действия пространств имен BizTag ограничена заголовком внешнего документа BizTalk. Если документ BizTalk целиком вложен в другой документ BizTalk, бизнес-тэги вложенного документа являются недействительными («спящими») — в процессе обработки объемлющего бизнес-документа они рассматриваются как бизнес-данные, не подлежащие синтаксическому анализу.

Маршрутные тэги <to> и <from>, подробно описанные ниже, часто используют информативные названия участников документооборота, а не адреса выбранного транспортного протокола (скажем, указатели ресурсов HTTP). Форма и интерпретация содержимого адреса определяются атрибутом xsi:type. Структура и функции документа BizTalk не зависят от транспорта, по которому передается документ.

6. Тело документа BizTalk

Элемент <Body> сообщения SOAP, образующего документ BizTalk. содержит передаваемый бизнес-документ. В принципе документ BizTalk может нести несколько связанных бизнес-документов (например, заказ, данные о заказчике и адрес, по которому следует выслать заказ).

Содержимое связанных бизнес-документов часто пересекается. SOAP предлагает простой механизм хранения данных, используемых несколько раз, — атрибут XML ID и относительные идентификаторы ресурсов. Рассмотрим несложное расширение примера с заказом,

когда заказ и данные о доставке ссылаются на данные о заказанной книге. Элемент <Body> приведенного ниже документа BizTalk демонстрирует решение этой задачи средствами SOAP,

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
   <SOAP-ENV:Header>
      <!- заголовки опущены для краткости ->
   </SOAP-ENV:Header>
   <SOAP-ENV:3ody>
      <po:PurchaseOrder
      xmlns:po="http://electrocommerce.org/purchase_order/">
         <po:item href="#theBook"/>
         <!- дополнительная информация о заказе ->
      </po:PurchaseOrder>
      <ship:shippingInfo xmlns:ship="http://electrocommerce.org/
      shippingInfo/">
         <ship:content href="#theBook"/>
         <!- дополнительная информация о доставке ->
      </ship:shippingInfo>
      <book xmlns="http://electrocommerce.org/bookInfo/" id="theBook"
         SOAP-ENC:root="0">
         <Title>Essential BizTalk</Title>
         <!- дополнительная информация о книге ->
      </book>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Этот пример также иллюстрирует и проблему, присущую этому подходу. Мы хотим, чтобы адресат рассматривал документ BizTalk как два бизнес-документа: заказ и информацию о доставке. Однако элемент <Body> сообщения SOAP содержит три дочерних элемента. Таким образом, нам нужно уметь отличать бизнес-документы от совместно используемых (посредством нескольких ссылок) прямых потомков элемента <Body>.

Спецификация SOAP предусматривает атрибут **SOAP-ENC:root**, сигнализирующий, что элемент не является независимым. Итак, любой прямой потомок элемента <Body> документа BizTalk — это вложенный бизнес-документ кроме случая, когда у потомка есть атрибут **SOAP-ENC:root** со значением 0.

7. Элементы заголовка документа BizTalk

В этом разделе описаны элементы заголовка SOAP, допустимые в документе BizTalk. Они служат для маршрутизации документа, каталогизации его содержимого и вложений и отслеживания контекста бизнес-процессов, в которых участвует документ. Бизнес-тэги заголовка и их свойства перечислены в приведенной ниже таблице. Каждый из заголовков подробно описан в следующих разделах; схемы соответствующих элементов приведены в приложении.

Тэг	Обязательный?	Тип	Тип данных	Вхождение
endpoints	да	элемент	ComplexType	однократное
properties	да	элемент	ComplexType	однократное
services	нет	элемент	CompIexType	однократное
manifest	нет	элемент	ComplexType	однократное
process	нет	элемент	ComplexType	однократное

7.1 Источник и пункт назначения документа

Отправитель и адресат документа указываются в SOAP-заголовке, отмеченном бизнес-тэгом <endpoints>. Ниже приведен соответствующий заголовок из нашего примера документа BizTalk:

```
<eps: endpoints SOAP-ENV:mustUnderstand="1"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:eps="http://schemas.biztalk.org/btf-2-0/endpoints"
   xmlns:agr="http://www.trading-agreements.org/types/">
   <eps:to>
      <eps:address xsi:type="agr:department">Book Orders
      </eps:address>
      </eps:to>
      <eps:from>
         <eps:address xsi:type="agr:organization">Book Lovers</
         eps:address>
      </eps:from>
</eps:endpoints>
```

При обсуждении примера мы отметили, что информативные названия отправителя и адресата задаются в элементе, отмеченном биз-

нес-тэгом <address>, и что эти имена, как правило, относятся к соответствующему прикладному пространству имен, а не к транспорту. Транспорт, по которому передается документ BizTalk, и пункты отправления и назначения в смысле транспортного протокола чаще всего выбираются сторонами отдельно. Более того, весьма вероятно наличие нескольких транспортов и, соответственно, пунктов отправления/назначения, а также их изменение со временем без изменения названий сторон (в элементе <address>) и структуры документов BizTalk, которыми обмениваются стороны. Правила маршрутизации доставки документа BizTalk по достижении BFC-сервера, выступающего в качестве пункта назначения, определяется при реализации. Тем не менее, использование транспортных пунктов или указателей ресурса в качестве содержимого элемента <address> не возбраняется.

Понимание и обработка заголовка <endpoints> и всего его содержимого BFC-сервером получателя обязательны для успешной обработки документа BizTalk. В связи с этим элемент <endpoints> должен всегда содержать атрибут **SOAP-ENV:mustUnderstand=»1"**. В таблице перечислены бизнес-тэги, предназначенные для создания дочерних элементов заголовка <endpoints>, и их свойства.

Тэг	Обязательный?	Тип	Типданных	Вхождение
to	**Да**	**элемент**	ComplexType	**однократное**
from	**Да**	элемент	**ComplexType**	**однократное**

Тэг <to> содержит описание получателя документа BizTalk. Этот элемент должен содержать один и только один вложенный элемент <address>.

Тэг <from> содержит описание отправителя документа BizTalk. Этот элемент должен содержать один и только один вложенный элемент <address>,

Элемент <address> содержит идентификатор участника обмена в текстовой форме. У элемента <address> есть обязательный атрибут **xsi:type**. Его значение указывает тип адреса, а также допустимую структуру его строчной формы. В примерах настоящей спецификации используется несколько категорий, включая название организации и универсальный идентификатор ресурса.

7.2 Идентификация документа и его свойства

Идентификатор и другие свойства документа задаются в элементе заголовка SOAP, обозначаемого бизнес-тэгом <properties>. Ниже приведен соответствующий заголовок из нашего примера документа BizTalk:

```
<prop:properties SOAP-ENV:mustUnderstand="1"
   xmlns:prop="http://schemas.biztalk.org/btf-2-0/properties">
   <prop:identity>uuid:74b9f5d0-33fb-4a81-b02b-5b760641c1d6
   </prop:identity>
   <prop:sentAt>2000-05-14T03:00:00+08:00</prop:sentAt>
   <prop:expiresAt>2000-05-15T04:00:00+08:00</prop:expiresAt>
   <prop:topic>http://electrocommerce.org/purchase_order/
   </prop:topic>
</prop:properties>
```

Понимание и обработка заголовка <properties> и всего его содержимого BFC-сервером получателя обязательны для успешной обработки документа BizTalk. В связи с этим элемент <properties> должен всегда содержать атрибут **SOAP-ENV:mustUnderstand=»1"**. **В** таблице перечислены бизнес-тэги, предназначенные для создания дочерних элементов заголовка <properties>, и их свойства.

Тэг	Обязательный?	Тип	Тип данных	Вхождение
identity	Да	элемент	uriReference	однократное
sentAt	да	элемент	timeInstant	однократное
expiresAt	да	элемент	time Instant	однократное
topic	ла	элемент	uriReference	однократное

Тэг <identity> - универсальный идентификатор ресурса, однозначно идентифицирующий документ BizTalk в целях протоколирования, отслеживания, обработки ошибок и решения других задач, связанных с обработкой и учетом документа. Значение тэга <identity> должно быть глобально уникальным. Это требование может быть выполнено, например, с помощью универсально-уникальных идентификаторов (Universally Unique Identifiers, UUIDs), как сделано в приведенном выше примере, или посредством применения к бизнес-документу *(документам)* криптографических алгоритмов хэширования (например, MD5). Выбор формы тэга <identity> и способа генерации его значений <identity> определяется при реализации.

Тэг <sentAt> содержит дату и время отправки документа. С учетом возможности повторной передачи документа (см. раздел 8), дата и время должны отражать момент генерации содержимого элемента <properties>.

Тэг <topic>, содержащий однозначный идентификатор назначения документа BizTalk, может служить для организации тематической маршрутизации (например, в модели «издатель — подписчик» с

тематической адресацией) и для проверки соответствия содержания документа BizTalk Document его назначению. Последнее, в частности, используется в HTTP-привязках (см. раздел 10). Хотя детали создания этого элемента выбираются при реализации, рекомендуется, чтобы тема документа задавалась приложением-отправителем или назначалась BFC-сервером, исходя из содержания бизнес-документов, передаваемых в составе документа BizTalk (например, в качестве темы может выступать указатель ресурса пространства имен первого бизнес-документа).

Тэг <expiresAt> указывает момент окончания срока действия документа. После указанного момента данный документ BizTalk считается устаревшим, и получатель не должен ни обрабатывать, ни подтверждать такой документ. Срок действия следует задавать с учетом времени, необходимого на синхронизацию в распределенных системах.

ПРИМЕЧАНИЕ Как правило, бизнес-процессы компании-отправителя документа задают ограниченный срок для выполнения операций, представленных (или описанных) документом BizTalk; кроме того, отправитель рассчитывает на уведомление в случае невозможности доставки документа за указанное время. Описать срок действия документов можно двумя методами: указав время жизни (Time To Live, TTL) или максимальный срок действия, а также задав момент, по наступлении которого документ считается устаревшим. У каждого подхода есть недостатки. Время жизни нечувствительно к проблемам синхронизации времени между разными компьютерами; этот метод оптимален при передаче средне- и краткосрочных документов по быстрым сетям. Этот метод исходит из предположения о мгновенной передаче сообщений, что в отношении некоторых протоколов может оказаться слишком оптимистичным. Таким образом, с учетом независимости инфраструктуры BizTalk от транспорта, применение времени жизни для описания срока действия документов проблематично. При использовании времени устаревания документа (задается в шкале всемирного координированного времени) возникает проблема синхронизации часов в распределенных системах. Однако ожидаемый срок действия сообщений, обмен которыми описывает настоящая спецификация, по-видимому, будет достаточно велик, чтобы проблема синхронизации часов не имела решающего значения. Поэтому в настоящей спецификации для описания срока действия применяется второй подход — с указанием момента времени устаревания документа. В будущих версиях инфраструктура BizTalk, возможно, будет поддерживать оба метода.

7.3 Доставка

Необходимость гарантированной доставки документа BizTalk указывается необязательным элементом заголовка SOAP, обозначаемого бизнес-тэгом <services>. Ниже приведен пример этого заголовка.

```
<services
   xmlns="http://schemas.biztalk.org/btf-2-0/services"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2000/08/XMLSchema"
   xmlns:agr="http://www.trading-agreements.org/types/"
   SOAP-ENV:mustUnderstand="1">
      <deliveryReceiptRequest>
         <sendTo>
            <address xsi:type="agr:httpURL">
               http://www.we-love-books.org/receipts
            </address>
         </sendTo>
         <sendBy>2000-05-14T08:00:00+08:00</sendBy>
      </deliveryReceiptRequest>
      <commitmentReceiptRequest>
         <sendTo>
            <address xsi:type="agr:duns_number">11-111-1111</address>
         </sendTo>
         <sendBy>2000-05-14T10:00:00+08:00</sendBy>
      </commitmentReceiptRequest>
</services>
```

При наличии заголовка <services> понимание и обработка его самого и всего его содержимого BFC-сервером получателя обязательны для успешной обработки документа BizTalk. В связи с этим элемент <services> должен всегда содержать атрибут **SOAP-ENV:mustUnderstand="1"**.

Тэг <deliveryReceiptRequest> — необязательный элемент, указывающий на необходимость подтверждения поступления и приема документа BizTalk. Подтверждение высылается по адресу, заданному во вложенном элементе <sendTo>. Об обработке документов с тэгом <deliveryReceiptRequest>, структуре и содержании подтверждения см. раздел 8.

Тэг <commitmentReceiptRequest> — необязательный элемент, который указывает на необходимость уведомить отправителя о реше-

нии получателя в отношении обработки полученного документа BizTalk. При наличии элемента <commitmentReceiptRequest> BFC-сервер получателя документа обязан отправить уведомление, сообщающее о принятом решении об обработке документа (как положительном, так и отрицательном). Подтверждение высылается по адресу, заданному во вложенном элементе <sendTo>. Об обработке документов с тэгом <commitmentReceiptRequest>, структуре и содержании уведомления см. раздел 8.

Элементы <deliveryReceiptRequest> и <commitmentReceiptRequest> содержат два обязательных вложенных элемента:

Тэг	Обязательный?	Тип	Тип данных	Вхождение
SendTo	да	элемент	uriReference	Однократное
SendBy	да	элемент	timeInstant	Однократное

Элемент <sendTo> содержит вложенный элемент, указывающий адрес (как правило, в компании-отправителе), по которому должно быть отправлено подтверждение относительно документа BizTalk.

Элемент <sendBy> задает момент, до которого подтверждение должно поступить BFC-серверу отправителя. Неполучение подтверждения до указанного момента времени скорее всего приведет к тому, что документ будет считаться недоставленным и отправитель приступит к выполнению принятых в этом случае процедур.

О достоинствах и недостатках использования абсолютного времени в этом контексте см. примечание к описанию тэга <expiresAt>. О протоколе гарантированной доставки, включая описание семантики срока поступления для каждого типа уведомления, см. раздел 8.

7.4 Каталог документа

Каталог документа описывается необязательным элементом заголовка SOAP, обозначенным бизнес-тэгом <manifest>; он может включать идентификаторы бизнес-документов, вложенных в основной документ BizTalk сообщения BizTalk, так и любых вложений, включая изображения, двоичные файлы и т. п., являющиеся частью сообщения BizTalk (независимо от того, входят ли они физически в состав сообщения). Структура ссылок на вложения, пересылаемые в сообщениях BizTalk, определяется спецификацией MIME; подробно об этом см. раздел 9. Ниже приведен пример заголовка с описанием каталога документа из раздела 9.1:

```
<fst:manifest xmlns:fst="http://schemas.biztalk.org/btf-2-0/manifest"
   SOAP-ENV:mustUnderstand="1">
   <fst:reference>
      <fst:document href="#insurance_claim_document_id">
      <fst:description>Insurance Claim</fst:description>
   </fst:reference>
   <fst:reference>
      <fst:attachment href="CID:claim.tiff@claiming-it.com">
      <fst:description>
         Facsimile of Signed Claim Document
      </fst:description>
   </fst:reference>
   <fst:reference>
      <fst:attachment href="CID:car.jpeg@claiming-it.com">
      <fst:description>Photo of Damaged Car</fst:description>
   </fst:reference>
</fst:manifest>
```

Декларация в основном документе BizTalk сообщения BizTalk. являющегося составным пакетом MIME с несколькими вложениями (см. раздел 9), решает две задачи.

Заголовок <manifest> указывает, что документ является частью составного пакета. В подобных ситуациях заголовок <manifest> обязателен, более того, при этом он должен содержать атрибут mustUnderstand со значением «1».

Заголовок <manifest> позволяет также проверить целостность составного пакета (например, перед отправкой подтверждения), поскольку в этом заголовке перечислены все вложения в объемлющее сообщение BizTalk.

И, наконец, этот заголовок может служить для каталогизации бизнес-документов, передаваемых в объемлющем документе BizTalk, и ссылок на внешние ресурсы (см. ниже).

Элемент <manifest> представляет собой последовательность элементов <reference>:

Тэг	Обязательный?	Тип	Тип данных	Вхождение
reference	да	элемент	complexType	один и более раз

Каждый элемент <reference> содержит вложенные элементы, каждый из которых включает ссылку на компонент сообщения BizTalk и необязательное текстовое описание компонента:

Тэг	Обязательный?	Тип	Тип данных	Вхождение
document/ attachment	ла	элемент	**SOAP reference**	однократное
description	нет	элемент	string	однократное

Фактическая ссылка на компонент содержится в атрибуте **href** элемента SOAP одного из двуч типов: <document> или <attachment>. Значение hrefдолжно представлять собой универсальный идентификатор ресурса, указывающий на ресурс, упомянутый в элементе <reference>. Точнее:

- элемент типа <document> должен содержать атрибут **href**, значение которого представляет собой идентификатор фрагмента вида #id, указывающий на бизнес-документ в теле объемлющего документа BizTalk;
- элемент типа <attachment> должен содержать атрибут **href**, значение которого представляет собой универсальный идентификатор ресурса за пределами документа BizTalk; в этом качестве могут выступать ресурсы двух типов:
  - вложения в объемлющее сообщение BizTalk;
  - внешние ресурсы — файлы большого размера, рассматриваемые как вложения, однако не пересылаемые в сообщении BizTalk.

Необязательный элемент <description> содержит текстовое описание бизнес-документа или вложения. Его содержание имеет смешанный тип (текст с XML-разметкой или без нее). Этот элемент может рассматриваться как сопроводительный комментарий или служить для дополнительной обработки или протоколирования.

7.5 Управление обработкой документа

Информация об управлении обработкой документа содержится в необязательном элементе заголовка SOAP, обозначенном бизнес-тэгом <process>, который описывает бизнес-процесс, образующий контекст обработки документа BizTalk. Вот пример такого элемента:

```
<prc:process SOAP-ENV:mustUnderstand="1"
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:dtl="http://www.trading-agreements.org/process/detail/"
   xmlns:eps="http://schemas.biztalk.org/btf-2-0/endpoints"
   xmlns:prc="http://schemas.biztalk.org/btf-2-0/process">
   <prc:type>purchasing:Book_Purchase_Process</prc:type>
   <prc:instance>purchasing:Book_Purchase_Process#12345</prc:instance>
```

```
    <prc:detail>
        <dtl:targetPort>po_request</dtl:targetPort>
        <dtl:exceptionAddress>
            <eps:address xsi:type="dtl:escrow">common_escrow_provider
            </eps:address>
        </dtl:exceptionAddress>
    </prc:detail>
</prc:process>
```

При наличии заголовка <process> понимание и обработка его самого и всего его содержимого BFC-сервером получателя обязательны для успешной обработки документа BizTalk. В связи с этим элемент <process> должен всегда содержать атрибут **SOAP-ENV:mustUnderstand=»1"**. В таблице перечислены бизнес-тэги, используемые для создания вложенных элементов элемента заголовка <process>, и их свойства:

Тэг	Обязательный?	Тип	Типданных	Вхождение
Type	да	элемент	UriReference	однократное
Instance	да	элемент	UriReference	однократное
detail	нет	элемент	AnyType	однократное

Элемент <type> содержит универсальный идентификатор, означающий тип бизнес-процесса — например, процесс покупки книги. Под бизнес-процессом понимается последовательность обмена (как правило, несколькими) документами BizTalk, согласованная партнерами. Эта последовательность задает «правила игры» и обычно многократно повторяется.

Элемент <instance> содержит универсальный идентификатор, однозначно идентифицирующий конкретный экземпляр бизнес-процесса, с которым ассоциирован данный документ BizTalk Document — например, процесс покупки обществом анонимных библиофилов книги «Основы BizTalk». Этот элемент нужен для связывания документа с конкретным экземпляром бизнес-процесса в (типичном) случае параллельного выполнения нескольких экземпляров этого процесса. Обычный способ конструирования значения этого элемента — дополнение универсального идентификатора типа процесса (см. выше) идентификатором фрагмента, описывающим экземпляр (часто в роли такого идентификатора, как в приведенном выше примере, выступает порядковый номер).

Элемент <detail> предназначен для хранения дополнительных данных, которые могут понадобиться для идентификации конкретного этапа или точки входа бизнес-процесса или дополнительных данных, например, о местонахождении обрабатывающих приложении или указаний по обработке ошибок. Содержание этого элемента определяется конкретной реализацией.

8. Гарантированная доставка документов BizTalk

Инфраструктура BizTalk упрощает асинхронный обмен документами — важнейшую составную часть решений в области электронной коммерции и интеграции корпоративных приложений. Обязательными условия решения этих задач — гарантированная доставка, обнаружение ошибок и развитые средства отчетности. Оптимальный метол развертывания подобных решений — привлечение высокопроизводительного современного ПО промежуточного слоя. Однако широкий спектр возможных сценариев развертывания интеграционных решений на базе инфраструктуры BizTalk и использование транспортных протоколов с невысоким уровнем обслуживания делают актуальной разработку стандартного решения для гарантированной доставки документов BizTalk, которое допускает простую реализацию. Отметим также, что стандартные бизнес-процессы требуют не только уведомления о доставке и физическом приеме сообщения, но и отдельного подтверждения верификации содержимого сообщения и намерения выполнить соответствующие бизнес-транзакции.

В данном разделе описано решение этих проблем на основе двух простых принципов:

- использование подтверждений;
- идемпотентная доставка.

Наша цель — формализовать процедуру доставки документа BizTalk, его приема и намерения выполнить затребованные операции. Ниже перечислены основные идеи нашего подхода.

- Подтверждения и уведомления позволяют BFC-серверу компании-отправителя быть уверенным, что документ BizTalk получен и принят (уведомление о доставке), его содержание проверено и принято к обработке (подтверждение). Идентификация документа осуществляется с помощью тэга <identity>.
- Учитываявозможность многократной повторной передачи документа BizTalk, BFC-сервер получателя должен иметь возможность применения правил идемпотентной доставки для выявления и удаления дубликатов документа, идентифицированного тэгом <identity>.

- Если BFC-сервер компании-отправителя за время, указанное тэгом <sendBy> элемента <deliveryReceiptRequest>, не получил уведомления о доставке, сервер должен создать сообщение о невозможности доставки и принять меры по исправлению ситуации.
- Если за время, указанное тэгом <sendBy> элемента<commitment-ReceiptRequest>, BFC-сервер компании-отправителя не получил подтверждение-обязательство, он должен трактовать этот факт как поступление отказа в обработке без описания причин.
- Существует малая, но ненулевая вероятность того, что документ будет доставлен компании-получателю, проверен и обработан, а все соответствующие уведомления утрачены по причине сбоев транспортного протокола, что приведет к генерации сервером-отправителем сообщений о невозможности доставки и/или об отказе в обработке документа.

Оставшаяся часть этого раздела посвящена описанию структуры подтверждений и типичного поведения отправителя и получателя в контексте гарантированной доставки документов BizTalk. Естественно, детализация поведения отправителя и получателя определяется реализацией; приведенное ниже описание следует рассматривать как рекомендации общего характера.

8.1 Структура и содержание уведомлений

8.1.1 Уведомление о доставке

В терминологии инфраструктуры BizTalk уведомление о доставке — это документ BizTalk с дополнительным элементом заголовка <deiiveryReceipt> и пустым телом; пример такого документа приведен ниже. Как и у любого BizTalk-документа, у уведомления есть идентификатор и срок действия. Время окончания срока действия уведомления должно совпадать с конечным сроком доставки, заданным в тэге <sendBy> элемента <deliveryReceiptRequest> заголовка <services> документа, доставку которого подтверждает данное уведомление.

Содержание уведомления однозначно идентифицирует документ BizTalk, доставку которого оно подтверждает, с помощью уникального идентификатора <identity> документа в специальном элемента заголовка < delivery Receipt > , наличие которого — отличительный признак уведомления. Кроме того, в этом элементе указывается время поступления документа к получателю.

Ниже приведено уведомление о получении документа из примера в разделе 7, необходимость уведомления о поступлении которого описана элементом <deliveryReceiptRequest> примера в разделе 7.3.

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
   xmlns:xsd="http://www.w3.org/2000/08/XMLSchema"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
   <SOAP-ENV:Header>
      <endpoints SOAP-ENV:mustUnderstand="1"
         xmlns:ta="http://schemas.trading-agreements.com/"
         xmlns="http://schemas.biztalk.org/btf-2-0/endpoints">
         <to>
            <address xsi:type="ta:httpURL">
               http://www.we-love-books.org/receipts
            </address>
         </to>
         <from>
            <address xsi:type="ta:department">Book Orders</address>
         </from>
      </endpoints>
      <properties SOAP-ENV:mustUnderstand="1"
         xmlns="http://schemas.biztalk.org/btf-2-0/properties">
         <identity>uuid:24d304a0-b6e1-493a-b457-4b86c684d6f3</identity>
         <sentAt>2000-05-13T10:34:00-08:00</sentAt>
         <expiresAt>2000-05-14T08:00:00+08:00</expiresAt>
         <!- expiration is at delivery deadline ->
         <topic>http://electrocommerce.org/delivery_receipt/</topic>
      </properties>
      <deliveryReceipt xmlns="http://schemas.biztalk.org/btf-2-0/
      receipts"
         SOAP-ENV:mustUnderstand="1">
         <receivedAt>2000-05-13T10:04:00-08:00</receivedAt>
         <identity>uuid:74b9f5d0-33fb-4a81-b02b-5b760641c1d6</identity>
         <!-the above is the identity of the original message ->
      </deliveryReceipt>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body/>
   <!- the body is always empty in a delivery receipt ->
</SOAP-ENV:Envelope>
```

Понимание и обработка заголовка <deliveryReceipt> и всего его содержимого BFC-сервером получателя обязательны для успешной обработки уведомления. В связи с этим элемент <deliveryReceipt>

должен всегда содержать атрибут SOAP-ENV:mustUnderstand=»1". **В** таблице перечислены бизнес-тэги, предназначенные для создания дочерних элементов заголовка <deliveryReceipt>, и их свойства.

Тэг	Обязательный?	Тип	Тип данных	Вхождение
receivedAt	да	элемент	TimeInstant	Однократное
identity	да	элемент	UriReference	Однократное

Элемент <receivedAt> указывает время поступления документа, доставку которого подтверждает уведомление. В случае поступления и принятия нескольких копий документа BizTalk (определение термина «принятие» см. ниже) отметка может указывать время поступления первой копии или той копии, поступление которой подтверждает уведомление.

Элемент <identity> идентифицирует документ, поступление которого подтверждается.

8.1.2 Подтверждение

В терминологии инфраструктуры BizTalk подтверждение — это документ BizTalk с дополнительным элементом заголовка <commitmentReceipt> и пустым телом; пример такого документа приведен ниже. Как и у любого другого документа BizTalk, у подтверждения есть идентификатор и срок действия. Окончание срока действия этого документа должно совпадать с предельным сроком поступления подтверждения, заданным в тэге <sendBy> элемента <commitmentReceiptRequest> заголовка <services> документа, обязательство об обработке которого подтверждается.

Содержание подтверждения однозначно идентифицирует документ BizTalk, обязательство об обработке которого оно подтверждает, используя уникальный идентификатор <identity> документа в специальном элемента заголовка <commitmentReceipt>, наличие которого — отличительный признак подтверждения. Кроме того, в этом элементе указывается время принятия решения об обработке документа.

Ниже приведено подтверждение для документа из примера в разделе 7 и элемента <commitmentReceiptRequest> из примера в разделе 7.3.

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope"
   xmlns:xsd="http://www.w3.org/2000/08/XMLSchema"
   xmlns:xsi="http://www.w3 org/1999/XMLSchema-instance">
   <SOAP-ENV:Header>
```

```
        <endpoints SOAP-ENV:mustUnderstand="1"
        xmlns="http://schemas.biztalk.org/btf-2-0/endpoints"
        xmlns:ta="http://schemas.trading-agreements.com/">
            <to>
                <address xsi:type="ta:duns_number">11-111-1111</address>
            </to>
            <from>
                <address xsi:type="ta:department">Book Orders</address>
            </from>
        </endpoints>
        <properties SOAP-ENV:mustUnderstand="1"
        xmlns="http://schemas.biztalk.org/btf-2-0/properties">
            <identity>uuid:1d394ac1-cadf-47cf-9a1e-aaa40531b97d</identity>
            <sentAt>2000-05-13T10:55:00-08:00</sentAt>
            <expiresAt>2000-05-14T10:00:00+08:00</expiresAt>
            <!- expiration is at commitment deadline ->
            <topic>http://electrocommerce.org/commitment_receipt/</topic>
        </properties>
        <commitmentReceipt xmlns="http://schemas.biztalk.org/btf-2-0/
        receipts"
        xmlns:cmt="http://schemas.electrocommerce.org/commitment/"
        SOAP-ENV:mustUnderstand="1">
            <decidedAt>2000-05-13T10:44:00-08:00</decidedAt>
            <decision>negative</decision>
            <identity>uuid:74b9f5d0-33fb-4a81-b02b-5b760641c1d6</identity>
            <!- the above is the identity of the original message ->
            <commitmentCode>cmt:outOfStock</commitmentCode>
            <commitmentDetail>
            <cmt:restockExpectedOn>2000-06-15</cmt:restockExpectedOn>
            </commitmentDetail>
        </commitmentReceipt>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body/>
</SOAP-ENV:Envelope>
```

Понимание и обработка заголовка <commitmentReceipt> и всего его содержимого BFC-сервером получателя обязательны для успешной обработки подтверждения. В связи с этим элемент <commitmentReceipt> должен всегда содержать атрибут **SOAP-ENV:mustUnderstand=»1"**. **В таб-**

лице перечислены бизнес-тэги, предназначенные для создания дочерних элементов заголовка <commitmentReceipt>, и их свойства.

Тэг	Обязательный?	Тип	Тип данных	Вхождение
decidedAt	да	элемент	TimeInstant	однократное
decision	да	элемент	positive/negative	однократное
identity	да	элемент	uriReference	однократное
commitmentCode	нет	элемент	Qname	однократное
commitmentDetail	нет	элемент	AnyType	однократное

Элемент <decidedAt> указывает время принятия решения об обработке документа, о котором идет речь в подтверждении,

Элемент <decision> содержит собственно решение — положительное или отрицательное.

Элемент <identity> идентифицирует документ, об обработке которого идет речь в подтверждении.

Необязательный элемент <commitmentCode> содержит допустимое (в смысле XSD [*XSD1*]) имя, детализирующее решение об обработке документа. Этот элемент особенно полезен в случае отрицательного решения, когда он играет роль кода ошибки.

Необязательный элемент <commitmentDetail> содержит дополнительный комментарий к решению об обработке.

Структура содержимого элементов <commitmentCode> и <commitmentDetail> определяется приложением. Элемент <SOAP-ENV:Body> документа-подтверждения может дополнительно содержать бизнес-документ(ы), ссылки на которые присутствуют в содержимом элемента <commitmentDetail>. Структура и интерпретация этих документов определяется приложением.

8.2 Поведение отправителя и получателя

Данный раздел содержит рекомендации по реализации поведения BFC-серверов отправителя и получателя в контексте гарантированной доставки документов и сообщений BizTalk на базе механизма уведомлений. Настоящая спецификация описывает лишь поведение на уровне сообщений. Логика и механизмы повторной передачи документов, удаления дубликатов, хранения и архивирования бизнес-документов выбираются при реализации и не влияют на соответствие настоящейспецификации.

В настоящем разделе не рассматривается и поведение отправителя и получателя относительно подтверждений об обработке бизнес-документов, поскольку семантика таких подтверждений, их генерация и обработка существенно зависят от специфики конкретной реализации. Механизм подтверждений, предложенный в настоящей спецификации, относится лишь к сообщениям и образует формальную основу для дальнейшего уточнения этих аспектов при конкретной реализации BFC-сервера.

8.2.1 Поведение отправителя

Согласно общепринятой практике предполагается, что переданный документ BizTalk сохранен в архиве компании-отправителя. Документы, требующие гарантированной доставки с предсказуемым поведением, настоятельно рекомендуется копировать в архив еще до отправки.

Для гарантированной доставки документа BizTalk от отправителя требуется лишь добавить соответствующий элемент <deliveryReceiptRequest> в элемент заголовка <services>, а также корректно идентифицировать и обрабатывать как поступающие уведомления, так и их отсутствие. Ниже даны эвристические рекомендации по повторной отправке документов, которые, по нашему мнению, должны повысить вероятность успешной доставки в условиях ненадежного транспорта. Единственное обязательное требование в контексте повторной отправки — неизменность содержимого документа BizTalk, включая элементы заголовка. В частности, содержимое свойства <sentAt> должно оставаться неизменным и указывать время первой попытки отправки документа.

Поведение отправителя при повторной передаче обычно определяется значением базового параметра — интервала между повторными попытками. Этот параметр может иметь фиксированное значение или устанавливаться по отдельности для каждого бизнес-партнера или даже для каждого транспортного протокола, используемого в рамках одного соглашения. Кроме того, обычно используется еще один параметр — максимальное число повторных попыток отправки документа.

Стандартный подход к повторной передаче очень прост: отправка документа получателю повторяется до тех пор, пока

- не будет получено уведомление о доставке или
- не истечет время уведомления, указанное в элементе <sendBy> заголовка <deliveryReceiptRequest> или
- не будет достигнуто максимально допустимое число повторных попыток.

Если по окончании этой процедуры время, указанное в элементе <sendBy> заголовка <deliveryReceiptRequest>, истекло, а уведомление не получено, отправка считается неудачной. Обычное поведение в этой ситуации включает отправку соответствующего сообщения приложению-автору документа (скажем, перемещением копии документа в очередь неотправленных сообщений), но конкретные детали этого поведения остаются на усмотрение авторов конкретной реализации.

Повторные попытки отправки не обязаны выполняться по тому же транспортному протоколу, что и первая. Если передача документа получателю, указанному в адресе <to> элемента <endpoints>, допускает применение нескольких транспортных протоколов, повторные попытки отправки могут осуществляться по любому из них.

Конкретная реализация может использовать более сложные алгоритмы повторной передачи для достижения нужной масштабируемости и эффективности и для учета особенностей контекста реализации (например, возможного разрыва соединения при передаче сообщения).

8.2.2 Поведение получателя

Согласно общепринятой практике предполагается, что принятый документ BizTalk сразу же скопирован в архив компании-получателя. Документы, требующие гарантированной доставки с предсказуемым поведением, настоятельно рекомендуется копировать в архив до отправки уведомления. Как указано ниже, стратегия идемпотентной доставки требует также максимально быстрой архивации информации.

Для обеспечения гарантированной доставки документа BizTalk от получателя требуется лишь уведомлять отправителя о принятии каждого документа, заголовок <services> которого содержит элемент <deliveryReceiptRequest>. Уведомление должно отправляться по адресу, заданному во вложенном элементе <sendTo>. Согласно определению (см. раздел 4.1) «принятие документа» в данном случае означает, что принимающая сторона распознала документ как предназначенный для получателя, которого она обслуживает. Это определение также распространяется на копии и дубликаты ранее полученных документов; вывод о том, что полученный документ — копия или дубликат, делается на основании значения тэга <identity>. Документы, полученные позже момента, заданного в тэге <expiresAt>, не должны приниматься, а отправитель не должен уведомляться об их поступлении. Документы, поступившие позже времени, указанного в тэге <sendBy> элемента <deliveryReceiptRequest>, но до заданного тэгом <expiresAt>, *должны приниматься с* уведомлением об их поступлении.

Структура уведомления и взаимное отождествление документа и уведомления описаны выше.

Кроме того, получатель может реализовать идемпотентную доставку документов BizTalk приложению, которое он обслуживает. Такое поведение может быть как составной частью бизнес-процедуры, согласованной участниками документооборота, так и (постоянной или настраиваемой для каждого соглашения) составляющей внутренней политики компании-получателя. Идемпотентность доставки требует, чтобы любой документ BizTalk, принятый компанией-получателем, был доставлен приложению-потребителю один и только один раз независимо от того, сколько копий документа поступило от отправителя.

Рекомендуемая стратегия удаления дубликатов (при необходимости идемпотентной доставки) заключается в хранении всех принятых документов BizTalk в архиве по крайней мере до истечения срока их действия (т. е, до наступления момента, заданного в тэге <expiresAt>). Фактически достаточно хранить лишь (уникальный) идентификатор каждого документа. Удаление дубликатов имеет смысл только после принятия (а не получения) документа.

8.2.3 Сроки доставки, уведомления и подтверждения

Необходимо понимать различия между тремя моментами времени, которые могут наличествовать в заголовке документа BizTalk. Тэг <sendBy> элемента <deliveryReceiptRequest> задает *крайний срок доставки,* тэг <sendBy> элемента <commitmentReceiptRequest> — *крайний срок подтверждения,* а свойство <expiresAt> — *крайний срок обработки.* Последний обязателен для любого документа BizTalk; остальных может не быть. Необходимость связать нескольких сроков с одним документом обусловлена тем, что и для подтверждения, и для принятия решения об обработке документа может понадобиться время. Если, например, подтверждение к выполнение операции, связанной с документом BizTalk, занимает 4 часа, и она должна быть завершена не позднее, чем через 12 часов после передачи документа, крайний срок доставки этого документа должен составлять 8, а не 12 часов. Гибкость такого рода нужна для поддержки бизнес-процессов.

Семантика крайних сроков формулируется следующим образом,

- Крайний срок доставки относится к *приему* документа BizTalk получателем. Определение термина «принятие» см. в разделе 4.1. Уведомление (в форме уведомления о доставке) о принятии должно быть получено отправителем не позднее крайнего срока доставки.
- Крайний срок подтверждения относится к *проверке содержания документа* и *намерению обработать его.* Семантика подтверждения оставлена на усмотрение авторов конкретной реализации. Под-

тверждение намерения получателя обработать документ должно быть получено отправителем не позднее крайнего срока подтверждения.

- Крайний срок обработки задает момент, по наступлении которого документ BizTalk считается недействительным. Такие документы не подлежат ни обработке, ни уведомлению.

Поясним, что происходит по наступлении каждого из перечисленных выше крайних сроков.

- По истечении срока доставки происходит следующее.
  - Если не получено уведомление, BFC-сервер отправителя должен сообщить об этом приложению — автору документа, извешая его таким образом о том, что документ либо не поступил к получателю, либо получателю не хватило времени, чтобы обработать его.
  - Для BFC-сервера получателя это обстоятельство не имеет значения — он продолжает получать такие документы и уведомлять отправителей об их поступлении. Это значит, что уведомления о доставке могут отправляться и после наступления крайнего срока доставки; фактически получателя крайний срок доставки не интересует. С другой стороны, если получатель не знал бы, к какому сроку отправитель должен получить уведомление, он мог бы задержать отправку уведомления (скажем, в целях оптимизации работы сервера) и тем самым поставить отправителя в затруднительное положение. Следовательно, крайний срок доставки — своего рода совет получателю относительно приоритета отправки уведомления, особенно полезный при работе с ненадежными соединениями.
- По истечении срока подтверждения происходит следующее.
  - BFC-сервер отправителя должен трактовать это как поступление подтверждения с отказом без разъяснения причин (тэги <commitmentCode> и <commitmentDetail> пусты). Такой подход позволяет уведомить приложение-отправитель о том, что приложение-получатель решило не обрабатывать документ или на это не хватило времени.
  - Поведение BFC-сервера получателя аналогично случаю истечения срока доставки.
- По истечении срока обработки происходит следующее.
  - К поведению BFC-сервера отправителя требований не предъявляется; его поведение полностью определяется особенностями конкретной реализации.

- BFC-сервер получателя должен отвергать все документы, поступающие по истечении срока обработки. Отправитель не должен уведомляться о поступлении таких документов; эти документы ни в коем случае не должны передаваться приложению-получателю для обработки.

9. Документы BizTalk с вложениями

В рамках бизнес-процессов часто возникает необходимость пересылки документов с различными вложениями: от факсимильных копии юридических документов до технических чертежей. Формат файлов вложений часто бывает двоичным. В настоящем разделе описаны:

- стандартный метод упаковки документа BizTalk с одним или несколькими вложениями в составной документ MIME для передачи;
- связь структуры вложений MIME с элементом заголовка<manifest> основного документа BizTalk.

Передать MIME-сообщения можно по большинству транспортных протоколов Интернета; особенности передачи по HTTP см. в разделе 11.1.

9.1 Структура составного документа MIME

Составное сообщение BizTalk. включающее основной документ BizTalk и одно или несколько вложений, должно пересылаться в составной структуре MIME, отвечающей правилам для носителей типа MIME multipart/related [см. *MULTIPART*].

Ниже приведен пример документа BizTalk с двумя вложениями, составляющий страховое требование по автомобильной страховке. Основной документ BizTalk описывает требование и пересылается вместе с факсимильной копией подписанного требования (Claim.tiff) и цифровой фотографией поврежденного автомобиля (Car.jpeg).

```
MIME-Version: 1.0
Content-Type: Multipart/Related;
              boundary=biztalk_2_0_related_boundary_example;
              type=text/xml;
              start="<claim.xml@claiming-it.com>"
Content-Description: This is the optional message description.
-biztalk_2_0_related_boundary_example
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <claim.xml@claiming-it.com>
<?xml version='1.0' ?>
```

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header>
      <!- endpoints and properties header entries omitted for brevity ->
   <manifest xmlns="http://schemas.biztalk.org/btf-2-0/manifest"
      mustUnderstand="1">
         <reference>
            <document href="#insurance_claim_document_id"/>
            <description>Insurance Claim</description>
         </reference>
         <reference>
            <attachment href="CID:claim.tiff@claiming-it.com"/>
            <description>Facsimile of Signed Claim
            Document</description>
         </reference>
         <reference>
            <attachment href="CID:car.jpeg@claiming-it.com"/>
            <description>Photo of Damaged Car</description>
         </reference>
      </manifest>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <claim:Insurance_Claim_Auto id="insurance_claim_document_id"
         xmlns:claim="http://schemas.risky-stuff.com/Auto-Claim">
         <!- ...claim details... ->
      </claim:Insurance_Claim_Auto>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
-biztalk_2_0_related_boundary_example
Content-Type: image/tiff
Content-Transfer-Encoding: base64
Content-ID: <claim.tiff@claiming-it.com>
...Файл TIFF в кодировке Base64...
-biztalk_2_0_related_boundary_example
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <car.jpeg@claiming-it.com>
...Файл JPEG...
-biztalk_2_0_related_boundary_example-
```

Структура описывающих вложения элементов <attachment> элемента заголовка <manifest> основного документа BizTalk детально рассмотрена в следующем разделе. Описание носителя типа multipart/related см. в [*MULTIPART*].Кроме того, основной документ BizTalk должен пересылаться в корневом разделе составного документа MIME. Каждая часть составного документа, включая корневую, должна быть помечена заголовком MIME Content-ID согласно правилам MIME [*MIME1*].

Применение описанного в настоящем разделе составного контейнера разрешено, даже когда вложений нет. Этот вырожденный случай полезен для изоляции содержания сообщения от транспортного протокола (см. раздел 10). Кроме того, такой подход может быть полезен для однозначного отделения содержания сообщения BizTalk от прочих составляющих (например, на случай отказа получателя принять сообщение). В вырожденном случае элемент заголовка <tanifest> в основном документе BizTalk необязателен.

9.2 Структура декларации

Структура элементов <reference> заголовка <manifest>, описывающих вложения MIME, довольно проста. Каждый элемент <reference> содержит вложенный элемент <attachment> с атрибутом **href,** значение которого задает местонахождение вложения. Значение атрибута имеет формат указателя Content-ID и создается на основе идентификатора Content-ID раздела составного документа MIME, содержащего вложение, Структура идентификаторов Content-ID описана в [*CID*].

10. Защита документов и сообщений BizTalk

Бизнес-процессы нуждаются в защите сообщений в целях аутентификации, гарантии целостности и авторства и конфиденциальности. Средства защиты транспортного уровня, такие как протокол Secure Socket Layer (SSL), достаточны для защиты и аутентификации прямых соединений, но не поддерживают подпись и шифрование отдельных сообщений и их составных частей, что необходимо в любом реальном сценарии электронного бизнеса. Инфраструктура BizTalk поддерживает версию 3 стандарта S/MIME [*SMIME*],обеспечивающего адекватную защиту сообщений BizTalk и их составных частей.

Защитить можно документ BizTalk, вложение или составной контейнер MIME (структурированный согласно описанию из раздела 9) целиком. Существует три метода защиты: помещение в защищенный

конверт (шифрование), только подпись или шифрование и подпись. Они применяются согласно правилам, описанным в пунктах 3.3, 3.4 и 3.5 [*SMIME*].В рамках BizTalk поддерживается только подпись, пересылаемая отдельно (как вложение в составном контейнере); подробное описание см. в разделе 3.4.3 документа [*SMIME*].

Сочетание вложений, поддержки S/MIME и наличия атрибута SOAP **href** позволяет по-разному применять шифрование к элементам заголовка и бизнес-документам, пересылаемым в сообщении BizTalk. Это часто необходимо, поскольку элементы заголовка (обычно) пересылаются незашифрованными, чтобы упростить их обработку компонентами-посредниками, тогда как бизнес-документы шифруются открытым ключом, причем соответствующий закрытый ключ недоступен посредникам. Перечисленные выше методы позволяют разместить зашифрованные бизнес-документы во вложениях и сослаться на них в теле документа BizTalk Document так же, как это делается в декларации. После этого документ BizTalk можно пересылать в незашифрованном виде, не поступаясь безопасностью бизнес-документов. Ниже показано, как передать страховое требование из примера в разделе 9, если требуется зашифровать основной XML-документ. Ссылка на зашифрованное требование в формате XML в теле документа BizTalk использует стандартный формат ссылок SOAP. Этот механизм позволяет придерживаться единой формы документа BizTalk независимо от того, как пересылается документ-требование: в теле документа BizTalk или во вложении. При таком подходе допустимо и пустое тело сообщения.

```
MIME-Version: 1.0
Content-Type: Multipart/Related:
boundary=biztalk_2_0_related_boundary_example;
type=text/xml;
start="<claim.xml@claiming-it.com>"
Content-Description: Необязательное описание сообщения.
-biztalk_2_0_related_boundary_example
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <claim.xml@claiming-it.com>
<?xml version='1.0' ?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
   <SOAP-ENV:Header>
   <!- заголовки опущены для краткости ->
```

```
      <manifest xmlns="http://schemas.biztalk.org/btf-2-0/manifest"
         mustUnderstand="1">
         <reference>
            <attachment href="CID:insurance_claim@claiming-it.com">
            <description>Insurance Claim</description>
         </reference>
         <reference>
            <attachment href="CID:claim.tiff@claiming-it.com">
            <description>Facsimile of Signed Claim
            Document</description>
         </reference>
         <reference>
            <attachment href="CID:car.jpeg@claiming-it.com">
            <description>Photo of Damaged Car</description>
         </reference>
      </manifest>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <claim:Insurance_Claim_Auto
      href="CID:insurance_claim@claiming-it.com"
      xmlns:claim="http://schemas.risky-stuff.com/Auto-Claim">
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
-biztalk_2_0_related_boundary_example
Content-Type: application/pkcs7-mime; smime-type=enveloped-data;
name=smime.p7m
Content-Transfer-Encoding: base64
Content-Description: encrypted claim document
Content-Disposition: attachment; filename=smime.p7m
Content-ID:<insurance_claim@claiming-it.com>
rfvbnj756tbBghyHhHUujhJhjH77n8HHGT9HG4VQpfyF467GhIGfHfYT6
7n8HHGghyHhHUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H
f8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
0GhIGfHfQbnj756YT64V
-biztalk_2_0_related_boundary_example
Content-Type: image/tiff
Content-Transfer-Encoding: base64
Content-ID: <claim.tiff@claiming-it.com>
...Файл TIFF в кодировке Base64...
```

```
-biztalk_2_0_related_boundary_example
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <car.jpeg@claiming-it.com>
...Файл JPEG...
-biztalk_2_0_related_boundary_example-
```

10.1 S/MIME

S/MIME позволяет защитить два вида документов инфраструктуры BizTalk:

- документ BizTalk, который может содержать вложения, — тогда документ шифруется средствами S/MIME отдельно от вложений и пересылается в зашифрованном виде в корневой части составного контейнера;
- весь составной контейнер, состоящий из основного документа BizTalk, и вложение(я), причем в этом случае основной документ BizTalk имеет формат text/xml и не зашишен средствами S/MIME.

Поскольку в качестве вложения в документ BizTalk может выступать что угодно, настоящая спецификация не накладывает ограничений на средства кодирования и шифрования вложений.

В обоих случаях спецификация BizTalk требует, чтобы защищенный компонент имел тип multipart/related или был «завернут» в конверт этого типа. При раздельной защите документа BizTalk и его вложений это достигается само собой. В противном случае — при защите средствами S/MIME документа BizTalk без вложений или пакета, состоящего из документа BizTalk и вложений, — результирующий контейнер имеет другой тип S/MIME, и его надо «завернуть» в вырожденный (т. е. содержащий только корневой компонент) контейнер типа multipart/related. Это ограничение вызвано необходимостью ограничить число возможных типов содержания сообщений BizTalk двумя: text/xml и multipart/related, что упрощает семантику и гарантирует совместимость с протоколами HTTP и SMTP. Прямая передача пакетов S/MIME по SMTP может привести к выполнению обратного преобразования почтовыми агентами. Кроме того, пакеты типа application/pkcs7-mime (используемые при передаче сообщений, зашифрованных средствами S/MIME) содержат заголовки (в частности, content-disposition и content-transfer-encoding), несовместимые с HTTP.

Ниже показан документ BizTalk из раздела 5, подписанный цифровой подписью и завернутый в вырожденный транспортный контейнер типа multipart/related.

```
MIME-Version: 1.0
Content-Type: Multipart/Related;
boundary=biztalk_2_0_related_boundary_example;
type=multipart/signed
-biztalk_2_0_related_boundary_example
Content-Type: Multipart/Signed;
protocol="application/pkcs7-signature";
micalg=sha1; boundary=smime-boundary-1234
-smime-boundary-1234
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
   <SOAP-ENV:Header>
      <eps:endpoints SOAP-ENV:mustUnderstand="1"
         xmlns:eps="http://schemas.biztalk.org/btf-2-0/endpoints"
         xmlns:agr="http://www.trading-agreements.org/types/">
         <eps:to>
            <eps:address xsi:type="agr:department">Book Orders</
            eps:address>
         </eps:to>
         <eps:from>
            <eps:address xsi:type="agr:organization">Book Lovers</
            eps:address>
         </eps:from>
      </eps:endpoints>
      <prop:properties SOAP-ENV:mustUnderstand="1"
         xmlns:prop="http://schemas.biztalk.org/btf-2-0/properties">
         <prop:identity>uuid:74b9f5d0-33fb-4a81-b02b-5b760641c1d6</
         prop:identity>
            <prop:sentAt>2000-05-14T03:00:00+08:00</prop:sentAt>
            <prop:expiresAt>2000-05-15T04:00:00+08:00</prop:expiresAt>
            <prop:topic>http://electrocommerce.org/purchase_order/</
            prop:topic>
      </prop:properties>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <po:PurchaseOrder xmlns:po="http://electrocommerce.org/
      purchase_order/">
         <po:Title>Essential BizTalk</po:Title>
         </po:PurchaseOrder>
```

```
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
-smime-boundary-1234
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s
...цифровая подпись (в кодировке base64)...
-smime-boundary-1234-
-biztalk_2_0_related_boundary_example-
```

11. Транспортировка

11.1 HIP

В этом разделе описана пересылка документов BizTalk (с вложениями или без) по протоколу HTTP.

HTTP — протокол типа «запрос — ответ», тогда как архитектура BizTalk построена на асинхронной передаче сообщений. Ресурс HTTP, выступающий как адресат сообщения BizTalk, должен рассматриваться только как агент передачи сообщений; ни в коем случае не следует предполагать, что это SOAP-процессор (см. спецификацию SOAP 1.1) или BFC-сервер, описанный в настоящей спецификации. Таким образом, содержание HTTP-ответа отражает только результат попытки передачи сообщения агенту. В частности, успешное завершение (код 2xx) не означает, что сообщение и соответствующий основной документ BizTalk принят получателем в смысле раздела 8.2.2 и не гарантирует проверки целостности и корректности конверта SOAP. Причина в том, что передающий агент и получатель документа — чаще всего не одно и то же; так. роль агента может играть HTTP-сервер. пересылающий сообщения получателю, ставя их в очередь.

Описанная в этом разделе транспортировка сообщений по HTTP — частный случай транспортировки сообщений SOAP 1.1 по HTTP, но данное описание расширяет эту спецификацию в части использования составных сообщений MIME для пересылки вложений в сообщениях SOAP[1]. Все принципы передачи сообщений SOAP 1.1 по HTTP применимы к пересылке обычных документов BizTalk (представляющих собой сообщения SOAP) по протоколу HTTP. Описанная ниже структура сообщений для документов BizTalk с вложениями отличает-

[1] В декабре 2000 г. опубликовано дополнение к спецификации SOAP 1.1 (*http://www.w3.org/TR/2000/NOTE-SOAP-attachments-20001211*) описывающее передачу сообщений SOAP с вложениями. — *Прим.ред.*

ся от спецификации SOAP 1.1. Описанная здесь HTTP-транспортировка не использует ни удаленные вызовы SOAP RPC, ни синхронную передачу по схеме «запрос — ответ». В частности, уведомления о доставке, являющиеся составной частью описанного в разделе 8 механизма надежной доставки, всегда являются независимыми сообщениями, пересылаемыми как HTTP-запросы. Они никогда не передаются в составе HTTP-отклика на HTTP-запрос, несущий соответствующий документ BizTalk.

Правила в отношении кодов статуса HTTP в HTTP-ответах применяются независимо от наличия вложений. В частности, в общем (асинхронном) случае при возвращении HTTP-отклика до обработки документа следует вернуть код 202 **accepted.** Допустимо (и рекомендуется) не отправлять ответ до окончания обработки документа; в этом случае код статуса HTTP-отклика согласно принципам SOAP 1.1 будет нести больше информации о результатах. При таком подходе, в частности, код 200 может означать, что получатель (или приложение, которому получатель передал сообщение для обработки) полностью проанализировал документ, проверил корректность обязательных заголовков и выполнил операции, затребованные в документе. Иначе говоря, этот код будет свидетельствовать об обработке сообщения BizTalk.

11.1.1 Пример BizTalk-сообщения

Ниже показана пересылка документа BizTalk из первого примера как сообщения HTTP командой POST.

```
POST /bookPurchase HTTP/1 Л
Host:www.we-have-books.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "http://electrocommerce.org/purchase_order/"
<?xml version='1.0' ?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
   <SOAP-ENV:Header>
      <eps:endpoints SOAP-ENV:mustUnderstand="1"
         xmlns:eps="http://schemas.biztalk.org/btf-2-0/endpoints"
         xmlns:agr="http://www.trading-agreements.org/types/">
         <eps:to>
            <eps:address xsi:type="agr:department">
```

```
            Book Order Department
            </eps:address>
        </eps;to>
        <eps; from>
            <eps:address xsi:type="agr:organization">
                Booklovers Anonymous
            </eps:address>
        </eps:from>
    </eps:endpoints>
    <prop:properties SOAP-ENV:mustUnderstand="1"
        xmlns:prop="http://schemas.biztalk.org/btf-2-0/properties">
        <prop:identity>uuid:74b9f5d0-33fb-4a81-b02b-5b760641c1d6</
        prop:identity>
        <prop:sentAt>2000-05-14T03:00:00+08:00</prop:sentAt>
        <prop:expiresAt>2000-05-15T04:00:00+08:00</prop:expiresAt>
        <prop:topic>http://electrocommerce.org/purchase_order/</
        prop:topic>
    </prop:properties>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <po:PurchaseOrder xmlns:po="http://electrocommerce.org/
    purchase_order/">
          <po:Title>Essential BizTalk</po:Title>
    </po:PurchaseOrder>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Этот случай полностью укладывается в спецификацию SOAP 1.1 в части пересылки по HTTP, поскольку тело сообщения BizTalk состоит из одного сообщения SOAP.

Единственное дополнение к SOAP 1.1 — значение элемента SOAP-Action заголовка HTTP. Его значением должен быть идентификатор URI из обязательного элемента <topic> обязательного элемента заголовка <properties>.

11.1.2 Пример BizTalk-сообщения с вложениями

```
POST /insuranceClaims HTTP/1,1
Host: www.risky-stuff.com
Content-Type: Multipart/Related;
            boundary=biztalk_2_0_related_boundary_example;
```

```
            type=text/xml;
            start="<claim.xml@claiming-it.com>"
Content-Length: nnnn
SOAPAction: "http://schemas.risky-stuff.com/Auto-Claim"
Content-Description: This is tne opt:.onal message description.
-biztalk_2_0_related_boundary_example
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <claim.xml@claim.ng-it.com>
<?xml version='1.0' ?>
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2000/08/XMLSchema-datatypes">
   <SOAP-ENV:Header      xmlns:agr="http://www.trading-agreements.org/
   types/">
      <!- декларация опушена для краткости ->
      <endpoints SOAP-ENV:mustUnderstand="1"
      xmlns="http://schemas.biztalk.org/btf-2-0/endpoints">
         <to>
            <address xsi:type="agr:dept">insurance_claim_department</
            address>
         </to>
         <from>
            <address xsi:type="agr:agent">agent:/WA/
            Issaquah#id=12345</address>
         </from>
      </endpoints>
      <prop:properties SOAP-ENV:mustUnderstand="1"
      xmlns:prop="http://schemas.biztalk.org/btf-2-0/properties">
      <!- otner elements omitted for brevity ->
      <topic>http://schemas.risky-stuff.com/Auto-Claim</topic>
      </prop:properties>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <Insurance_Claim_Auto xmlns="http://schemas.risky-stuff.com/
      Auto-Claim"
         id="insurance_claim_document_id">
         <!- ...дополнительное описание... ->
```

```
      </Insurance_Claim_Auto>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
-biztalk_2_0_related_boundary_example
Content-Type: image/tiff
Content-Transfer-Encoding: base64
Content-ID: <claim.tiff@claiming-it.com>
...Изображение в формате TIFF в кодировке Base64...
-biztalk_2_0_related_boundary_example
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <car.jpeg@claiming-it.com>
...Файл JPEG...
-biztalk_2_0_related_boundary_example-
```

Основа предложенного в настоящей спецификации подхода к передаче контейнеров MIME в сообщении HTTP — использование заголовка составного носителя на уровне HTTP с его трактовкой как обычного заголовка HTTP. Правила организации сообщения BizTalk для документа BizTalk с вложениями в составной контейнер MIME согласно разделу 9.1 таковы.

- В заголовке HTTP должен присутствовать заголовок MIME Content-Type: multipart/related. Сформулированные в разделе 9.1 правила в отношении параметров этого заголовка применимы и здесь.
- В заголовке HTTP недопустимы другие заголовка MIME (как-то: Content-Transfer-Encoding). В частности, заголовок MIME-Version: 1.0 не должен присутствовать в заголовке HTTP. Это требование не распространяется на заголовки HTTP 1.1, использующие семантику, аналогичную MIME.
- Части сообщения MIME, содержащие документ BizTalk и вложения, должны размещаться в теле HTTP-сообщения в точном соответствии с требованиями раздела 9.1 (включая требования к заголовкам MIME).

11.2 SMTP

Транспортировка по SMTP требует значительно меньшей проработки, нежели случай HTTP. SMTP — асинхронный протокол, что полностью соответствует асинхронной модели сообщений инфраструктуры BizTalk, так что в описании ответных сообщений нет нужды. Спецификация SOAP 1.1 не определяет конкретных требований к транспортировке по протоколу SMTP; отсутствуют и SOAP-заголов-

ки для передачи по SMTP. SMTP полностью совместим с MIME и полностью поддерживает семантику MIME (она, собственно, и была предназначена для SMTP). При пересылке по SMTP для документов BizTalk, не нуждающихся Б защите и не содержащих вложений, применяется тип text/xml; во всех остальных случаях, включая фрагменты документов BizTalk, защищенные средствами S/M1M E (см. раздел 10.1), используется тип multipart/related. Поскольку в документах XML часто применяются кодировки, несовместимые с 7-битным представлением (UTF-8, UTF-16 и т. п.). при пересылке сообщений по почтовой инфраструктуре, способной нести только 7-битные текстовые сообщения, рекомендуется предварительно преобразовывать сообщение в подходящее представление (bin64, quoted-printable и т. п.).

12. Цитированная литература

12.1 Нормативные источники

Спецификация BizTalk Framework основана на стандартах Интернета. Мы сделали все возможное, чтобы избежать противоречий между содержанием нормативных документов и их трактовкой в спецификации. Если все же такое противоречие обнаружится, приоритет отдается нормативному документу, а спецификация BizTalk Framework должна быть приведена в соответствие с нормативным источником.

Перечисленные ниже источники являются нормативными для настоящей спецификации.

[XML] Extensible Markup Language (XML) 1.0: *http://www.w3.org/TR/1998/REC-xml-19980210.*

[SOAP] Simple Object Access Protocol (SOAP) Version 1.1: *http://www.w3.org/TR/SOAP.*

[XMLNS] Namespaces in XML: *http://www.w3.org/TR/1999/REC-xml-names-199901]4.*

[URI] Uniform Resource Identifiers (URI): Generic Syntax: *http://www.ietf.org/rfc/rfc2396.txt.*

[ISO8601J ISO 8601: Representations of dates and times: *http://www.iso.ch/markete/8601.pdf.*

[HTTP] Hypertext Transfer Protocol—HTTP/1.1: *http://www.ietf.org/rfc/rfc2616.txt.*

[XML-MIME] XML Media Types: *http://www.ietf.org/rfc/rfc2376.txt.*

[MULTIPART] The MIME Multipart/Related Content-type: *http://www.ietf.org/rfc/rfc2387.txt.*

[MIME1] MIME Part One: Format of Internet Message Bodies: *http://www.ietf.org/rfc/rfc2045.txt.*

[MIME2] MIME Part Two: Media Types: *h ttp://www. ietf.org/rfc/rfc204fxt.*

[MIME3] MIME Part Three: Message Header Extensions for Non-ASCII Text: *http://www.ietf.org/rfc/rfc2047.txt.*

[MIME4] MIME Part Four: Registration Procedures: *http://www.ietf.org/rfc/rfc2048.txt.*

[SMIME] S/MIME Version 3 Message Specification: *http://www. ietf.org/rfc/rfc2633.txt.*

[CID] Content-ID and Message-ID Uniform Resource Locators: *http://www.ietf.org/rfc/rfc2111.txt.*

12.2 Ненормативные источники

Перечисленные ниже источники использованы в настоящей спецификации, но не являются нормативными.

[XDR] XML-Data Reduced (XDR): *http://www.ltg.ed.ac.uk/~ht/XMLData-Reduced.htm.*

[XSD1] XML Schema Part 1: Structures: *http://www.w3: org/TR/xmlschema-1.*

[XSD2] XML Schema Part 2: Data types: *http://www.w3.org/TR/xmlschema-2.*

[XP] XML Protocol: *http://www.w3.org/2000/xp/.*

Дополнение. Схемы документа BizTalk

1.1 XDR-схемы

1.1.1 Элемент заголовка <endpoints>

```
<?xml version= 1.0" ?>
<!-
 BizTalk Framework 2.0
 Схема документа BizTalk: заголовок endpoints
 (C) 2000 Microsoft Corporation
->
<Schema
 name="biztalk_2_0_endpoints.xml"
 xmlns="urn:schemas-microsoft-com:xml-data"
 xmlns:dt="urn:schemas-microsoft-com:datatypes"
```

```
 xmlns SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:xsi="http //www.w3.org/1999/XMLSchema-instance">
 <!- элемент заголовка endpoints -->
 <ElementType name="endpoints" content="eltOnly">
 <attribute type="SOAP-ENV:mustUnderstand" default="1" required="yes"/>
 <element type="to" minOccurs="1" maxOccurs="1"/>
 <element type="from" minOccurs="1" maxOccurs="1"/>
 </ElementType>
 <ElementType name="to" content="eltOnly">
 <element type="address" minOccurs="1" maxOccurs="1"/>
 </ElementType>
 <ElementType name="from" content="eltOnly">
 <element type="address" minOccurs="1" maxOccurs="1"/>
 </ElementType>
 <ElementType name="address" content="textOnly" dt:type="string">
 <attribute type="xsi:type" required="yes"/>
 </ElementType>
</Schema>
```

1.1.2 Элемент заголовка <properties>

```
<?xml version="1.0" ?>

<!-
 BizTalk Framework 2.0
 Схема документа BizTalk: заголовок properties
 (C) 2000 Microsoft Corporation
->
<Schema
 name="biztalk_2_0_properties.xml"
 xmlns="urn:schemas-microsoft-com:xml-data"
 xmlns:dt="urn:schemas-microsoft-com:datatypes"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <ElementType name="properties" content="eltOnly">
 <attribute type="SOAP-ENV:mustUnderstand" default="1" required="yes"/>
 <element type="identity" minOccurs="1" maxOccurs="1"/>
 <element type="sentAt" minOccurs="1" maxOccurs="1"/>
 <element type="expiresAt" minOccurs="1" maxOccurs="1"/>
 <element type="topic" minOccurs="1" maxOccurs="1"/>
 </ElementType>
 <ElementType name="identity" content="textOnly" dt:type="uri"/>
 <ElementType name="sentAt" content="textOnly" dt:type="dateTime.tz"/>
```

```
 <ElementType name="expiresAt" content="textOnly" dt:type="dateTime.tz"/>
 <ElementType name="topic" content="textOnly" dt:type="uri"/>
</Schema>
```

1.1.3 Элемент заголовка <services>

```
<?xml version="1.0" ?>
<!-
 BizTalk Framework 2.0
 Схема документа BizTalk: заголовок services
 (C) 2000 Microsoft Corporation
->
<Schema
 name="biztalk_2_0_services.xml"
 xmlns="urn:schemas-microsoft-com:xml-data"
 xmlns:dt="urn:schemas-microsoft-com:datatypes">
 <!- services header entry element ->
 <ElementType name="services" content="eltOnly">
 <element type="deliveryReceiptRequest" minOccurs="0" maxOccurs="1"/>
 <element type="commitmentReceiptRequest" minOccurs="0" maxOccurs="1"/>
 </ElementType>
 <ElementType name="deliveryReceiptRequest" content="eltOnly" ;
 <element type="sendTo" minOccurs="1" maxOccurs="1"/>
 <element type="sendBy" minOccurs="1" maxOccurs="1"/>
 </ElementType>
 <ElementType name="commitmentReceiptRequest" content="eltOnly" >
 <element type="sendTo" minOccurs="1" maxOccurs="1"/>
 <element type="sendBy" minOccurs="1" maxOccurs="1"/>
 </ElementType>
 <ElementType name="address" content="textOnly" dt:type="string">
 <attribute type="xsi:type" required="yes"/>
 </ElementType>
 <ElementType name="sendTo" content="eltOnly">
 <element type="address" minOccurs="1" maxOccurs="1"/>
 </ElementType>
 <ElementType name="sendBy" content="textOnly" dt:type="dateTime.tz"/>
</Schema>
```

1.1.4 Элемент заголовка <manifest>

```
<?xml version="1.0" ?>
<!-
 BizTalk Framework 2.0
```

```
 Схема документа BizTalk: заголовок manifest
 (C) 2000 Microsoft Corporation
->
<Schema
 name="biztalk_2_0_manifest.xml"
 xmlns="urn:schemas-microsoft-com:xml-data"
 xmlns:dt="urn:schemas-microsoft-com:datatypes">
 <ElementType name="manifest" content="eltOnly">
 <element type="reference" minOccurs="1" maxOccurs="*"/>
 </ElementType>
 <ElementType name="reference" content="eltOnly">
 <group minOccurs="1" maxOccurs="1" order="one">
 <element type="document">
 <element type="attachment">
 </group>
 <element type="description" minOccurs="0" maxOccurs="1"/>
 </ElementType>
 <ElementType name="document" content="eltOnly">
 <attribute type="href" required="yes"/>
 </ElementType>
 <ElementType name="attachment" content="eltOnly">
 <attribute type="href" required="yes"/>
 </ElementType>
 <AttributeType name="href" dt:type="uri"/>
 <ElementType name="description" content="mixed"/>
</Schema>
```

1.1.5 Элемент заголовка <process>

```
<?xml version="1.0" ?>
<!-
 BizTalk Framework 2.0
 Схема документа BizTalk: заголовок process
 (C) 2000 Microsoft Corporation
->
<Schema
 name="biztalk_2_0_process.xml"
 xmlns="urn:schemas-microsoft-com:xml-data"
 xmlns:dt="urn:schemas-microsoft-com:datatypes"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <ElementType name="process" content="eltOnly">
 <attribute type="SOAP-ENV:mustUnderstand" default="1" required="yes"/>
 <element type="type" minOccurs="1" maxOccurs="1"/>
```

```
 <element type="instance" minOccurs="1" maxOccurs="1"/>
 <element type="detail" minOccurs="0" maxOccurs="1"/>
 </ElementType>
 <ElementType name="type" content="textOnly" dt:type="uri"/>
 <ElementType name="instance" content="textOnly" dt:type="uri"/>
 <ElementType name="detail" content="eltOnly" model="open"/>
</Schema>
```

1.1.6 Элемент заголовка <deliveryReceipt>

```
<?xml version="1.0" ?>
<!-
 BizTalk Framework 2.0
 Схема документа BizTalk: заголовок deliveryReceipt
 (C) 2000 Microsoft Corporation
->
<Schema
 name="biztalk_2_0_process_receipt_header.xml"
 xmlns="urn:schemas-microsoft-com:xml-data"
 xmlns:dt="urn:schemas-microsoft-com:datatypes"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <ElementType name="deliveryReceipt" content="eltOnly">
 <attribute type="SOAP-ENV:mustUnderstand" default="1" required="yes"/>
 <element type="receivedAt" minOccurs="1" maxOccurs="1"/>
 <element type="identity" minOccurs="1" maxOccurs="1"/>
 </ElementType>
 <ElementType name="receivedAt" content="textOnly" dt:type="dateTime.tz"/>
 <ElementType name="identity" content="textOnly" dt:type="uri"/>
</Schema>
```

1.1.7 Элемент заголовка <commitmentReceipt>

```
<?xml version="1.0" ?>
<!-
 BizTalk™ Framework 2.0
 Схема документа BizTalk: заголовок commitmentReceipt
 (C) 2000 Microsoft Corporation
->
<Schema
 name="biztalk_2_0_process_receipt_header.xml"
 xmlns="urn:schemas-microsoft-com:xml-data"
 xmlns:dt="urn:schemas-microsoft-com:datatypes"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
```

```
<ElementType name="commitmentReceipt" content="eltOnly">
<attribute type="SOAP-ENV:mustUnderstand" default="1" required="yes"/>
<element type="identity" minOccurs="1" maxOccurs="1"/>
<element type="decidedAt" minOccurs="1" maxOccurs="1"/>
<element type="decision" minOccurs="1" maxOccurs="1"/>
<element type="commitmentCode" minOccurs="0" maxOccurs="1"/>
<element type="commitmentDetail" minOccurs="0" maxOccurs="1"/>
</ElementType>
<ElementType name="identity" content="textOnly" dt:type="uri"/>
<ElementType name="decidedAt" content="textOnly" dt:type="dateTime.tz"/>
<ElementType name="decision" content="textOnly" dt:type="string"/>
<ElementType name="commitmentCode" content="textOnly" dt:type="string"/>
<ElementType name="commitmentDetail" content="eltOnly" model="open"/>
</Schema>
```

1.1.8 Конверт SOAP 1.1 для документов BizTalk

```
<?xml version="1.0" ?>
<!-
BizTalk Framework 2.0
Схема документа BizTalk: конверт
(C) 2000 Microsoft Corporation
Схема основана на SOAP 1.1
->
<Schema
   name="biztalk_2_0_document_envelope.xml"
   xmlns="urn:schemas-microsoft-com:xml-data"
   xmlns:dt="urn:schemas-microsoft-com:datatypes">
   <!-
SOAP: конверт, заголовок и тело
   ->
   <ElementType name="Envelope" content="eltOnly">
      <element type="Header" minOccurs="1" maxOccurs="1"/>
      <element type="Body" minOccurs="1" maxOccurs="1"/>
   </ElementType>
   <ElementType name="Header" content="eltOnly" model="open"
         xmlns:eps="http://schemas.biztalk.org/btf-2-0/endpoints"
         xmlns:prop="http://schemas.biztalk.org/btf-2-0/properties"
         xmlns:fst="http://schemas.biztalk.org/btf-2-0/manifest"
         xmlns:orc="http://schemas.biztalk.org/btf-2-0/process">
      <element type="eps:endpoints"          minOccurs="1" maxOccurs="1"/>
      <element type="prop:properties"        minOccurs="1" maxOccurs="1"/>
      <element type="fst:services"           minOccurs="0" maxOccurs="1"/>
```

```
        <element type="fst:manifest"            minOccurs="0" maxOccurs="1"/>
        <element type="prc:process"             minOccurs="0" maxOccurs="1"/>
    </ElementType>
    <ElementType name="Body" content="eltOnly" model="open"/>
    <!-
        Глобальные атрибуты. Перечисленные ниже атрибуты требуют
        ссылки по полному имени
    ->
    <AttributeType name="mustUnderstand" default="0" dt:type="Boolean"/>
    <AttributeType name="actor" dt:type="uri"/>
<!-
Атрибут 'encodingStyle' указывает используемое соглашение
о кодировании. Например, значение
'http://schemas.xmlsoap.org/soap/encoding/' указывает
на использование схемы кодирования из спецификации SOAP.
->
<AttributeType name="encodingStyle" dt:type="string"/>
<!-
Структура для уведомления об ошибке
->
    <ElementType name="Fault" content="eltOnly">
        <element type="faultcode" minOccurs="1" maxOccurs="1"/>
        <element type="faultstring" minOccurs="1" maxOccurs="1"/>
        <element type="faultactor" minOccurs="0" maxOccurs="1"/>
        <element type="detail" minOccurs="0" maxOccurs="1"/>
    </ElementType>
<ElementType name="faultcode" content="textOnly" dt:type="string"/>
<ElementType name="faultstring" content="textOnly" dt:type="string"/>
<ElementType name="faultactor" content="textOnly" dt:type="uri"/>
<ElementType name="detail" content="eltOnly" model="open"/>
</Schema>
```

1.2 XSD-схемы

1.2.1 Элемент заголовка <endpoints>

```
<?xml version="1.0" ?>
<schema xmlns="http://www.w3.org/2000/08/XMLSchema"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:tns="http://schemas.biztalk.org/btf-2-0/endpoints"
 targetNamespace="http://schemas.biztalk.org/btf-2-0/endpoints"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
```

```
elementFormDefault="qualified"
attributeFormDefault="qualified">
<annotation>
<documentation>
BizTalk Framework 2.0
Схема документа BizTalk: заголовок endpoints
(C) 2000 Microsoft Corporation
</documentation>
</annotation>
<element name="endpoints" type="tns:endpoints"/>
<complexType name="endpoints">
<attribute ref="SOAP-ENV:mustUnderstand" use="required" value="1" />
<sequence>
<element name="to" type="tns:endpoint"/>
<element name="from" type="tns:endpoint"/>
</sequence>
</complexType>

<complexType name="endpoint">
<element name="address" type="tns:address"/>
</complexType>

<complexType name="address">
<simpleContent>
<extension base="string">
<attribute ref="xsi:type" use="required"/>
</extension>
</simpleContent>
</complexType>
</schema>
```

1.2.2 Элемент заголовка <properties>

```
<?xml version="1.0" ?>
<schema xmlns="http://www.w3.org/2000/08/XMLSchema"
 xmlns:xsi="http://www.w3 org/1999/XMLSchema-instance"
 xmlns:tns="http://schemas.biztalk.org/btf-2-0/properties"
 targetNamespace="http://schemas.biztalk.org/btf-2-0/properties"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 elementFormDefault="qualified"
 attributeFormDefault="qualified">
 <annotation>
```

```
<documentation>
BizTalk Framework 2.0
Схема документа BizTalk: заголовок properties
(C; 2000 Microsoft Corporation
</documentation>
</annotation>
<element name="properties" type="tns:properties"/>
<complexType name="properties">
<attribute ref="SOAP-ENV:mustUnderstand" use="required" value="1" />
<sequence>
<element name="identity" type="uriReference"/>
<element name="sentAt" type="timeInstant"/>
<element name="expiresAt" type="timeInstant"/>
<element name="topic" type="uriReference"/>
</sequence>
</complexType>
</schema>
```

1.2.3 Элемент заголовка <services>

```
<?xml version="1.0" ?>
<schema xmlns="http://www.w3.org/2000/08/XMLSchema"
        xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
        xmlns:tns="http://schemas.biztalk.org/btf-2-0/services"
        targetNamespace="http://schemas.biztalk.org/btf-2-0/services"
        xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
        elementFormDefault="qualified"
        attributeFormDefault="qualified">
<annotation>
    <documentation>
        BizTalk Framework 2.0
        Схема документа BizTalk: заголовок services
        (C) 2000 Microsoft Corporation
    </documentation>
</annotation>
<element name="services" type="tns:services"/>
<complexType name="services">
    <attribute ref="SOAP-ENV:mustUnderstand" use="required" value="1"/>
    <sequence>
        <element name="deliveryReceiptRequest" type="tns:receiptRequest"
minOccurs="0"/>
    <element name="commitmentReceiptRequest" type="tns:receiptRequest"
```

```
minOccurs="0"/>
            </sequence>
</complexType>
<complexType name="receiptRequest">
            <element name="sendTo" type="eps:endpoint"/>
            <element name="sendBy" type="timeInstant"/>
</complexType>
</schema>
```

1.2.4 Элемент заголовка <manifest>

```
<?xml version="1.0" ?>
<schema xmlns="http://www.w3.org/2000/08/XMLSchema"
 xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
 xmlns:tns="http://schemas.biztalk.org/btf-2-0/manifest"
 targetNamespace="http://schemas.biztalk.org/btf-2-0/manifest"
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 elementFormDefault="qualified"
 attributeFormDefault="qualified">
 <annotation>
 <documentation>
 BizTalk Framework 2.0
 Схема документа BizTalk: заголовок manifest
 (C) 2000 Microsoft Corporation
 </documentation>
 </annotation>
 <element name="manifest" type="tns:manifest"/>
 <complexType name="manifest">
 <attribute ref="SOAP-ENV:mustUnderstand" use="optional" default="0"/>
 <element name="reference" minOccurs="1" maxOccurs="unbounded">
 <complexType>
 <sequence>
 <choice>
 •element name="document" type="tns:itemType">
 <element name="attachment" type="tns:itemType">
 </choice>
 <element name="description" type="tns:markup" minOccurs="0"/>
 </sequence>
 </complexType>
 </element>
 </complexType>
 <complexType name="itemType">
```

```
 <attribute ref= SOAP-ENC:ref" use="required"/>
 </complexType>
 <complexType name="markup' mixed="true">
 <simpleContent>
 <extension base="string"/>
 </simpleContent>
 </complexType>
</schema>
```

1.2.5 Элемент заголовка <process>

```
<?xml version="1.0" ?>
<schema xmlns="http://www.w3.org/2000/08/XMLSchema"
 xmlns:xsi="http://www.w3 org/1999/XMLSchema-instance"
 xmlns:tns="http://schemas.biztalk.org/btf-2-0/process"
 targetNamespace="http://schemas biztalk.org/btf-2-0/process"
 xmlns SOAP-ENV="http://schemas. xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas. xmlsoap.org/soap/encoding/"
 elementFormDefault="qualified"
 attributeFormDefault="qualified ·
 <annotation>
 <documentation>
 BizTalk Framework 2.0
 Схема документа BizTalk: заголовок process
 (C) 2000 Microsoft Corporation
 </documentation>
 </annotation>
 <element name="process" type="tns:process"/>
 <complexType name="process">
 <attribute ref="SOAP-ENV:mustUnderstand" use="required" value="1"/>
 <sequence>
 <element name="type" type="UriReference"/>
 <element name="instance" type="UriReference"/>
 <element name="detail" type="anyType" minOccurs="0">
 </sequence>
 </complexType>
</schema>
```

1.2.6 Элемент заголовка <receipt>

```
<?xml version="1.0" ?>
<schema xmlns="http://www.w3.org/2000/08/XMLSchema"
 xmlns:tns="http://schemas biztalk.org/btf-2-0/receipts"
```

```
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
targetNamespace="http://schemas.biztalk.org/btf-2-0/receipts"
elementFormDefault="qualified"
attributeFormDefault="qualified">
<annotation>
<documentation>
BizTalk Framework 2.0
Схема документа BizTalk: заголовок receipt
(C) 2000 Microsoft Corporation
</documentation>
</annotation>
<element name="deliveryReceipt" type="tns:deliveryReceipt"/>
<element name="commitmentReceipt" type="tns:commitmentReceipt"/>
<complexType name="deliveryReceipt">
<attribute ref="SOAP-ENV:mustUnderstand" value="1" use="required"/>
<sequence>
<element name="receivedAt" type="timeInstant"/>
<element name="identity" type="uriReference"/>
</sequence>
</complexType>
<complexType name="commitmentReceipt">
<attribute ref="SOAP-ENV:mustUnderstand" value="1" use="required"/>
<sequence>
<element name="identity" type="uriReference"/>
<element name="decidedAt" type="timeInstant"/>
<element name="decision" type="tns:decision"/>
<element name="commitmentCode" type="QName"/>
<element name="commitmentDetail" type="anyType" minOccurs="0"/>
</sequence>
</complexType>
<simpleType name="decision">
<restriction base="string">
<enumeration value="positive"/>
<enumeration value="negative"/>
</restriction>
</simpleType>
</schema>
```

1.2.7 Конверт документа BizTalk

```
<?xml version="1.0" ?>
<schema xmlns="http://www.w3.org/2000/08/XMLSchema"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
```

```
    targetNamespace="http://schemas.xmlsoap.org/soap/envelope/"
    elementFormDefault="qualified"
    attributeFormDefault="qualified">
    <annotation>
        <documentation>
            BizTalk Framework 2.0
            Схема документа BizTalk  конверт
            (C) 2000 Microsoft Corporation
        </documentation>
    </annotation>
    <!- specify required and optional BTF2 headers ->
    <redefine schemaLocation="http://schemas.xmlsoap.org/soap/envelope/">
    <complexType name= Header"
        xmlns eps="http://schemas biztalk.org/btf-2-0/endpoints"
        xmlns:prop="http://schemas.biztalk.org/btf-2-0/properties"
        xmlns:svc="http://schemas biztalk.org/btf-2-0/services"
        xmlns:fst="http://schemas.biztalk.org/btf-2-0/manifest"
        xmlns:prc="http://schemas.biztalk.org/btf-2-0/process"
        xmlns:rct="http://schemas biztalk.org/btf-2-0/receipts >
        <all>
            <sequence>
                <element ref="eps:endpoints"/>
                <elementref="prop:properties"/>
                <element ref="svc services" minOccurs="0"/>
                <element ref="fst manifest  minOccurs="0"/>
                <element ref="prc:process" minOccurs="0"/>
                <choice>
                    <element ref="rct:deliveryReceipt" minOccurs="0"/>
                    <element ref="rct:commitmentReceipt" minOccurs="0"/>
                </choice>
            </sequence>
            <any minOccurs="0" maxOccurs="unbounded"/>
            <anyAttribute/>
        </all>
    </complexType>
    </redefine>
</schema>
```

Корпорация Microsoft настоящим предоставляет всем пользователям спецификации BizTalk Framework, версия 2.0 (именуемой далее «спецификация») не ограниченное по времени, неисключительное, свободное от каких-либо отчислений право и лицензию на копирование, публикацию и распространение спецификации. Корпорация Microsoft дополнительно предоставляет всем пользователям право (с сохранением прав интеллектуальной собственности Microsoft) на реализацию и использование описанных в настоящей спецификации XML-тэгов BizTalk и рекомендаций по созданию схем при разработке компьютерных программ. Единственным условием предоставления этого права является отказ компании, реализующей спецификацию, от патентных претензий к Microsoft и других компаний, реализующих спецификацию. Корпорация Microsoft оставляет за собой все остальные права на материал и тематику спецификации. Корпорация Microsoft декларирует отказ от каких-либо гарантий в отношении спецификации, включая гарантию того, что спецификация или ее реализация не нарушает прав других сторон.

Информация, приведенная в настоящем документе, может изменяться без предварительного уведомления.

© 1999-2000, корпорация Microsoft, Все права охраняются.

Microsoft и BizTalk Framework — зарегистрированные товарные знаки корпорации Microsoft в США и/или других странах.

Приложение В

Web-сайты, посвященные XML и BizTalk

На многих сайтах есть разделы, посвященные XML, схемам и BizTalk. На BizTalk специализируются два сайта: *http://biztalk.org* (поддерживается Комитетом BizTalk) и сайт BizTalk компании Microsoft *http://mvw.microsoft.com/biztalk*. Я большой поклонник и частый посетитель этих и других сайтов, описанных в данном приложении.

BizTalk.org

На сайте BizTalk.org можно найти массу информации об XML вообще и BizTalk в частности. Здесь есть разделы, которые помогут начинающему специалисту изучить технологию и ее практическое применение, связаться с другими специалистами, использующими XML и BizTalk, и овладеть инструментами, которые помогут создать более мощные и эффективные B2B- или B2C-приложения.

Если завтра ваш начальник скажет: «Нам нужно работать с Web-службами в такой-то B2B-системе, причем уже завтра», — начать стоит именно с BizTalk.org. Этот сайт обеспечит вас необходимой технической информацией и поможет понять, какие именно схемы и приложения использует ваш конкретный партнер или группа компаний.

Прямая поддержка

Многие компании, занимающиеся стандартизацией деловых отношений, обладают гораздо большим опытом в моделировании бизес-процессов, чем в программировании и XML. На сайте BizTalk.org они

найдут средства для применения в своих приложениях существующих и создания собственных схем.

Возможно, самое важное достоинство BizTalk.org — это свободный доступ для любого, кто желает опубликовать свою XML-схему или найти опубликованную. Задача сайта — поддержка XM L-решений для бизнеса, для чего BizTaIk.org использует два основных раздела. Во-первых, учебный — здесь можно найти новости и пообщаться с коллегами. Этот раздел представляет собой группу форумов, организованных по принципу «вопрос — ответ», где обсуждается разработка схем.

Второй раздел BizTalk.org — библиотека схем. Все схемы предварительно проверены на действительность и доступны всем желающим. Разработчики могут быть уверены, что любая схема, найденная на BizTalk.org, протестирована я действительна. По приблизительной оценке до 80% схем, доступных в Интернете, не проходили проверку на действительность, Недействительные схемы бесполезны для проверки XML-документов (даже документов, созданных автором схемы). Схема, допущенная для публикации на BizTalk.org, проходит специальную проверку на действительность, чтобы гарантировать, что не содержит ошибок. Примеры, предлагаемые со схемой, проверены и гарантированно действительны согласно своей схеме.

BizTalk.org предоставляет всем желающим круглосуточный доступ к размещенным на нем проверенным схемам.

Сайт предоставляет описание и уникальный указатель ресурса для каждой опубликованной схемы, а также возможность контроля версий. В схему, опубликованную на BizTalk.org, никто, даже автор, не может внести изменения — можно лишь опубликовать следующую версию. Контроль версий — обязательный атрибут любого хранилища схем; в самом деле, вряд ли разумно иметь дело с хранилищем, схемы которого могут измениться в любое время.

Домашняя страница Microsoft BizTalk

Домашняя страница Microsoft BizTalk — *http://www.microsoft.com/biztalk* -- посвящена информации об инфраструктуре BizTalk и Microsoft BizTalk Server 2000. В период разработки BizTalk Server 2000 Web-сайт Microsoft BizTalk предложил пробную версию продукта всем желающим, чтобы потенциальные пользователи могли оценить особенности программы и ее применимость в каждом конкретном случае. Этот сайт также предлагает учебные курсы по работе с XML, демонстрирующие примеры применения инфраструктуры BizTalk различными компаниями для поддержки электронного бизнеса. Microsoft планирует использовать этот сайт для поддержки BizTalk Server 2000.

XML.com

В 1998 г. Songline Studios, дочерняя компания фирмы O'Reilly and Associates, объединилась с компанией Seybold Publications для создания XML.com (*http://www.xml.com*)— рекламно-коммерческого сайта, посвященного описанию текущего состояния дел в разработке и применении XML. Главной задачей XML.com является подробное освещение текущего состояния разработок и достижений в области стандартов и XML-решений.

XML.com не просто распространяет техническую информацию и пресс-релизы. Штатные и внештатные авторы публикуют здесь статьи, посвященные конфликтным ситуациям в промышленности и специфическим особенностям применения XML, которые могут вызвать проблемы несовместимости. Так, недавно были рассмотрены результаты тестирования нескольких анализаторов, полученные при использовании комплекса структурного тестирования XML (XML Conformance Test Suite) — специального набора тестов, разработанного Организацией содействия развитию структурированных информационных систем (Organization for the Advancement of Structured Information Systems, OASIS) и Национальным институтом стандартов и технологий США (U.S. National Institute of Standards and Technology, NIST) для проверки совместимости анализаторов со спецификациями XML, публикуемыми консорциумом W3C.

OASIS

Эта промышленная группа по XML предоставляет независимую информацию о состоянии дел в мире XML. OASIS — это консорциум компаний, заинтересованных в применении XML для решения проблем бизнеса. Большинство этих компаний поставляет продукцию, в той или иной степени поддерживающую XML, но в OASIS входят и пользователи XML.

Лучший ресурс на Web-сайте OASIS *(http://www.oasis-open.org)* — титульные страницы XML (*XML Cover Pages),* которые (по крайней мере на момент публикации этой книги) неустанно поддерживает Робин Ковер (Robin Cover). Робин много лет имел дело с SGML, так что он очень хорошо представляет, что в первую очередь должны узнать люди, столкнувшиеся с XML.

Титульные страницы XML обновляются ежедневно (а иногда и несколько раз в день) и предоставляют не только новости разных компаний, но и информацию о разработках и стандартах XML.

XML.ORG

В мае 1999 г. группа OASIS подписала соглашение с 8 компаниями, формирующими рынок XML, о поддержке нового сайта XML.ORG (*http://www.xml.org*) посвященного промышленному применению XML. Сайт призван служить надежным, независимым источником информации для тех, чьи данные должны быть совместимы с системами различных поставщиков.

Управляющий комитет XML.ORG и Совет директоров OASIS работают над созданием промышленного портала для регистрации и хранения схем. Они занимаются и другими важными проблемами, например, вопросами интеллектуальной собственности (ИС).

Ни один из законов прецедентного права США в настоящее время не занимается проблемой прав ИС владельцев схем. OASIS объединяет ведущих специалистов в области ИС и будет способствовать выработке стратегии в этой области. Очевидно, такая стратегия должна быть сформулирована до начала масштабной регистрации и публикации схем.

Сформированный группой технический комитет по регистрации и хранению (Registry and Repository Technical Committee) работает над стандартизацией методов регистрации и упрощения доступа к опубликованным схемам. Это очень важное направление, так как промышленное применение XML требует надежных хранилищ схем для предоставления доступа к пространствам имен. Как только рабочая группа решит свою задачу, будет создано приложение для демонстрации принципов работы спецификации.

Поскольку OASIS аккумулирует данные всей промышленности, результаты работы этой группы носят более комплексный характер, чем у малых групп, организованных конкретными поставщиками. Хотя промышленная стандартизация занимает больше времени, ее результаты имеют больше шансов быть принятыми в качестве стандарта, чем предложения отдельного поставщика, который может преследовать прежде всего собственные интересы.

Информационный бюллетень <TAG>

Информационный бюллетень *<TAG>* (*http://tagnewsletter.com*) публикуется с 1987 г. Это технический бюллетень в области управления информационными технологиями. Я был его редактором с 1991 г. В 1992 г. бюллетень стал ежемесячным, а в 1999 г. мы сделали его интерактивную версию на основе баз данных с использованием XML и XSL для создания персонализированной информации для подписчиков.

Задача <*TAG*> — обеспечить разработчиков в области **XML** инструментами и информацией. < *TAO* носит главным образом технический характер; мы публикуем в основном руководства с примерами кода и техническими советами. Кроме того, мы стараемся публиковать максимум упражнений и книжных обзоров.

В 1999 г. мы коренным образом переработали наш сайт, чтобы обеспечить читателей возможностями персонализации информации Мы конвертировали более 1 000 статей в формат XML и собрали их в БД. Используя ее, мы можем предоставить каждому посетителю возможность выбора статей в зависимости от его интересов. Так, обратившись на наш сайт с помошью Netscape Navigator, пользователь получит страницу совершенно иной структуры, нежели посетитель, применяющий Microsoft Internet Explorer. Наш сайт распознает метод доступа и создает соответствующее представление информации,

Одно из преимуществ такой персонифицированной системы -- возможность мгновенного доступа (для платных подписчиков) ко всем статьям на сайте, включая текущие издания. Подписчики, не оплачивающие доступ, также могут получить все статьи, но они видят только заголовки, фамилии авторов и краткий реферат статей со сроком публикации менее года.

Мы применяем персонализацию и обработку, зависящую от типа от обозревателя, используя **XML,** XSLT и сценарии промежуточного уровня. В процессе разработки мы сделали немало открытий. Например, если в тексте встречается аббревиатура, мы размечаем ее соответствующим образом. Это позволяет нам применять «всплывающее» определение при подготовке электронного представления документа. Многие наши открытия освещены в постоянно обновляемом учебном курсе, который мы создали для наших подписчиков в июне 1999 г.

XMLSOFTWARE

Если вы хотите узнать об инструментах для работы с XML, вы не найдете ничего лучше, чем регулярно обновляемый сайт *XML*SOFTWARE (*http://www.xmlsoftware.com*) который ведут Джеймс Таубер и Линда ван ден Бринк (James Tauber and Linda van den Brink).

*XML*Software оповещает подписчиков по электронной почте о появлении новинок. Авторы сайта также ведут ряд других сайтов, посвященных применению XML и соответствующим инструментам, включая хранилища схем, такие как *XML*INFO *(http://www.xmlinfo.com)* и SCHEMA.NET *(http://www.schema.net)*.

Приложение Г

Simple Object Access Protocol (SOAP) 1.1

Дата публикации: 8 мая 2000 г.
Статус: нота консорциума W3C
Данная версия:
http://www.w3.org/TR/2000/NOTE-SOAP-20000508
Последняя версия:
http://www.w3.org/TR/SOAP
Авторы (в алфавитном порядке):
Дон Бокс (Don Box), DevelopMentor
Дэвид Энебаск (David Ehnebuske), IBM
Гопал Какивая (Gopal Kakivaya), Microsoft
Эндрю Лэймэн (Andrew Layman), Microsoft
Ноа Мендельсон (Noah Mendelsohn), Lotus Development Corp.
Хенрик Ф. Нильсен (Henrik Frystyk Nielsen), Microsoft
Сатиш Тэт (Satish Thatte), Microsoft
Дэйв Винер (Dave Winer), UserLand Software, Inc.

Резюме

SOAP — простой протокол для обмена информацией в децентрализованных распределенных средах. Он основан на XML и состоит из трех частей: конверта, предоставляющего инфраструктуру для описания содержания сообщения и его обработки, правил описания прикладных типов данных и синтаксиса удаленных вызовов процедур и представления их результатов. SOAP можно использовать в сочетании с многими транспортными протоколами, однако в настоящей спецификации описан только случай HTTP (и его расширений).

Статус

Настоящий документ представлен в Консорциум World Wide Web для организации рабочей группы по протоколам на базе XML. Авторы будут признательны за комментарии, однако мы призываем всех заинтересованных высказать свое мнение в публичном списке рассылки W3C <xml-dist-app@w3.org> (см. архив списка).

Настоящий документ имеет статус ноты, опубликованной для обсуждения членами W3C. Публикация ноты не свидетельствует об одобрении заявки ни консорциумом, ни рабочей группой, ни кем-либо из членов консорциума. Консорциум W3C не участвовал в подготовке ноты. Данный документ является рабочим и в любой момент может быть изменен, заменен или признан устаревшим.

Список технических документов W3C и их статус см. на странице отчетов Консорциума W3C (*http://www.w3c.org/TR*).

1. Введение

SOAP реализует простой и максимально облегченный механизм обмена структурированной и типизированной информацией на основе XML между узлами децентрализованной распределенной среды. SOAP не фиксирует прикладную семантику, например, программные модели или требования к реализации, а лишь предлагает простой механизм описания прикладной семантики, включая модель пакетирования и механизм представления данных в пакете. Этот подход обеспечивает применимость SOAP в широком спектре сред, от систем обмена сообщениями до средств удаленного вызова процедур.

SOAP состоит из трех частей.

- Конверт SOAP (раздел 4) образует базовую инфраструктуру для описания того, **что** находится в сообщении, **кто** должен им заниматься и **какие** аспекты сообщения являются обязательными.
- Правила кодирования SOAP (раздел 5) описывают механизм сериализации, позволяющий обмениваться экземплярами прикладных типов данных.
- Механизм удаленного вызова процедур SOAP (SOAP RPC, раздел 7) описывает соглашения о представлении вызовов удаленных процедур и результатов выполнения таких вызовов.

Хотя эти составляющие объединены в одну спецификацию, функционально они не связаны между собой. В частности, конверт и правила представления данных определены в разных пространствах имен с целью добиться максимума простоты за счет модульного подхода.

Помимо конверта, правил представления данных и соглашений о вызове, настоящая спецификация описывает привязку для двух транспортных протоколов, описывающих передачу сообщений SOAP по HTTP [5] с использованием и без использования расширений HTTP [6].

1.1 Цели проекта

Главные цели, которые ставились при проектировании SOAP, — простота и расширяемость. В итоге некоторые возможности, традиционные для систем обмена сообщениями и распределенных объектных систем, изъяты из базовой спецификации SOAP. К ним относятся:

- распределенная сборка мусора;
- пакетная передача сообщений;
- обращение к объектам по ссылке (требует распределенной сборки мусора);
- активация (требует обращения к объектам по ссылке).

1.2 Обозначения

В настоящем документе слова «ДОЛЖЕН», «НЕ ДОЛЖЕН», «ОБЯЗАН», «БУДЕТ», «НЕ БУДЕТ», «МОЖЕТ», «НЕ МОЖЕТ», «РЕКОМЕНДУЕТСЯ» и «НЕ ОБЯЗАТЕЛЬНО» употребляются в соответствии с рекомендациями документа RFC-2119 [2].

Используемые в настоящей рекомендации префиксы пространств имен *SOAP-ENV* и *SOAP-ENC* относятся к пространствам имен SOAP *http://schemas.xmlsoap.org/soap/envelope/* и *http://schemas.xmlsoap.org/soap/encoding/* соответственно.

Префикс пространства имен *xsi* относится к ресурсу *http://www.w3.org/1999/XMLSchema-instance*, описанному в спецификации XML-схем [11], а префикс *xsd* — к ресурсу *http://www.w3.org/1999/XMLSchema*, определенному в [10]. Префикс *tns* обозначает пространство имен текущего документа. Все прочие пространства имен упоминаются исключительно в иллюстративных целях.

Идентификатор ресурса пространства имен вида «некоторый идентификатор» (some-uri) обозначает идентификатор, зависящий от контекста или от приложения [4].

Для описания некоторых конструкций в настоящей спецификации используется расширенная форма Бэкуса-Наура, описанная в RFC-2616 [5].

1.3 Пример сообщения SOAP

В этом примере SOAP-запрос GetLastTradePrice передается службе StockQuote. У запроса есть параметр — строка, задающая аббревиату-

ру ценной бумаги; значение, возвращаемое в ответе SOAP, имеет вещественный тип. Элемент SOAP Envelope является корневым элементом XML-документа, представляющего сообщение SOAP. Чтобы отличить идентификаторы SOAP-элементов от идентификаторов прикладных элементов, применяются пространства имен XML. В примере также использована привязка к HTTP, описанная в разделе 6. Отметим: XML-синтаксис документа SOAP не зависит от факта передачи по транспортному протоколу HTTP.

Другие примеры приведены в приложении А,

Пример 1. Сообщение SOAP, упакованное в HTTP-запрос

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

Ниже приведен пример ответного сообщения SOAP, упакованного в HTTP-ответ.

Пример 2. Сообщение SOAP, упакованное в HTTP-ответ

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
 <SOAP-ENV:Body>
 <m:GetLastTradePriceResponse xmlns:m="Some-URI">
 <Price>34.5</Price>
 </m:GetLastTradePriceResponse>
```

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

2. Модель обмена сообщениями SOAP

Сообщения SOAP представляют собой сообщения, передаваемые в одну сторону — от отправителя получателю, однако, как показано выше, сообщения SOAP часто комбинируются для реализации поведения «запрос — ответ».

Реализация SOAP допускает оптимизацию в соответствии с характеристиками конкретной сети. Так, описанная в разделе 6 транспортировка по HTTP позволяет передавать ответные сообщения SOAP в HTTP-ответах по тому же соединению, что и исходный запрос.

Независимо от транспортного протокола, сообщения SOAP маршрутизируются по т. н. «пути сообщения», что позволяет реализовать обработку сообщения промежуточными узлами до передачи получателю.

SOAP-приложение, получившее сообщение SOAP, ДОЛЖНО обработать его, выполнив следующие действия в указанном порядке.

1. Выделить все составляющие сообщения SOAP, предназначенные для приложения (см. раздел 4.2.2).
2. Убедиться, что приложение поддерживает все обязательные составляющие (см. раздел 4.2.3), выявленные на первом шаге, и соответствующим образом обработать их. Если приложение не поддерживает какой-либо обязательный компонент, сообщение должно быть отвергнуто (см, раздел 4.4). Процессор МОЖЕТ проигнорировать все необязательные для обработки компоненты сообщения, причем на результаты обработки это не влияет.
3. Если SOAP-приложение не является конечным получателем сообщения, оно ДОЛЖНО удалить из сообщения все компоненты, выявленные на первом шаге, до пересылки сообщения получателю.

Кроме того, для обработки сообщения или его части SOAP-процессор должен знать о схеме обмена (односторонний, запрос — ответ, вещательный и т. д.), о роли получателя, о механизмах удаленного вызова (если они применяются), о представлении данных и других семантических особенностях обмена, необходимых для корректной обработки сообщения.

Некоторые аспекты синтаксиса сообщения могут быть описаны различными атрибутами SOAP, такими как *encodingStyle* (см. раздел 4.1.1), однако настоящая спецификация не регулирует способы выяснения получателем этих аспектов в общем случае. Например, для одного приложения элемент <getStockPrice> будет сигнализировать о выполнении RPC-вызова в терминах соглашения, обсуждаемого в

разделе 7. а другое приложение может трактовать тот же элемент как признак одностороннего обмена сообщениями.

3. Связь с XML

Все сообщения SOAP используют синтаксис XML [7],

SOAP-приложение ДОЛЖНО использовать префикс соответствующего пространства имен SOAP для всех элементов и атрибутов создаваемых им SOAP-сообщений. SOAP-приложение ОБЯЗАНО обрабатывать ссылки на пространства имен SOAP в получаемых сообщениях. SOAP-приложение ОБЯЗАНО отказываться от обработки сообщений с некорректно указанными пространствами имен (см. раздел 4.4), но МОЖЕТ обрабатывать SOAP-сообщения без ссылки на пространство имен SOAP так, как если бы пространство имен было указано корректно.

Спецификация SOAP определяет два пространства имен (дополнительную информацию о пространствах имен XML см. в [8]):

- пространство имен конверта SOAP с идентификатором *http://schemas.xmlsoap.org/soap/envelope/*;
- пространство имен сериализации SOAP с идентификатором *http://schemas.xmlsoap.org/soap/encoding/*.

Сообщение SOAP НЕ МОЖЕТ содержать определение типа документа (Document Type Declaration, DTD) или ссылку на него, так же как и инструкции по обработке [7].

SOAP использует локальный атрибут *id* типа *ID* для описания уникальных идентификаторов элементов. Локальный атрибут *href* типа *uri-reference* применяется для ссылки на эти значения в формате, определенном спецификациями XML [7], XML-схемы [11] и языка связывания XML [9].

Атрибуты и их значения, кроме *mustUnderstand* (см. раздел 4.2.3) и *actor (см.* раздел 4.2.2), могут с одинаковым успехом определяться как в XML-документе, так и в схеме. Таким образом, объявление фиксированного значения или значения по умолчанию в схеме семантически эквивалентно указанию значения в документе.

4. Конверт SOAP

Сообщение SOAP — это документ XML, состоящий из обязательного конверта SOAP, необязательного SOAP-заголовка и обязательного тела сообщения SOAP. Далее мы будем называть этот XML-документ *сообщением SOAP.* Элементы и атрибуты, определение которых дается в этом разделе, относятся к пространству имен *http://schemas.xmlsoap.org/soap/envelope/* Сообщение SOAP состоит из:

- конверта — корневого элемента XML-документа сообщения;
- заголовка — общего механизма расширения функциональных возможностей сообщений SOAP без предварительного согласования между обменивающимися сторонами; спецификация SOAP определяет несколько атрибутов, позволяющих указать, кто должен обрабатывать элемент и является ли он обязательным или необязательным (см. раздел 4.2);
- тела сообщения — контейнера для обязательной информации, адресованной конечному получателю сообщения (см. раздел 4.3); спецификация SOAP определяет единственный элемент тела сообщения — *Fault,* который служит для сообщений об ошибках.

Грамматические правила для этих элементов таковы.

1. Конверт.
   - Имя элемента — *Envelope.*
   - Элемент ДОЛЖЕН присутствовать в сообщении SOAP.
   - Элемент МОЖЕТ содержать объявления пространств имен и дополнительные атрибуты. При наличии дополнительных атрибутов они ДОЛЖНЫ быть описаны с указанием пространства имен. Точно так же элемент МОЖЕТ содержать вложенные элементы; эти элементы должны быть приведены с указанием пространства имен и вложены в элемент SOAP *Body.*
2. Заголовок (см. раздел 4.2).
   - Имя элемента — *Header.*
   - Элемент МОЖЕТ присутствовать в сообщении SOAP. Если он присутствует, он ДОЛЖЕН быть первым прямым потомком элемента SOAP *Envelope.*
   - Элемент МОЖЕТ содержать дополнительные элементы заголовка — прямые потомки элемента SOAP *Header.* Все прямые потомки элемента SOAP *Header* ДОЛЖНЫ именоваться с указанием пространства имен.
3. Тело (см. раздел 4.3)
   - Имя элемента — *Body.*
   - Элемент ДОЛЖЕН присутствовать в сообщении SOAP и ДОЛЖЕН быть прямым потомком элемента SOAP *Envelope.* В случае присутствия элемента SOAP *Header* этот элемент ДОЛЖЕН следовать сразу за ним; в противном случае этот элемент ДОЛЖЕН быть первым прямым потомком элемента SOAP *Envelope.*
   - Элемент МОЖЕТ содержать набор элементов, каждый из которых является прямым потомком элемента SOAP *Body.* Прямые потомки элемента SOAP *Body* могут именоваться с указанием пространства имен. Спецификация SOAP определяет эле-

мент *Fault,* который служит для передачи сообщений об ошибках (см. раздел 4.4).

4.1.1 Атрибут SOAP *encodingStyle*

Глобальный атрибут SOAP *encodingStyle* служит для указания правил представления данных в сообщении SOAP. Этот атрибут может появляться в любом элементе; область его действия аналогично области действия пространств имен XML распространяется на элемент, в котором он использован, и на его прямые потомки, не содержащие этого атрибута. Представление данных по умолчанию не определено.

Значение атрибута — упорядоченный список идентификаторов правил сериализации, которые могут применяться для извлечения данных из сообщения SOAP; правила перечисляются в порядке уменьшения степени детализации (от наиболее конкретных к наименее конкретным). Пример значения атрибута *encodingStyle:*

```
"http://schemas.xmlsoap.org/soap/encoding/"
"http://my.host/encoding/restricted http://my.host/encoding/"
""
```

Правила сериализации SOAP (см. раздел 5) идентифицируются указателем пространства имен «*http://schemas.xmlsoap.org/soap/encoding/*». Конкретные правила, используемые сообщением, ДОЛЖНЫ быть указаны в атрибуте *encodingStyle.* Значение этого атрибута, начинающееся с *http://schemas.xmlsoap.org/soap/encoding/* трактуется как декларация соответствия правилам сериализации SOAP (с возможным уточнением).

Пустой идентификатор («») явно указывает на отсутствие правил кодирования элементов. Так, в частности, можно отключить правила, заданные в охватывающих элементах.

4.1.2 Версии конверта

SOAP не использует традиционную модель описания версии с помощью номеров. Любое SOAP-сообщение ДОЛЖНО включать элемент *Envelope* из пространства имен *http://schemas.xmlsoap.org/soap/envelope/.* Если сообщение получено приложением, которое связывает элемент SOAP *Envelope* с другим пространством имен, сообщение ОБЯЗАНО трактовать это как несоответствие версий и отказаться от обработки сообщения. Если сообщение получено по транспорту, поддерживающему схему «запрос — ответ» (такому как HTTP), приложение ОБЯЗАНО вернуть сообщение об ошибке с кодом SOAP *VersionMismatch* (см. раздел 4.4), используя пространство имен SOAP *http://schemas.xmlsoap.org/soap/envelope/.*

4.2 Заголовок SOAP

Спецификация SOAP предлагает гибкий механизм расширения функциональных возможностей сообщений без предварительного согласования между обменивающимися сторонами. Типичные примеры расширений, реализуемых с помощью элементов заголовка, — аутентификация, управление транзакциями и т. д.

Элемент *Header* должен быть первым прямым потомком элемента SOAP *Envelope*. Все прямые потомки элемента *Header* называются элементами заголовка.

Правила кодирования элементов заголовка таковы:

1. элемент заголовка описывается полным именем, включая идентификатор пространства имен и локальное имя; названия всех прямых потомков элемента SOAP *Header* ДОЛЖНЫ включать пространство имен;
2. для описания правил представления элемента заголовка МОЖЕТ использоваться атрибут SOAP *encodingStyle* (см. раздел 4.1.1);
3. для описания адресата (см. раздел 4.2.1) и правил обработки элемента заголовка МОГУТ использоваться атрибуты SOAP *mustUnderstand* (см. раздел 4.2.3) и *actor* (см, раздел 4.2.2).

4.2.1 Атрибуты заголовка

Описанные в этом разделе атрибуты элемента SOAP *Header* определяют способ обработки сообщения SOAP получателем (см. раздел 2). Приложение, создающее SOAP-сообщения. МОЖЕТ применять атрибуты элемента SOAP *Header* только к прямым потомкам этого элемента. Получатель SOAP-сообщения ОБЯЗАН игнорировать любые атрибуты элемента SOAP *Header* у непрямых потомков этого элемента.

Ниже приведен пример элемента заголовка с именем «Transaction», атрибутом «mustUnderstand» со значением «1», и значением 5.

```
<SOAP-ENV:Header>
 <t:Transaction
 xmlns:t="some-URI" SOAP-ENV:mustUnderstand="1">
 5
 </t:Transaction>
</SOAP-ENV:Header>
```

4.2.2 Атрибут actor

Сообщение SOAP, пересылаемое от отправителя конечному получателю, возможно, проходит на своем пути через несколько промежуточных SOAP-приложений. Такие приложения способны принимать

и переадресовывать сообщения SOAP. И промежуточные приложения, и конечный получатель задаются своими универсальными идентификаторами.

Не все компоненты сообщения SOAP могут быть предназначены конечному получателю — некоторые могут быть адресованы промежуточным приложениям. Адресат элемента заголовка связан с отправителем контрактом на обработку этого элемента. В частности, адресат элемента заголовка ОБЯЗАН НЕ пересылать этот заголовок следующему приложению в списке получателей сообщения SOAP. Получатель может добавить в сообщение аналогичный элемент заголовка, однако тогда контракт будет считаться заключенным между ним и получателем нового элемента.

Получатель элемента заголовка указывается глобальным атрибутом SOAP *actor*, значение которого — универсальный идентификатор получателя. Идентификатор *http://schemas.xmlsoap.org/soap/actor/next* указывает, что элемент адресован первому приложению SOAP в списке получателей (аналогично модели передачи от узла к узлу, используемой в поле Connection заголовка HTTP),

Отсутствие атрибута *actor* означает, что единственный адресат — конечный получатель сообщения SOAP.

Данный атрибут действителен только в сообщении SOAP (см. разделы 3 и 4.2.1).

4.2.3 Атрибут *mustUnderstand*

Глобальный атрибут *mustUnderstand* указывает, обязан ли получатель обработать элемент заголовка. Получатель элемента заголовка задается атрибутом SOAP *actor* (см. раздел 4.2.2). Атрибут *muss Understand* может иметь значение «1» или «О». Отсутствие атрибута семантически эквивалентно его наличию со значением «О».

Если элемент заголовка содержит атрибут SOAP *mustUnderstand* со значением «1», получатель ОБЯЗАН понимать семантику элемента (она описана полным именем элемента) и корректно обработать его или ОБЯЗАН вернуть сообщение об ошибке (см. раздел 4.4).

Атрибут *must Understand* гарантирует надежную работу протокола в ходе его последующей эволюции. Элементы с атрибутом SOAP *must-Understand* со значением «I» должны рассматриваться как изменяющие семантику своих родительских и/или дочерних элементов. Применение атрибута *mustUnderstand* гарантирует, что измененная семантика не будет молчаливо (и скорее всего ошибочно) проигнорирована не понимающим ее получателем.

Данный атрибут действителен только в сообщении SOAP (см. разделы 3 и 4.2.1),

4.3 Тело сообщения SOAP

Элемент SOAP *Body* реализует механизм обмена обязательной информацией, адресованной конечному получателю сообщения. Типичные примеры использования элемента *Body* — маршалинг удаленных вызовов и передача сообщений об ошибках.

Элемент *Body* — прямой потомок элемента SOAP *Envelope.* Если присутствует элемент *Header,* элемент *Body* ОБЯЗАН следовать сразу за элементом *Header,* а в противном случае ДОЛЖЕН быть первым прямым потомком элемента *Envelope.*

Все потомки элемента *Body* называются элементами тела сообщения. Их синтаксис описывается следующими правилами.

1. Элемент тела идентифицируется полным именем, состоящим из идентификатора пространства имен и имени элемента. Название прямых потомков элемента *Body* МОЖЕТ включать идентификатор пространства имен.
2. Для описания правил представления элементов тела **МОЖЕТ** использоваться атрибут *encodingStyle (см.* раздел 4.1.1).

Спецификация SOAP определяет один элемент тела сообщения — *Fault,* предназначенный для передачи сообщений об ошибках (см. раздел 4.4).

4.3.1 Связь элементов Header и *Body*

Хотя элементы *Header и Body* определены независимо, на деле они связаны: элемент тела семантически эквивалентен элементу заголовка с атрибутом *mustUnderstand*=«1», предназначенному получателю по умолчанию. Получатель по умолчанию указывается отсутствием атрибута *actor* (см. раздел 4.2.2).

4.4 Элемент *Fault*

Элемент *Fault* служит для передачи сообщений об ошибках и/или информации о статусе сообщения SOAP. Этот элемент ДОЛЖЕН быть размещен в теле сообщения и НЕ ДОЛЖЕН появляться в элементе *Body* более одного раза.

Элемент *Fault* содержит четыре вложенных элемента.

- Элемент *faultcode* предназначен для приложений; он служит для алгоритмической реализации выявления ошибок. Элемент *faultcode* ДОЛЖЕН присутствовать в элементе *Fault,* а его значением ДОЛЖНО быть полное имя согласно определению (см. [8], раздел 3). Спецификация SOAP определяет несколько кодов основных ошибок SOAP (см. раздел 4.4.1).
- Элемент *faultstring* содержит описание ошибки и не предназначен для алгоритмической обработки. По своему назначению этот эле-

мент аналогичен заголовку HTTP 'Reason-Phrase' (см. [5], раздел 6.1). Это элемент ДОЛЖЕН присутствовать в элементе *Fault*; РЕКОМЕНДУЕТСЯ, чтобы он содержал хотя бы минимум информации о причине ошибки.

- Элемент *faultactor* указывает виновника ошибки, т. е. одно из приложений по пути следования сообщения (см. раздел 2). По своему назначению этот элемент аналогичен атрибуту SOAP *actor* (см. раздел 4.2.2), но указывает не получателя элемента заголовка, а источник ошибки. Значение элемента *faultactor*— универсальный идентификатор источника ошибки. Приложение, не являющееся конечным получателем SOAP-сообщения, ОБЯЗАНО включить элемент *faultactor* в SOAP-элемент *Fault*. Конечный получатель сообщения МОЖЕТ, используя элемент *faultactor,* явно указать, что именно он вернул сообщение об ошибке (см. также описание элемента *detail* ниже).
- Элемент *detail* служит для передачи прикладной информации об ошибке в теле сообщения. Этот элемент ДОЛЖЕН присутствовать, если содержание элемента *Body* не может быть обработано. Этот элемент НЕ ДОЛЖЕН использоваться для передачи информации об ошибке в элементах заголовка — она ДОЛЖНА пересылаться в элементах заголовка.

Отсутствие элемента *detail* в элементе *Fault* означает, что ошибка не связана с обработкой элемента *Body*. В случае ошибки так можно указать, было ли обработано тело сообщения или нет.

Все прямые потомки элемента *detail* называются элементами описания ошибки; их синтаксис описывается следующими правилами:

1. элемент описания идентифицируется полным именем, состоящим из идентификатора пространства имен и имени элемента; название прямых потомков элемента *detail* МОЖЕТ включать идентификатор пространства имен;
2. для описания правил представления элементов МОЖЕТ применяться атрибут *encodingStyle* (см. раздел 4.1.1).

В элементе *Fault* ДОПУСКАЕТСЯ присутствие других вложенных элементов с обязательным указанием пространства имен.

4.4.1 Коды ошибок SOAP

Коды ошибок, определенные ниже, ДОЛЖНЫ использоваться в элементе *faultcode* при описании ошибок, определенных в настоящей спецификации. Идентификатор пространства имен значений кодов — *http://schemas.xmlsoap.org/soap/envelope/*. Это пространство рекомендуется применять при описании методов, не входящих в данную спецификацию.

Значения кодов ошибок SOAP по умолчанию определены так, чтобы обеспечить возможность добавления новых кодов без нарушения обратной совместимости. Этот механизм аналогичен базовым классам состояния 1xx, 2xx, 3xx и т. д., определенным в спецификации (см. [5], раздел 10). Однако вместо целых чисел в настоящей спецификации использованы полные имена в смысле **XML** (см. [8], раздел 3). Символ «.» («точка») служит разделителем значений кодов ошибок, перечисленных в порядке уточнения. Например:

```
Client.Authentication
```

В настоящем документе определены следующие значения элемента *faultcode*:

Значение	Смысл
VersionMismatch	Обрабатывающее приложение обнаружило некорректное пространство имен в элементе SOAP *Envelope* (см. раздел 4.1.2).
MustUnderstand	Прямой потомок элемента SOAP *Header* с атрибутом *mustUnderstand*=«1» (см. раздел 4.2.3) не был понят или обработан.
Client	Этот класс ошибок указывает, что сообщение некорректно или не содержит информации, требуемой для обработки (например, аутентификационной информации). Обычно этот код свидетельствует об отсутствии необходимости повторной отправки сообщения без устранения ошибки (см. также описание вложенного элемента *detail* элемента SOAP Fault в разделе 4.4).
Server	Этот класс служит для сообщения об ошибках обработки, не связанных с содержимым сообщения, — например, если в обработке сообщения должно участвовать еше одно приложение, с которым не удалось связаться, Сообщение этого класса обычно свидетельствует о том, что сообщение может быть обработано при повторной отправке (см. также описание вложенного элемента *detail* элемента SOAP Fault в разделе 4.4).

5. Представление данных в SOAP

Правила представления данных в SOAP основаны на простой системе типов, обобщающей системы типов данных языков программирования и систем управления базами данных. Данные в SOAP могут

быть простого (скалярного) или составного типа. Последний представляет собой композицию типизированных частей; составные типы подробнее описаны ниже. Ниже описаны правила сериализации дерева графа типизированных объектов. Она подразделяется на два уровня. Первый: построение XML-словаря по схеме типов данных (совместимой с описываемой). Второй: построение документа XML по схеме типов данных и конкретному графу значений. Правила обеспечивают и возможность выполнения обратной процедуры — восстановления исходного графа значений по XML-документу, созданному в соответствии с ними, и схеме,

Определенные ниже элементы и атрибуты относятся к пространству имен *http://schemas.xmlsoap.org/soap/encoding*Во всех примерах этого раздела предполагается, что все необходимые пространства имен объявлены в объемлющем элементе,

Использование описанной здесь модели и правил представления данных приветствуется, но не является обязательным; в сочетании с SOAP могут применяться и другие модели данных (см. раздел 4.1.1).

5.1 Представление типов данных в XML

XML обладает очень гибким синтаксисом представления данных, а правила SOAP более узки. Ниже определены общие правила представления данных; детальное описание правил кодирования для конкретных типов описаны в следующих разделах. Описанные здесь правила представления данных можно применять при отображении параметров удаленных вызовов и их результатов, описанных в разделе 7.

При определении правил используется следующая терминология.

1. *Значение* — это строка, название типа (число, дата, список и т. д.) или набор таких примитивов; каждое значение должно иметь тип.
2. *Простое значение* — значение без именованных компонентов: конкретная строка, целое, список и т. д,
3. *Составное значение* — агрегат нескольких значений: адрес, заказ, массив значений и т. п.
4. *Отдельное значение* в составе составного значения идентифицируется *аксессором* ~ именем. порядковым номером или и тем, и другим. Имена аксессоров могут совпадать (как, например, в RDF).
5. *Массив* — составное значение, составляющие которого идентифицируются только порядковым номером.
6. *Структура* — составное значение, составляющие которого идентифицируются только (уникальными) именами.
7. *Простой тип* — класс простых значений: «string» (строка), «integer» (целое): перечисление и т. п.

8. *Составной тип* — класс составных значений, например, класс компонентов заказа с одинаковым именем (shipTo, totalCost и т. д.). но, возможно, разными значениями (с возможными ограничениями на допустимые значения).
9. Если имя аксессора уникально лишь в рамках охватывающего составного типа (т. е. для глобально уникальной идентификации требуются и имя аксессора, и имя составного типа), он называется аксессором с *локальной областью действия.* Если же имя — универсальный идентификатор ресурса, однозначно идентифицирующий аксессор независимо от типа, в котором он используется, аксессор называется *универсальным.*
10. С помощью схемы значения можно разделить на *одиночные* (в графе одна и только одна ссылка на значение) и *множественные* (несколько ссылок). Отметим: одиночные по отношению к одной схеме значения могут быть множественными по отношению к другой.
11. Синтаксически элемент может быть *независимым* или *вложенным.* Независимым считается любой элемент на верхнем уровне графа; остальные элементы считаются вложенными.

Применение атрибута *xsi:type* позволяет сделать граф значений самодокументированным в отношении как структуры, так и типов значений, однако правила сериализации допускают определение типов значений только на основе ссылок на схему. Схема может иметь синтаксис, определенный документами «XML-схемы. Часть 1: структура» [10] и «XML-схемы. Часть 2: типы данных» [11]; ДОПУСТИМ и другой синтаксис. Хотя правила сериализации применимы к любым составным типам, большинство схем ограничивает набор составных типов массивами и структурами.

Правила сериализации таковы.

1. Все значения хранятся в содержании элементов. Множественное значение ДОЛЖНО содержаться в независимом элементе, Одиночное значение НЕ ОБЯЗАНО (но МОЖЕТ) содержаться в независимом элементе.
2. Если элемент содержит значение, тип этого значения ДОЛЖЕН быть описан хотя бы одним из следующих способов: а) охватывающий элемент содержит атрибут *xsi:type*; б) охватывающий элемент сам входит в элемент с атрибутом *SOAP-ENC:arrayType* (возможно, по умолчанию); в) имя элемента однозначно связано с типом, так что тип определяется по схеме.
3. Простые значения представляются символьными данными без вложенных элементов. Каждое простое значение должно содержать описание типа; в этом качестве может выступать любой из

типов, определенных во второй части спецификации XML-схем [11], или любой тип, производный от них (см. также раздел 5.2).

4. Составное значение представляется последовательностью элементов; каждый аксессор представляется вложенным элементом, имя которого совпадает с именем аксессора. Названия элементов для аксессоров с локальной областью действия могут быть неполными; все остальные имена должны быть полными (см. также раздел 5.4).
5. Множественное простое или составное значение представляется независимым элементом, который должен содержать атрибут с именем «id» типа ID (см. спецификацию XML [7]). Аксессоры для таких значений представляют собой пустой элемент типа *uri-reference* с локальным именем *href* (см. спецификацию XML-схем [11]); значение атрибута *href*— идентификатор фрагмента со ссылкой на соответствующий независимый элемент.
6. Строковые и байтовые массивы представляются как множественные простые типы; дополнительные правила (см. раздел 5.2.1 и 5.2.3) обеспечивают эффективное представление данных в большинстве типичных случаев. Аксессор для строкового или байтового массива МОЖЕТ иметь атрибут типа ID с именем «id» (см, спецификацию XML [7]). В этом случае все другие аксессоры для этого значения ДОЛЖНЫ представлять собой пустой элемент типа *uri-reference* с локальным именем *href* (см. спецификацию XML-схем [11]); значение атрибута *href*— идентификатор фрагмента со ссылкой на элемент, содержащий значение.
7. Множественные ссылки на одно значение как на разные ДОПУСКАЮТСЯ, только если контекст гарантирует неизменность значения в XML-документе.
8. Массивы относятся к составному типу (см. также раздел 5,4.2). Массивы SOAP относятся к типу *SOAP-ENC:Array* или его производным.

   Массив SOAP имеет одну или несколько размерностей; члены массива идентифицируются порядковым номером. Значение массива представляется как последовательность элементов в порядке возрастания индекса. Для многомерных массивов сначала пробегает все значения последний индекс, затем — предпоследний и т. д."* Каждый элемент именуется независимо (см. правило 2).

   Массивы SOAP могут быть одиночными и множественными значениями и, как следствие, могут представляться вложенными или независимыми элементами.

* Иначе говоря, двумерные массивы записываются построчно.— *Прим. ред.*

Массив SOAP ДОЛЖЕН содержать атрибут *SOAP-ENC:arrayType*, значение которого задает тип элементов массива и его размерность (размерности). Значение *SOAP-ENC:arrayType* задается так:

```
arrayTypeValue = atype asize
atype = QName *( rank )
rank = "[" *( "," ) "]"
asize = "[" #length "]"
length = 1*DIGIT
```

Конструкция *atype* задает тип элементов в виде выражения QName (так же, как в атрибуте *type* объявления элемента XML-схемы) и действует как ограничение типа, т. е. все значения элементов должны быть указанного типа (иначе говоря, все члены массива должны иметь тип, указанный в атрибуте *SOAP-ENC:arrayType* или производный от него). В случае массива массивов сначала задается тип, за которым следует перечисление размерностей (начиная с 1). Роль разделителя размерностей в представлении многомерных массивов играет запятая.

Конструкция *asize* содержит список (через запятую) целых чисел, задающих размерность массива по каждому измерению. Значение О показывает, что размер можно определить после изучения членов массива. Например, в случае массива из 5 членов типа «массив целых чисел» атрибут *arrayTypeValue* будет иметь значение «int[][5]»; здесь атрибут *atype* имеет значение «int[]», а *asize* — «[5]». Точно так же в случае массива из 3 членов типа «двумерный массив целых чисел» *array Type Value*=«int[,][3]», *atype*=«int[,]», *as-size*=«[3]».

Член SOAP-массива МОЖЕТ содержать атрибут *SOAP-ENC:offset*, задающий его смещение от начала массива, что можно использовать для представления части массива (см. раздел 5.4.2.1). Другой аналогичный необязательный атрибут члена массива SOAP - *SOAP-ENC:position* — задает его положение в массиве, что можно использовать для представления разреженных массивов (см. раздел 5.4.2.2). Значение атрибутов *SOAP-ENC:offset* и *SOAP-ENC:position* задается так:

```
arrayPoint = "[" fllength "]"
```

Смещение и положение отсчитываются от 0.

Значение NULL и значение по умолчанию МОЖЕТ представляться отсутствием соответствующего элемента. NULL, кроме того,

МОЖНО представить элементом с атрибутом *xsi:null* со значением '1'.

Правило 2 допускает имена независимых элементов и элементов, представляющих массивы, не идентичные типу представленных в них значений.

5.2 Простые типы

В качестве простых типов SOAP допустимы все типы данных, определенные в разделе «Встроенные типы данных» спецификации «XML-схемы. Часть 2: типы данных» [11]. Примеры простых типов:

Тип	Пример значения
Int	58502
Float	314159265358979E+1
negative Integer	-32768
string	Луис «Сачмо» Армстронг

Типы данных, определенные в спецификации XML-схем, и их производные можно использовать в схемах элементов. Ниже приведен пример фрагмента схемы и фрагмент данных соответствующих типов.

```
<element name="age" type="int"/>
<element name="height" type="float"/>
<element name="displacement" type="negativeInteger"/>
<element name="color">
 <simpleType base="xsd:string">
 <enumeration value="Green"/>
 <enumeration value="Blue"/>
 </simpleType>
</element>
<age>45</age>
<height>5.9</height>
<displacement>-450</displacement>
<color>Blue</color>
```

Простые значения ДОЛЖНЫ быть представлены как содержимое элементов, тип которых определен в спецификации «XML-схемы. Часть 2: типы данных» [11] или является производным от этих типов (механизм построения производных типов описан в спецификации XML-схем).

Если простое значение представлено как независимый элемент или как член гетерогенного массива, часто удобно иметь определение элемента для соответствующего типа данных. Поскольку в спецификации XML-схем [11] определены лишь типы (но не элементы), схема SOAP-ENC и соответствующее пространство имен содержат объявления элементов для каждого простого типа данных, которые МОЖНО использовать в описанной выше ситуации. Приведем пример:

```
<SOAP-ENC:int id="int1">45</SOAP-ENC:int>
```

5.2.1 Строки

Тип данных *string* определен в документе «XML-схемы. Часть 2: типы данных» [11]. Его определение отличается от определения строчного типа в большинстве языков программирования и систем управления базами данных; в частности, определение ограничивает набор символов, допустимых в переменных этого типа. Для представления значений, содержащих такие символы, следует использовать другие типы данных. Строка МОЖЕТ быть представлена как одиночное или множественное значение.

Охватывающий элемент строчного значения **МОЖЕТ** иметь атрибут *id,* а дополнительные аксессоры — соответствующие атрибуты *href.*

Например, вот как можно организовать две ссылки на одну строку:

```
<greeting id="String-0">Hello</greeting>
<salutation href="#String-0"/>
```

Однако если наличие двух ссылок на одно строковое значение не имеет решающего значения, предыдущий пример можно записать как два одиночных значения (значений с одной ссылкой):

```
<greeting>Hello</greeting>
<salutation>Hello</salutation>
```

Соответствующие элементы схемы для этих примеров могут выглядеть так:

```
<element name="greeting" type="SOAP-ENC:string"/>
<element name="salutation" type="SOAP-ENC:string"/>
```

В этом примере тип *SOAP-ENC:string* задействован в качестве типа элемента для удобства объявления элемента типа данных *xsd:string,* допускающего атрибуты *id* и *href.*

5.2.2 Списки

Спецификация «XML-схемы. Часть 2: типы данных» [11] определяет механизм *списков* (enumeration). Модель данных SOAP заимствовала этот механизм без изменений. Однако поскольку в языках программирования этот тип данных определен несколько иначе, в этом разделе мы подробно опишем этот механизм и покажем, как представляются данные, являющиеся значением из списка.

Список представляет собой набор различных имен. Конкретный список — это список разных значений одного базового типа. Например, набор названий цветов («Зеленый», «Синий», «Коричневый») можно определить как список строковых значений. Значения («1», «3», «5») можно рассматривать как список значений целого типа и т. д. Спецификация «XML-схемы. Часть 2: типы данных» [11] допускает списки значений всех базовых типов, кроме *boolean*. Для объявления списков применяется синтаксис, описанный в спецификации «XML-схемы. Часть 1: структуры» [10]. Если схема создается на другой основе, не содержащей нужного базового типа, надо использовать тип *string*. Ниже значение «EyeColor» (цвет глаз) объявляется как строка с допустимыми значениями из списка «Green», «Blue», «Brown» («Зеленый», «Синий», «Коричневый»):

```
<element name="EyeColor" type="tns:EyeColor"/>
<simpleType name="EyeColor" base="xsd:string">
 <enumeration value="Green"/>
 <enumeration value="Blue"/>
 <enumeration value="Brown"/>
</simpleType>
<Person>
 <Name>Henry Ford</Name>
 <Age>32</Age>
 <EyeColor>Brown</EyeColor>
</Person>
```

5.2.3 Байтовый массив

Массив байтов МОЖЕТ представляться как одиночное или множественное значение. Правила представления байтовых массивов аналогичны правилам для строк.

В частности, охватывающий элемент байтового массива МОЖЕТ иметь атрибут *id*. В этом случае дополнительные ссылки на элемент МОГУТ содержать соответствующий атрибут *href*.

В качестве стандартного представления байтовых массивов рекомендуется кодировка base64, определенная в спецификации XML-схем [10, 11], использующая алгоритм base64 [13]. Принятое в спецификации MIME ограничение на длину строки данных в кодировке base64 в SOAP не применяется. Кроме того, в спецификации SOAP определен тип *SOAP-ENC:base64*:

```
<picture xsi:type="SOAP-ENC:base64">
 aG93IG5vDyBicm73biBb3cNCg==
</picture>
```

5.3 Полиморфные ссылки

Многие языки определяют полиморфные ссылки, позволяющие (во время выполнения) ссылаться на значения разных типов. Полиморфная ссылка ДОЛЖНА содержать атрибут *xsi:type*, описывающий тип фактического значения.

Например, полиморфная ссылка с именем cost и значением типа *xsd:float* может быть представлена так:

```
<cost xsi:type="xsd:float">29.95</cost>
```

Одноименная ссылка с инвариантным типом выглядит иначе:

```
<cost>29.95</cost>
```

5.4 Составные типы

SOAP определяет типы для двух широко распространенных в языках программирования структур данных:

- Struct (структура) — составное значение, компоненты которого идентифицируются только именами, причем никакие два имени не совпадают;
- **Array** (массив) — составное значение, компоненты которого идентифицируются только своим положением (порядковым номером).

Спецификация SOAP допускает также представление данных, не являющихся ни массивами, ни структурами, — например, ориентированных помеченных графов. Правила представления данных SOAP не предполагают упорядочения элементов таких данных, но если модель данных предполагает упорядочение, то элементы ДОЛЖНЫ быть представлены в соответствующем порядке.

5.4.1 Составные значения, структуры и ссылки на значения

Компоненты составного значения представляются набором элементов. Если компоненты идентифицируются именами (например, в

структуре), имя элемента должно совпадать с именем компонента. Элементы, представляющие локальные компоненты, могут иметь сокращенные названия; имена остальных элементов должны быть указаны полностью, включая пространство имен.

Пример представления структуры типа «Book»:

```
<e:Book>
 <author>Henry Ford</author>
 <preface>Prefatory text</preface>
 <intro>This is a book.</intro>
</e:Book>
```

Вот схема для приведенного выше фрагмента документа:

```
<element name="Book">
<complexType>
 <element name="author" type="xsd:string"/>
 <element name="preface" type="xsd:string"/>
 <element name="intro" type="xsd:string"/>
</complexType>
</e:Book>
```

Следующий пример иллюстрирует тип с простыми и составными компонентами. Заметьте: атрибут *href* элемента *Author* содержит ссылку на элемент с соответствующим значением атрибута *id*. Похожая конструкция использована и в компоненте *Address:*

```
<e:Book>
 <title>My Life and Work</title>
 <author href="#Person-1"/>
</e:Book>
<e:Person id="Person-1">
 <name>Henry Ford</name>
 <address href="#Address-2"/>
</e:Person>
<e:Address id="Address-2">
 <email>mailto:henryford@hotmail.com</email>
 <web>http://www.henryford.com</web>
</e:Address>
```

Приведенная выше форма соответствует случаю наличия нескольких ссылок на значения компонентов *Person* и *Address*. Если они используются только один раз, их представления ДОЛЖНЫ быть вложенными:

```
<e:Book>
 <title>My Life and Work</title>
 <author>
 <name>Henry Ford</name>
 <address>
 <email>mailto:henryford@hotmail.com</email>
 <web>http://www.henryford.com</web>
 </address>
 </author>
</e:Book>
```

Если, напротив, никакие два адресата не могут иметь одного адреса (а таковым может быть и почтовый, и электронный адрес), сведения о книге с двумя авторами будут представлены так:

```
<e:Book>
 <title>My Life and Work</title>
 <firstauthor href="#Person-1"/>
 <secondauthor href="#Person-2"/>
</e:Book>
<e:Person id="Person-1">
 <name>Henry Ford</name>
 <address xsi:type="m:Electronic-address">
 <email>mailto:henryford@hotmail.com</email>
 <web>http://www.henryford.com</web>
 </address>
</e:Person>
<e:Person id="Person-2">
 <name>Samuel Crowther</name>
 <address xsi:type="n:Street-address">
 <street>Martin Luther King Rd</street>
 <city>Raleigh</city>
 <state>North Carolina</state>
 </address>
</e:Person>
```

Представление данных МОЖЕТ содержать ссылки на внешние данные:

```
<e:Book>
 <title>Paradise Lost</title>
 <firstauthor href="http://www.dartmouth.edu/~milton/"/>
</e:Book>
```

Ниже приведен фрагмент схемы для данных предыдущего примера:

```
<element name="Book" type="tns:Book"/>
<complexType name="Book">
 <!- Either the following group must occur or else the
 href attribute must appear, but not both. ->
 <sequence minOccurs="0" maxOccurs="1">
 <element name="title" type="xsd:string"/>
 <element name="firstauthor" type="tns:Person"/>
 <element name="secondauthor" type="tns:Person"/>
 </sequence>
 <attribute name="href" type="uriReference"/>
 <attribute name="id" type="ID"/>
 <anyAttribute namespace="##other"/>
</complexType>
<element name="Person" base="tns:Person"/>
<complexType name="Person">
 <!- Either the following group must occur or else the
 href attribute must appear, but not both. ->
 <sequence minOccurs="0" maxOccurs="1">
 <element name="name" type="xsd:string"/>
 <element name="address" type="tns:Address"/>
 </sequence>
 <attribute name="href" type="uriReference"/>
 <attribute name="id" type="ID"/>
 <anyAttribute namespace="##other"/>
</complexType>
<element name="Address" base="tns:Address"/>
<complexType name="Address">
 <!- Either the following group must occur or else the
 href attribute must appear, but not both. ->
 <sequence minOccurs="0" maxOccurs="1">
 <element name="street" type="xsd:string"/>
 <element name="city" type="xsd:string"/>
 <element name="state" type="xsd:string"/>
 </sequence>
 Ottribute name="href" type="uriReference"/>
 <attribute name="id" type="ID"/>
 <anyAttribute namespace="##other"/>
</complexType>
```

5.4.2 Массивы

Массивы SOAP по определению имеют тип «SOAP-ENC:Array» или производный от него (см. также правило 8). Массивы кодируются как значения элементов без ограничений на имя охватывающего элемента (значения элемента, как правило, не влияют на выбор его имени),

Элементы массива могут иметь любой тип; в частности, допустимы вложенные массивы. С другой стороны, ограничивая тип *SOAP-ENC:Array*, можно получить новые типы массивов — скажем, для хранения только целых чисел или только значений из определенного списка.

Массив представляется как упорядоченная последовательность своих элементов. Члены массива в этом представлении не идентифицируются именами; соответствующие элементы могут иметь любое имя. На практике имя элемента часто подбирается согласно требованиям схемы или из соображений описания типа массива. Как и в составных значениях, элемент массива, используемый один раз, хранится в значении; если ссылок на элемент массива несколько, элемент содержит ссылку на значение с помощью атрибута *href*.

Ниже приведены фрагмент схемы и образец данных массива, содержащего целые элементы:

```
<element name="myFavoriteNumbers"
 type="SOAP-ENC:Array"/>
<myFavoriteNumbers
 SOAP-ENC:arrayType="xsd:int[2]">
 <number>3</number>
 <number>4</number>
</myFavoriteNumbers>
```

Здесь массив *my Favorite Numbers* содержит несколько элементов типа *SOAP-ENC:int*. Тип элемента указан в атрибуте *SOAP-ENC:array Type,* Помните: тип *SOAP-ENC:Array* допускает неполные имена элементов. Поскольку имя такого элемента не несет информации о типе значения, он либо должен быть снабжен атрибутом *xsi:type*, либо охватывающий элемент должен иметь атрибут SOAP-*ENC:arrayType* Типы, производные от *SOAP- ENC:Array*, могут содержать определение локальных элементов с указанием типа.

Как уже отмечалось, схема SOAP-ENC содержит определение соответствующих элементов для каждого простого типа, определенного в спецификации «XML-схемы. Часть 2: типы данных» [11]. Там же определен элемент *Array*. С его помощью предыдущий пример можно переписать так:

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:int[2]">
 <SOAP-ENC:int>3</SOAP-ENC:int>
 <SOAP-ENC:int>4</SOAP-ENC:int>
</SOAP-ENC:Array>
```

Массив может содержать значения любого типа, вложенного в тип *arrayType*. Точнее, элементы массива могут иметь любой тип, который может заменять тип, указанный в атрибуте *arrayType;* правила замены задаются в схеме. Так, массив целых чисел может содержать значения любого типа, производного от *integer* (скажем, int или любого пользовательского целого типа). Аналогично массив типа *address* может содержать значения суженного или расширенного типа *internationalAddress*. Поскольку встроенный тип *SOAP-ENC:Array* допускает значения любого типа, массив может содержать элементы разных типов, если это явным образом не запрещено значением атрибута *arrayType*.

В этом примере двумерного массива тип значений элементов может быть указан атрибутом *xsi:type* в документе или в объявлении элементов в схеме:

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:ur-type[4]">
 <thing xsi:type="xsd:int">12345</thing>
 <thing xsi:type="xsd:decimal">6.789</thing>
 <thing xsi:type="xsd:string">
 Of Mans First Disobedience, and the Fruit
 Of that Forbidden Tree, whose -mortal tast
 Brought Death into the World, and all our woe.
 </thing>
 <thing xsi:type="xsd:uriReference">
 http://www.dartmouth.edu/~milton/reading_room/
 </thing>
</SOAP-ENC:Array>
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:ur-type[4]">
 <SOAP-ENC:int>12345</SOAP-ENC:int>
 <SOAP-ENC:decimal>6.789</SOAP-ENC:decimal>
 <xsd:string>
 Of Mans First Disobedience, and the Fruit
 Of that Forbidden Tree, whose mortal tast
 Brought Death into the World, and ail our woe,
 </xsd:string>
 <SOAP-ENC:uriReference>
 http://www.dartmouth.edu/~milton/reading_room/
```

```
 </SOAP-ENC:uriReference >
</SOAP-ENC:Array>
```

В качестве значений элементов массива могут выступать структуры и другие составные значения. Пример массива структур *xyz:Order*:

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xyz:Order[2]">
 <Order>
 <Product>Apple</Product>
 <Price>1.56</Price>
 </Order>
 <Order>
 <Product>Peach</Product>
 <Price>1.48</Price>
 </Order>
</SOAP-ENC:Array>
```

В качестве элемента массива может выступать другой массив. Пример массива из двух массивов строк:

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:string[][2]">
 <item href="#array-1"/>
 <item href="#array-2"/>
</SOAP-ENC:Array>
<SOAP-ENC:Array id="array-1" SOAP-ENC:arrayType="xsd:string[2]">
 <item>r1c1</item>
 <item>r1c2</item>
 <item>r1c3</item>
</SOAP-ENC:Array>
<SOAP-ENC:Array id="array-2" SOAP-ENC:arrayType="xsd:string[2]">
 <item>r2c1</item>
 <item>r2c2</item>
</SOAP-ENC:Array>
```

Элемент, содержащий значение массива, НЕ ОБЯЗАН называться *SOAP-ENC:Array*. Его имя может быть произвольным; нужно лишь, чтобы элемент имел тип *SOAP-ENC:Array* или был производным от типа *SOAP-ENC:Array* (с сужением), как в приведенном ниже примере.

```
<simpleType name="phoneNumber" base="string"/>
<element name="ArrayOfPhoneNumbers">
 <complexType base="SOAP-ENC:Array">
 <element name="phoneNumber" type="tns:phoneNumber"
```

```
 maxOccurs="unbounded"/>
 </complexType>
 <anyAttribute/>
</element>
<xyz:ArrayOfPhoneNumbers SOAP-ENC:arrayType="xyz:phoneNumber[2]">
 <phoneNumber>206-555-1212</phoneNumber>
 <phoneNumber>1-888-123-4567</phoneNumber>
</xyz:ArrayOfPhoneNumbers>
```

Массивы могут быть многомерными. В этом случае в аргументе *asize* атрибута *arrayType* указывается несколько размерностей:

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:string[2,3]">
 <item>r1c1</item>
 <item>r1c2</item>
 <item>r1c3</item>
 <item>r2c1</item>
 <item>r2c2</item>
 <item>r2c3</item>
</SOAP-ENC:Array>
```

Хотя в приведенных выше примерах массивы представлены в независимых элементах, они МОГУТ представляться вложенными элементами; это ОБЯЗАТЕЛЬНО, если элементы массива упоминаются лишь один раз.

Ниже приведен фрагмент схемы и массив телефонных номеров, вложенный в структуру типа *Person*; доступ к массиву осуществляется по имени *phone-numbers*:

```
<simpleType name="phoneNumber" base="string"/>
<element name="ArrayOfPhoneNumbers">
 <complexType base="SOAP-ENC:Array">
 <element name="phoneNumber" type="tns:phoneNumber"
 maxOccurs="unbounded"/>
 </complexType>
 <anyAttribute/>
</element>
<element name="Person">
 <complexType>
 <element name="name" type="string"/>
 <element name="phoneNumbers" type="tns:ArrayOfPhoneNumbers"/>
 </complexType>
```

```
</element>
<xyz:Person>
 <name>John Hancock</name>
 <phoneNumbers SOAP-ENC:arrayType="xyz:phoneNumber[2]">
 <phoneNumber>206-555-1212</phoneNumber>
 <phoneNumber>1-888-123-4567</phoneNumber>
 </phoneNumbers>
</xyz:Person>
```

Вот еще пример одиночного массива, представленного вложенным элементом; доступ к массиву осуществляется по имени охватывающего элемента.

```
<xyz:PurchaseOrder>
 <CustomerName>Henry Ford</CustomerName>
 <ShipTo>
 <Street>5th Ave</Street>
 <City>New Yorк</City>
 <State>NY</State>
 <Zip>10010</Zip>
 </ShipTo>
 <PurchaseLineItems SOAP-ENC arrayType="Order[2]">
 <Order>
 <Product>Apple</Product>
 <Price>1.56</Price>
 </Order>
 <Order>
 <Product>Peach</Product>
 <Price>1.48</Price>
 </Order>
 </PurchaseLineItems>
</xyz:PurchaseOrder>
```

5.4.2.1 **Часть массива**

SOAP позволяет пересылать часть массива (иногда их называют массивами переменной длины [12]). В этом случае атрибут *SOAP-ENC:offset* задает смещение первого элемента. Смещение по умолчанию равно 0.

Пример передачи третьего и четвертого (считая от 0) элементов массива длины 5:

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:string[5]" SOAP
ENC:offset="[2]">
```

```
 <item>The third element</item>
 <item>The fourth element</item>
</SOAP-ENC:Array>
```

5.4.2.2 Разреженные массивы

Спецификация SOAP предусматривает поддержку разреженных массивов. Каждый элемент, представляющий член массива, содержит атрибут *SOAP-ENC:position*,указывающий положение члена массива. Ниже приведен пример разреженного массива двумерных массивов строк. Размер массива равен 4, но передается только второй элемент:

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:string[,][4]">
 <SOAP-ENC:Array href="#array-1" SOAP-ENC:position="[2]"/>
</SOAP-ENC:Array>
<SOAP-ENC:Array id="array-1" SOAP-ENC:arrayType="xsd:string[10,10]">
 <item SOAP-ENC:position="[2,2]">Third row, third col</item>
 <item SOAP-ENC:position="[7,2]">Eighth row, third col</item>
</SOAP-ENC:Array>
```

Если бы в охватывающем массиве была ссылка только на массив array-1, предыдущий пример можно было бы переписать так:

```
<SOAP-ENC:Array SOAP-ENC:arrayType="xsd:string[,][4]">
 <SOAP-ENC:Array SOAP-ENC:position="[2]" SOAP
ENC:arrayType="xsd:string[10,10]>
 <item SOAP-ENC:position="[2,2]">Third row, third col</item>
 <item SOAP-ENC:position="[7,2]">Eighth row, third col</item>
 </SOAP-ENC:Array>
</SOAP-ENC:Array>
```

5.4.3 Другие составные типы

Правила представления не ограничиваются составными типами, идентификаторы компонентов которых известны заранее. Правила применимы и в том случае, когда для выяснения идентификаторов компонентов составного типа необходимо изучение фактических значений; компонент кодируется элементом, имя которого совпадает с именем компонента, а сам он содержит значение компонента или ссылку на него. Компоненты, тип которых нельзя определить заранее, ДОЛЖНЫ содержать соответствующий атрибут *xsi:type*, указывающий тип фактического значения.

Точно так же описанные выше правила достаточны для представления составных типов, содержащих компоненты, идентифицируе-

мые именами, и компоненты, идентифицируемые именами и порядковыми номерами.

```
<xyz:PurchaseOrder>
 <CustomerName>Henry Ford</CustomerName>
 <ShipTo>
 <Street>5th Ave</Street>
 <City>New York</City>
 <Sta e>NY</State>
 <Zip>10010</Zip>
 </ShipTo>
 <PurchaseLineItems>
 <Order>
 <Product>Apple</Product>
 <Price>1.56</Price>
 </Order>
 <Order>
 <Product>Peach</Product>
 <Price>1.48</Price>
 </Order>
 </PurchaseLineItems>
</xyz:PurchaseOrder>
```

Те же правила позволяют представить и составное значение, структурно напоминающее массив, тип которого не совпадает с *SOAP-ENC:Array*. Например:

```
<PurchaseLineItems>
 <Order>
 <Product>Apple</Product>
 <Price>1.56</Price>
 </Order>
 <Order>
 <Product>Peach</Product>
 <Price>1.48</Price>
 </Order>
</PurchaseLineItems>
```

5.5 Значения по умолчанию

Пропущенный элемент может трактоваться как значение по умолчанию, так и отсутствие значения. Какой именно из этих случаев имеет

место, зависит от компонента, метода и фактического значения. Например, пропуск полиморфного компонента обычно трактуется как значение *Null* (хотя точное значение величины *Null* определяется контекстом). Пропуск компонента типа *Boolean* трактуется как *False* или неизвестное значение, а в случае целого типа — «О» или неизвестное значение.

5.6 Атрибут root

Атрибут *root* позволяет пометить как корневые элементы значения, не являющиеся реальными корневыми вершинами графа объекта. Это позволяет разделить представление объекта на несколько частей. Атрибут может иметь значение «1» или «0». Для настоящих корневых вершин графа атрибут имеет значение «1»; то же значение можно присвоить некорневому элементу. «О» явно указывает, что элемент не корневой.

Атрибут *roof* МОЖЕТ присутствовать в любом элементе, вложенном в элемент SOAP *Header* и *Body*. У атрибута нет значения по умолчанию.

6. SOAP и НИР

Этот раздел посвящен использованию SOAP в рамках стандартного и расширенного протокола HTTP (HTTP Extension Framework). Привязки SOAP для HTTP позволяют использовать все преимущества формализма и гибкости SOAP в сочетании с широкой функциональностью HTTP. Передача данных SOAP по HTTP не означает, что SOAP заменяет семантику HTTP; напротив, семантика реализации SOAP поверх HTTP естественно соответствуют семантике HTTP.

В соответствии с моделью обмена HTTP «запрос — ответ» спецификация SOAP определяет параметры запроса для HTTP-запроса и параметры ответа для HTTP-ответа. Промежуточные получатели сообщений SOAP и HTTP — это НЕ одно и то же, а именно: не следует ожидать, что промежуточный получатель HTTP-сообщения (указанный в поле Connection заголовка HTTP) будет проверять или обрабатывать тело сообщения SOAP, передаваемого в HTTP-запросе.

При передаче сообщений SOAP в сообщениях HTTP приложения в соответствии с RFC 2376 [3] ДОЛЖНЫ использовать тип «text/xml».

6.1 HTTP-запрос SOAP

SOAP может использоваться с разными методами запросов HTTP, однако здесь мы коснемся только передачи сообщений SOAP методом HTTP POST (о применении SOAP для удаленного вызова процедур см. раздел 7; об использование расширений HTTP см. раздел 6.3).

6.1.1 Поле заголовка HTTP *SOAPAction*

Поле *SOAPAction* заголовка HTTP-запроса может применяться для описания получателя запроса. Значение атрибута — универсальный идентификатор. Спецификация SOAP не накладывает ограничений на формат, степень детализации или возможность определения идентификатора. Этот заголовок ОБЯЗАТЕЛЕН для всех HTTP-клиентов, выполняющих HTTP-запрос SOAP.

```
soapaction = "SOAPAction" ":" [ <"> URI-reference <"> ]
URI-reference = <as defined in RFC 2396 [4]>
```

Наличие и содержание поля *SOAPAction* могут использовать серверы (например, брандмауэры) для фильтрации сообщений SOAP, передаваемых по HTTP. Пустое значение заголовка («») указывает, что назначение сообщения SOAP совпадает со значением поля Request-URI заголовка HTTP. Отсутствие значения трактуется как неуказанный получатель.

Примеры:

```
SOAPAction: "http://electrocommerce.org/abc#MyMessage"
SOAPAction: "myapp.sdl"
SOAPAction: ""
SOAPAction:
```

6.2 HTTP-ответ SOAP

Для передачи сообщений о статусе используется семантика кодов состояния HTTP. Например, код 2xx трактуется как подтверждение успешного приема и обработки запроса (включая SOAP-компонент).

В случае ошибки при обработке сообщения SOAP, HTTP-сервер ОБЯЗАН вернуть сообщение с кодом HTTP 500 «Internal Server Error» (Ошибка сервера) и включить в ответ сообщение SOAP с элементом SOAP *Fault* (см. раздел 4.4), указывающим причину ошибки.

6.3 Инфраструктура расширения HTTP

Для описания адресата HTTP-запроса SOAP можно применить средства расширения HTTP (HTTP Extension Framework) [6].

Выбор между стандартным и расширенным HTTP определяется политикой и коммуникационными средствами партнеров. Чтобы сделать применение расширений HTTP обязательным, клиент может воспользоваться обязательным объявлением расширения и префиксом «M-» в именах методов HTTP. Сервер может добиться этого с помощью кода статуса HTTP 510 «Not Extended». Таким образом, один дополнительный цикл обмена сообщениями позволяет каждо-

му из партнеров выяснить политику другой стороны и действовать сообразно ей.

Идентификатор ресурса сообщений SOAP, использующих расширение HTTP, — *http://schemas.xmlsoap.org/soap/envelope/*.

6.4 Примеры **использования SOAP и HTTP**

Пример 3. HTTP-запрос SOAP с помощью метода POST.

```
POST /StockQuote HTTP/1.1
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "http://electrocommerce.org/abc#MyMessage"
<SOAP-ENV:Envelope...
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length; nnnn

<SOAP-ENV:Envelope...
```

Пример 4. Запрос SOAP с использованием инфраструктуры расширения HTTP.

```
H-POST/StockQuote HTTP/1.1
Man: "http://schemas.xmlsoap.org/soap/envelope/"; ns=NNNN
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
NNNN-SOAPAction: "http://electrocommerce.org/abc#MyMessage"
<SOAP-ENV:Envelope...
HTTP/1.1 200 OK
Ext:
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
<SOAP-ENV:Envelope...
```

7, SOAP как средство удаленного вызова методов

Одна из задач SOAP — инкапсуляция и обмен вызовами удаленных методов на основе синтаксиса XML. Ниже описан синтаксис вызовов и ответов при удаленном вызове методов.

Мы предполагаем, что данный синтаксис будет применяться в сочетании с методами представления данных, описанными в разделе 5, однако ДОПУСТИМО и использование других представлений. Ме-

тод представления данных при выполнении вызова или возврате результатов описанным в настоящем разделе способом может быть указан SOAP-атрибутом *encodingStyle* (см. раздел 4.3.2).

Применение SOAP для удаленных вызовов не зависит от транспортного протокола (см. раздел 6). В случае HTTP удаленные вызовы естественным образом отождествляются с HTTP-запросами, а возврат результатов — с HTTP-ответами. И все же применение SOAP для выполнения удаленных вызовов не требует использования в качестве транспортного протокола именно HTTP.

Для выполнения вызова необходима следующая информация:

- универсальный идентификатор вызываемого объекта;
- имя метода;
- сигнатура метода (не обязательно);
- параметры вызова;
- заголовок (не обязательно).

Механизм передачи идентификатора метода определяется транспортным протоколом. Например, в случае HTTP вызываемый объект описывается идентификатором запроса. SOAP не накладывает других ограничений на идентификатор объекта, кроме соответствия синтаксису универсальных идентификаторов (дополнительную информацию о синтаксисе идентификаторов см. в [4]).

7.1 Удаленный вызов процедур и тело сообщения SOAP

Параметры вызова и ответ передаются в элементе SOAP *Body* (см. раздел 4.3) с использованием следующего синтаксиса.

- Вызов метода описывается структурой.
- Параметры описываются в виде структуры, включающей элемент для каждого аргумента. Имя и тип структуры и метода должны совпадать.
- Каждый аргумент рассматривается как член структуры, имя и тип которого совпадают с именем и типом соответствующего аргумента метода; порядок членов структуры должен совпадать с порядком аргументов в сигнатуре метода.
- Результат выполнения вызова моделируется как структура.
- Результат выполнения вызова описывается структурой, включающей элемент для возвращаемого значения и для каждого аргумента. Первый элемент структуры соответствует возвращаемому значению; последующие элементы описывают аргументы в порядке их появления в сигнатуре метода.
- Каждый аргумент рассматривается как член структуры, имя и тип которого совпадают с именем и типом соответствующего аргумен-

та метода; порядок членов структуры должен совпадать с порядком аргументов в сигнатуре метода; имя возвращаемого значения, как и имя структуры, в данном случае несущественно; общее правило — называть структуру именем метода с добавлением строки «Response».

- Сообщение об ошибке при вызове метода описывается в элементе SOAP *Fault (см.* раздел 4.4). Если транспортный протокол предусматривает дополнительные правила сообщения об ошибке, их ОБЯЗАТЕЛЬНО придерживаться.

Как сказано выше, структуры, описывающие вызов метода и результаты выполнения вызова, представляются согласно правилам раздела 5 или другим методом кодирования, указанным в атрибуте *encodingStyle* (см. раздел 4.1.1).

Приложения МОГУТ обрабатывать запросы вызова метода с неполным набором параметров, но МОГУТ и возвращать сообщение об ошибке.

Поскольку возврат результатов выполнения вызова по умолчанию свидетельствует об успехе, метод не должен возвращать одновременно результаты и сообщение об ошибке.

7.2 Удаленный вызов процедур и заголовок сообщения SOAP

Дополнительная информаиия, описывающая представление вызова метода, но не являющаяся частью формальной сигнатуры метода, МОЖЕТ пересылаться в составе RPC-представления. При этом она ДОЛЖНА быть представлена в элементе, вложенном в элемент SOAP *Header.*

В качестве примера такого использования элемента заголовка приведем передачу идентификатора транзакции вместе с сообщением. Поскольку идентификатор транзакции не является частью сигнатуры метода и обычно принадлежит инфраструктурным компонентам, а не коду приложения, прямого способа передачи этой информации в рамках вызова нет. Добавление соответствующего элемента (с фиксированным именем) в заголовок позволит диспетчеру транзакций пункта назначения сообщения извлечь идентификатор транзакции из сообщения без изменения инфраструктуры выполнения удаленного вызова.

8. Вопросы защиты

Методы обеспечения целостности и конфиденциальности сообщений в настоящем документе не рассматриваются. Эти вопросы будут затронуты в следующих версиях настоящей спецификации.

9. Цитированная литература

1. S. Bradner, The Internet Standards Process — Revision 3, RFC2026, Harvard University, October 1996.
2. S. Bradner, Keywords for use in RFCs to Indicate Requirement Levels, RFC 2119, Harvard University, March 1997.
3. E. Whitehead, M. Murata, XML Media Types, RFC2376, UC Irvine, Fuji Xerox Info. Systems, July 1998.
4. T. Berners-Lee, R. Fielding, L. Masinter, Uniform Resource Identifiers (URI): Generic Syntax, RFC 2396, MIT/LCS, U.C. Irvine, Xerox Corporation, August 1998.
5. R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, T. Berners-Lee, Hypertext Transfer Protocol — HTTP/1.1, RFC 2616, U.C. Irvine, DEC W3C/MIT, DEC, W3C/MIT, W3C/MIT, January 1997.
6. H. Nielsen, P. Leach, S. Lawrence, An HTTP Extension Framework, RFC 2774, Microsoft, Agranat Systems.
7. W3C Recommendation «The XML Specification».
8. W3C Recommendation «Namespaces in XML».
9. W3C Working Draft «XML Linking Language» (предварительный вариант).
10. W3C Working Draft «XML Schema Part 1: Structures» (рабочий вариант).
11. W3C Working Draft «XML Schema Part 2: Datatypes» (предварительный вариант).
12. Transfer Syntax NDR, DCE 1.1: Remote Procedure Call.
13. N. Freed, N. Borenstein, Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, RFC2045, Innosoft, First Virtual, November 1996.

A. Примеры конвертов SOAP

A. 1 Пример запроса вызова метода

Пример 5. Аналогичен примеру 1, но с обязательным заголовком.

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

```
<t:Transaction
xmlns:t="some-URI"
SOAP-ENV:mustUnderstand="1">
5
</t:Transaction>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
<m:GetLastTradePrice xmlns:m="Some-URI">
<symbol>DEF</symbol>
</m:GetLastTradePrice>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Пример 6. Аналогичен примеру 1, но с несколькими параметрами.

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length; nnnn
SOAPAction: "Some-URI"
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
<SOAP-ENV:Body>
<m:GetLastTradePriceDetailed
xmlns:m="Some-URI">
<Symbol>DEF</Symbol>
<Company>DEF Corp</Company>
<Price>34.1</Price>
</m:GetLastTradePriceDetailed>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

А.2 Примеры ответов

Пример 7. Аналогичен примеру 2, но с обязательным заголовком.

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
<SOAP-ENV:Envelope
```

```
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
 <SOAP-ENV:Header>
 <t:Transaction
 xmlns:t="some-URI"
 xsi:type="xsd:int" mustUnderstand="1">
 5
 </t:Transaction>
 </SOAP-ENV:Header>
 <SOAP-ENV:Body>
 <m:GetLastTradePriceResponse
 xmlns:m="Some-URI">
 <Price>34.5</Price>
 </m:GetLastTradePriceResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Пример 8. Аналогичен примеру 2, но с использованием структуры данных.

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
 <SOAP-ENV:Body>
 <m:GetLastTradePriceResponse
 xmlns:m="Some-URI">
 <PriceAndVolume>
 <LastTradePrice>
 34.5
 </LastTradePrice>
 <DayVolume>
 10000
 </DayVolume>
 </PriceAndVolume>
 </m:GetLastTradePriceResponse>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Пример 9. Сообщение об ошибке (отсутствие обязательного заголовка).

```
HTTP/1.1 500 Internal Server Error
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Body>
 <SOAP-ENV:Fault>
 <faultcode>SOAP-ENV:MustUnderstand</faultcode>
 <faultstring>SOAP Must Understand Error</faultstring>
 </SOAP-ENV:Fault>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Пример 10. Сообщение об ошибке обработки тела сообщения.

```
HTTP/1.1 500 Internal Server E--ОГ
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Body>
 <SOAP-ENV:Fault>
 <faultcode>SOAP-ENV:Server</faultcode>
 <faultstring>Server Error</faultstring>
 <detail>
 <e:myfaultdetails xmlns:e="Some-URI">
 <message>
 My application didn't work
 </message>
 <errorcode>
 1001
 </errorcode>
 </e:myfaultdetails>
 </detail>
 </SOAP-ENV:Fault>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Предметный указатель

& (амперсант) 64
' (апостроф) 65
" (двойная кавычка) 65
#! (псевдокомментарий) 337, 353
/ (слеш) 64
80112.xml (файл) 124-126

A

ACORD (спецификация) 34, 60
ActiveXObject (функция) 91
actor (глобальный атрибут SOAP) 430
addAttribute (функция) 108
ADO Recordset (объект) 10
Adobe PostScript 58
Air Transport Association (ATA) 59
American National Standards Institute (ANSI) 49
ANPA 1312 (спецификация) 23
Apache для Win32 337
appendChild (метод) 108, 109
async (свойство) 91

B

BFC-сервер 361
BizTalk Editor
- запуск 262
- пример создания схемы 262
— сохранение документов 265
- сравнение с XML Authority 262
BizTalk Framework 2.0 (спецификация) 356
BizTalk Mapper
- выбор схем 267
— дополнительные возможности 273
- запуск 267
- создание отождествлений 267
— — двух схем заказа 268
- сохранение отождествлений 271
BizTalk Workflow Designer 273
BizTalk-серверы *см. также* Microsoft BizTalk Server 2000
- взаимодействие с партнерами 213, 273
- возможность расширения 214
- маршрутизация и документооборот 210
- масштабируемость 213
- отслеживание документов 212
- поддержка EDI 215
- поддержка очереди сообщений 212
- поддержка пакетной обработки 212
- преобразование документов 214
— синхронные коммуникации 211
— системные требования 209
— создание 217
- требования 190, 208
- установка примеров 225
Body (элемент SOAP)
- определение в спецификации 366, 427, 428, 431
- роль в структуре сообщений 161, 162
— связь с элементом *Header* 432
BTServer.xom (файл) 221

C

CarPoint (Web-сайт) 28
CGI-программы
- заголовки HTTP 346
- компиляция 353
- конфигурирование Web-сервера 338
- переменные среды 344
- перехват ошибок 348
- примеры 171, 348
- псевдокомментарий #! 337, 353
- разработка на языке Omni-Mark 343
childNodes (свойство) 94, 95
Confirm.xml (файл) 202, 203

createAttribute (функция) 107
create Element (метод) !08

D

<delivery> (бизнес-тэг) 196, 368, 381, 385
<description> (бизнес-тэг) 374
Distributed COM (DCOM) 152
do sgml-parse (правило OmniMark) 285
do xml-parse (правило OmniMark) 285, 286
DOM *см.* объектная модель документа (DOM)
Duck.xml (файл) 288
Duckone.xml (файл) 127

E

EDI *см.* электронный документооборот
encodingStyle (глобальный атрибут SOAP) 428
Enterprise Java Beans (EJB) 152
<Envelope> (элемент SOAP) 196, 427, 458
<expiresAt> (бизнес-тэг) 372. 385

F

Fault (элемент SOAP) 432
Financial Information Exchange (FIX) 34
find (правило OmniMark) 282. 283, 304
FORM (элемент) 158, 159
<from> (бизнес-тэг) 369, 371
FTP-сервер, порт по умолчанию 160

G

generic.xsl (шаблон) 123-126
get Named Item (метод) 94
GetXMLNews.xom 179

H

<handle> (бизнес-тэг) 376, 377
Header (элемент SOAP)
- определение в спецификации BizTalk Framework 366, 368
- роль в структуре сообщений 161
— связь с элементом <Body> 432
HL7 (спецификация) 34
HTML *см. также* документы HTM1.
- история 147
— ограничения 40, 147
- преимущества 39
- связь с SGML 48
- сравнение с XML 11, 48
- стандарт W3C 52
HTTP *см. также* Web-серверы
- GET (метод) 156
- HTTPS и HTTP 160
— - порт по умолчанию 160
- вопросы безопасности 209
- заголовки в CGI-программах 346
- запрос 156
- и документы BizTalk 395
- как транспорт для доставки документов BizTalk 209
- ответ 157
- пакеты 159
- привязки 395
- расширение 454
- серверы 8

I

<identity> (бизнес-тэг) 372
IETF (Internet Engineering Task Force) 57
integer (тип данных OmniMark) 292
<instance> (бизнес-тэг) 376
InteliiSense (система подсказок в Visual Basic) 102, 104
Internet Information Server (IIS), настройка для запуска CGI-программ на OmniMark 338
Hem (метод объекта *XMLDOMNodeList*) *95*

J

J2008 (спецификация) 34, 60
Java 36, 152

JavaScript 91
joke.xsl (шаблон) 127-131
JScript, применение к клиентских приложениях SOAP 162, 166

L

length (свойство объекта *XMLDOM-NodeList*) 95
load (метод) 92
loadXML (метод) 92, 107

M

<manifest> (бизнес-тэг) 196. 368, 373, 387, 392
Microsoft BizTalk Server 2000 261
- BizTalk Editor 262
- BizTalk Mapper 266
- организация взаимодействия с партнерами 273
Microsoft Data Access Components (MDAC) 10
Microsoft Visual Basic
- IntelliSense 102
- запуск 98
- использование COM-сервера MSXML 101
- поддержка объектной модели документа 97
- форма BaseballStats 100
Microsoft Windows DNA 7
Microsoft XML DOM (объект) 86
MIME 387, 390
MSN CarPoint (Web-сайт) 28
mustUnderstand (глобальный атрибут SOAP) 161,431

N

newChild (объект) 109
NewPO.xml (файл) 197
News Industry Text Format (NITF) (спецификация) 23
Newspaper Association of America (NAA) 18,23
nextNode (метол объекта *XMLDOM NodeList*) 95
NITF (спецификация) 23
NODE_ATTRIBUTE (тип узла) 89
NODE_CDATA_SECTION (тип узла) 89
NODE_COMMENT (тип узла) 90
NODE_DOCUMENT (тип узла) 90
NODE_DOCUMENT_FRAGMENT (тип узла) 90
NODE_DOCUMENT_TYPE (тип узла) 90
NODE_ELEMENT (тип узла) 89
NODE_ENTITY (тип узла) 89
NODE_ENTITY_REFERENCE (тип узла) 89
NODE_NOTATION (тип узла) 90
NODE_PROCESSING_INSTRUCTION (тип узла) 89
NODE_TEXT (тип узла) 89
nodeData Type (свойство) 90
NodeLisr (объект) 95
nodeName (свойство) 93
node Type (свойство) 91

O

OASIS 33, 50, 419
objDocumen (объект) 93
OmniMark
- *output* (оператор) 294
- *put* (оператор) 294
— *set* (оператор) 294
- *TCPConnectionGetOutput* (функция) 291
- *TCPConnectionGetSource* (функция) 291
- *TCPConnectionOpen* (функция) 291
- ввод-вывод и переменные 294
- виды правил 282
- генерация и перехват исключений 312
- группы правил 323
- компоненты OMX 289
- константы 324
- макросы 324
— модель ввода-вывода 282
— область ввода 282, 286

- область вывода 282, 287
- обработка исключений 312
- операторы 304
— цикла 303
— ошибки и восстановление работы сервера 334
- переменные 292
- перехват ошибок 312
- поддержка XML 285
- поддержка прикладных протоколов 328
- поддержка транспортных протоколов 328
- поиск по шаблону 304
- **реализация сервера SOAP** 170
- сетевое программирование 328
- системные библиотеки 342
— сравнение с Perl 280
— структуры данных 296
- типы данных 292
- условные **выражения 301**
- файл аргументов 352, 353
- файл проекта 353
— финансовые вычисления 326
- функции 310
- шаблоны 309
OMX (компоненты OmniMark) 289
Oracle !0
output (оператор OmniMark) 294

P

parseError (объект) 93
Perl 280
Personal Web Server, настройка для запуска CGI-программ на OmniMark 339
PlaceOrder.htm (файл) 221, 244
PlaceOrder.xom (файл) 221, 245
PO1-2.xsl (шаблон) 271
poConfirm.asp (файл) 256
poGen.exe (файл)
- подпрограмма *createPO_Click* 239
- подпрограмма *Form.Load* 238, 239
— функция *sendMsg* 242
POST-запрос 158, 343, 344
PostScript 58
prEnter.asp (файл) 226. 227, 229
<process> (бизнес-тэг) 196, 368, 376
<properties> (бизнес-гэг) 196, 368, 371, 379
prProcess.asp (файл) 230
PurchaseOrder1.xml (файл) 263, 266
put (оператор OmniMark) 294

R

readyState (свойство) 91, 92
RealServer (порт по умолчанию) 160
Receipt.xml (файл) 201, 202
<receipt> (бизнес-тэг) 379
<receiptRequiredBy> (бизнес-тэг) 371, 381, 385
<receivedAt> (бизнес-тэг) 381
<reference> (бизнес-тэг) 374, 392
<reliability> (бизнес-тэг) 369, 371, 381, 385
reset (метод объекта *XMLDOMNodeList*) 95
root (атрибут SOAP) 453
ROPE 234
RTF (формат файлов) 116, 117

S

SAX 114
selectNodes (метод) 86, 96
selectSingleNode (метод) 86, 96
<sendReceiptTo> (бизнес-тэг) 371, 381, 385
<sentAt> (бизнес-тэг) 372
Services.xml (файл) 235, 236
set (оператор Omni Mark) 294
SGML 42
- **DSSSL** 115
— достоинства 42
- недостатки 44
- связь с XML 46
SOAP 18, 153, 155, 421
- *actor* (глобальный атрибут) 430
- *encodingStyle* (глобальный атрибут) 428
- *mustUnderstand* (глобальный атрибут) 161, 431

- атрибут *root* 453
- атрибуты заголовка 430
— документ как концерт 161, 196
- использование HTTP 453
- кодировка данных 434
- массивы 442, 446
- модель обмена сообщениями 425
- нота W3C 421
- правила представления данных XML 435
- применение для удаленного вызова процедур 455
- пример пересылки по HTTP 454
- примеры сообщений 424
- простые типы 439
- роль XML 153
- роль в инфраструктуре BizTalk 359
- связь с XML 426
- связь элементов *Header* и *Body* 432
— создание приложений, использующих Web-службы 162
- создание сервера на OmniMark 170
— составные типы 452
— структура сообщений 161
- цели разработки 424
- элементы
— - *Body* 161, 162, 427, 428, 431
— - *Envelope* 196, 427—429, 458
— - *Fault* 432
— - *Header* 161, 427—429
— элементы сообщения 427
SOAPAction (заголовок запроса HTTP) 454
SOAPServer.xom (программа) 171
SQL Server 10
stream (тип данных OmniMark) 292
string (тип данных SOAP) 440
SQL и XML 10

T

TCP/IP 147
TCPConnectionGetOutput (функция OmniMark) 291
TCPConnectionGetSource (функция OmniMark) 291
TCPConnectionOpen (функция OmniMark) 291
text (свойство) 93
<to> (бизнес-тэг) 369
<topic> (бизнес-тэг) 372
<Transaction> (элемент SOAP) 161
<type> (бизнес-тэг) 376

U

UNIX 152

V

Visual Basic *см.* Microsoft Visual Basic

W

weathersheet.xsl (шаблон) 121—123
Web-обозреватель как гонкий клиент 5
Web-сайт
- BUTalk.org 416
- XML.com 417
- XML.org 419
- XMLSOFTWARE 420
- агрегирование информации 28, 147
Web-сервер Apache, настройка для выполнения CGI-программ OmniMark 338, 341
Web-серверы 8
- порт по умолчанию 160
Web-службы 150
Windows DNA *см.* Microsoft Windows DNA
W3C 33, 41, 50

X

XDR-схемы
- конверт SOAP 1.1
— - для документа BizTalk 407
— - для уведомления BizTalk 41(1
- элемент заголовка
— -- *delivery* 401
— - *manifest* 403

— — *process* 405
— - *properties* 403
— - *receipt* 407
XLink 53
XML
— Java 36
- OmniMark 285
- SQL 10
- агрегирование информации 28
— анализ документов 86, 87, 285
- атрибуты элементов 65
- взаимосвязь с EDI 33
- вложенность элементов 63
— добавление
— — атрибутов 106
— - элементов 108
— документы как сообщения SOAP 427
- имена элементов 63
- инфраструктура BizTalk 191
- комментарии в документах 68
- корневой элемент документа 63
- объявление в документе 62
- открывающий и закрывающий тэги 62
— открытый статус 36
- подстановочные строки 64
- правила кодирования данных в SOAP 435
- правильные документы 61
- преимущества 45
- пример
— — действительного документа 69
— — документов ПО
— пространства имен 55. 73
- простые типы данных SOAP 439
- пустые элементы 67
- роль в SOAP 153
- роль в электронной коммерции 28, 34, 35
- связь с HTML 48
- связьс SGML 46
- связь с SOAP 426
- связь с объектной моделью документа 85
- синтаксисы схем 80
— создание документов 105
- специальные символы в документе 64
- сравнение с HTML 11,48
- стандарты 49, 53
- схемы 55, 79
- трехуровневая архитектура 6
- цели проектирования 45
— чувствительность к регистру символов 63
- языки обработки 36
XML Authority 80, 81, 262
XML Path Language (XPath) 118
XML-анализатор 86. 87, 285
XML.com (Web-сайт) 417
XML.org (Web-сайт) 419
XMLDOMElement (объект) 93
XMLDOMNodeList (объект) 95
XMLSOFTWARE (Web-сайт) 420
XPointer 53
XRay (XML-редактор) 97, 138, 141
XSD-схемы 83, 410
XSL 115
- назначение 116
- рекурсия 123
- средства преобразования см. XSLT
- стандарт W3C 56
- шаблон как правильный документ XML 123
- эволюция 115
XSLT 11, 118, 121, 123, 127, 138. 143. 214
- использование редактора XRay 138, 141
- преобразование документа 138
- стандарт W3C 56

А

автоматизация снабжения *см.* заказ
агрегирование информации
- важность 150
- ограничения HTML 147, 150
- роль XML 28
административная консоль BizTalk Server 273

анализ документов XML 86. 87, 285
анализатор MSXML 86, 101, 118
арифметические операторы OmniMark 304
архитектура COM 151
асинхронные коммуникации 211
атрибуты SOAP
- *actor* 430
- *encodingStyle* 428
- *mustUnderstand* 161, 431
- *root* 453
атрибуты заголовка SOAP 430
атрибуты элементов XML
— добавление в документ 106
- набор объекта *XMLDOMElement* 93

Б

базы данных 7, 222
байтовый массив как тип данных SOAP 442
бизнес-документы 362
бизнес-тэги
- структура документа BizTalk 366
- <delivery> 196, 368, 381, 385
- <manifest> 196, 368. 373, 387, 392
- <process> 196, 368, 376
— <properties> 196, 368, 371, 379
- элементы заголовка документов BizTalk 368
брандмауэры и IP-порты 54, 154

В

версии XMLDOM 86
взаимодействие с партнерами 18
вложения
- в спецификации BizTalk Framework 387
- пример сообщения BizTalk 398
внешние источники данных *см. также* поставщики информации
- New York Times 18
- Weather Channel 14
— в новостном бизнесе 18
- вопросы совместимости форматов 18
- метеоданные 14
- роль XML 28

Г

генерация и перехват исключений в OmniMark 312

Д

двоичные файлы, пересылка 347
двойная кавычка (") 65
документы BizTalk
- гарантированная доставка 377
- и сообщения BizTalk 364
- передача по протоколу HTTP 395
- пересылка в элементе SOAP *Body* 366
- преобразование *см.* XSLT
- примеры 365
- с вложениями 365, 387
— структура 365
— элементы заголовка SOAP 366. 368
документы HTML *см. также* HTML
- PlaceOrder.htm (файл) 221, 244
- в клиентских приложениях SOAP 165
- применение XSL 116
— элементы 158
домашняя страница Microsoft BizTalk 417

Ж

жетоны 8
журнал <TAG> 420

З

заказ
- использование BizTalk Mapper для отождествления схем 267
- обработка в компании Toy Car Parts 244
- обработка подтверждений 256
- подготовка 192
- подтверждение 202, 256

- создание схемы с помощью BizTalk Editor 262
- установка BizTalk для выполнения примера 225
закрывающий тэг 63
запрос GET 156, 343. 344
зашита и протокол HTTP 209

И

иерархические таблицы стилей 52
инициатива ebXML 33
инфраструктура BizTalk
- компоненты 190
— логическая структура 364
- модель версий 360
- спецификации 360, 361
- применение в транзакциях B2B 192
- пример сценария 192
— примеры документов BizTalk и сообщений SOAP 1.1 196
- роль SOAP 359
- роль XML 191
- сравнение версий 359
— терминология 361
- типы данных XML-схем 361

К

клиентский уровень трехуровневых приложений 8
комментарии в документах XML 68
консорциум W3C 33, 41. 50
константы OmniMark 324
корневой элемент
- SOAP 427
- XML 63

М

макросы OmniMark 324
массивы SOAP 442
Международная организация стандартов 49
метод 151

О

обработка исключений OmniMark 312
обратный аукцион 151
объектная модель документа
- и документы XML 85
- использование в Visual Basic 97
- реализация Microsoft 86
- создание объектов 91
- сравнение с событийными моделями 112
— стандарт W3C 57, 85
объявление XML 62
объявление типа документа (DOCTYPE) 71
операторы OmniMark 304
— сравнения 304
- цикла 303
определение типа документа (DTD) 55, 70
открывающий тэг 63
отслеживание документов BizTalk Document 273

П

переменные OmniMark 292
- переменные шаблона 309
- полки 296
- ссылки 299
переменные среды
- CGI 344
- OmniMark 342
подстановки 64
поиск по шаблону в OmniMark 304
полиморфный доступ 442
полки OmniMark 296
порты IP 160
поставщики информации *см. также* внешние источники данных
- New York Times 18
- Weather Channel 14
- в новостном бизнесе 18
- вопросы совместимости форматов 18

- метеоданные 14
- роль XML 28
потоковые языки программирования 280
- OmniMark 280
- достоинства 280
правила OmniMark
— обработки данных 282
- обработки ошибок разметки 282
- обработки разметки 282. 286
- обработки элементов 282, 288
- объединение в группы 323
- поиска 282. 304
преобразование *см.* XSLT
прикладные протоколы, поллержка в OmniMark 328
приложения BizTalk 362
пространства имен XML 55 см. также схемы
пространство имен форматирующего объекта 116
профиль подписчика 8
псевдокомментарий #! 337, 353
пустые элементы XML 67

Р

разметка
— документов XML 62
- обработка в OmniMark 285
разреженные массивы 451
результирующее дерево 116
рекурсивное выполнение шаблонов XSL 123

С

свойства 151
сильно связанные системы 152
синтаксис схем XDR 90
событийные модели, область применения 112
сообщения *см.* сообщения BizTalk; сообщения SOAP
сообщения BizTalk
- вложения в контейнере MIME 390
- область действия спецификации 359
- пример 397
- сравнение с документами BizTalk 364
сообщения SOAP 161. 427
- примеры из спецификации BizTalk Framework 366, 397
- структура 161
- элемент *Body* 161, 162, 366. 427, 428, 431
- элемент *Header* 161, 427–429
- элементы заголовка 366, 368
список как тин данных SOAP 441
средний уровень трехуровневой архитектуры 8
средства
- сетевого программирования OmniMark 328
— управления BizTalk 273
ссылочные переменные OmniMark 299
стандарты XML
- IETF 57
- ISO 49
- OASIS 33, 50, 419
- консорциум W3C 33, 41, 50
стандарты де-факто 58
структуры данных OmniMark 296
схемы
- XML Data Reduced (XDR) 401
- XSD 83, 410
- обзор 55, 362
— разработка 58
- рекомендации W3C 79

Т

типы данных OmniMark 292
«тощие клиенты» 8
транспорт в инфраструктуре BizTalk 365
транспортные протоколы, поддержка в OmniMark 328
трехуровневая архитектура
— внешние источники данных 13

- клиентский уровень 8
- применение XML 6
- средний уровень 8
- уровень данных 7

У

уведомления BizTalk 201, 379
удаленный вызов процедур, синтаксис SOAP 455
узлы документа XML 87
- потомки 94
уровень данных трехуровневой архитектуры
- база данных профилей подписчиков 8
— базы и источники данных 7
условные операторы OmniMark 301
устройства ввода в OmniMark 286
устройство вывода, создание в OmniMark 287

Ф

финансовые вычисления в OmniMark 326
функции OmniMark 310
функция *headerNode* (создание элементов XML) 108

Ч

частичная передача массивов 451
чувствительность к регистру в XML 63

Э

эволюция WWW 146
электронная коммерция
- «бизнес — бизнес» и «бизнес - потребитель» 4. 32
- инфраструктура BizTalk 189
- обзор 2
- роль XML 28, 34, 35
электронная коммерция в секторе «бизнес — бизнес»
- агрегирование информации 32
- инфраструктура BizTalk 189
- сравнение с сектором «бизнес — клиент» 4, 32
электронная коммерция в секторе «бизнес — клиент» 2, 28, 32
электронный документооборот (EDI)
- поддержка серверами BizTalk 215
- роль XML 33
элементные правила OmniMark 282. 2S8
элементы XML *см. также* элементы документа XML
- вложенные 63
— доступ к свойствам 93
- **корневые 63**
- пустые 67
- свойства
— - *documentElement* 93
— — *nodeName* 93
— - *text* 93
элементы заголовка SOAP 366, 368

Я

язык запросов XML (XQL) 57
язык описания семантики документа (DSSSL) 115
язык описания служб (SDL) 234

Брайан Трэвис

Брайан Трэвис — основатель и технический директор компании Architag (Колорадо), занимающейся обучением и консалтингом (*http://www.architag.com*).

Брайан — один из ведущих специалистов по XML. Основав в 1993 г. компанию Information Architects (впоследствии Architag), он создал множество «умных» систем документооборота и решений для электронной коммерции по заказам клиентов со всего света,

Кроме того, Брайан — известный инструктор и лектор по XML и связанным с ним темам. Будучи ведущим инструктором курсов компании Architag, он читает лекции по XML в Америке, Европе и Азии. Брайан проводит беседы и семинары об XML на крупнейших выставках и мероприятиях, посвященных Интернету и электронной коммерции, включая Internet World. Web Design and Development, Microsoft TechEd и COMDEX.

Брайан — автор нескольких книг о системах документооборота. Кроме того, он выпускает бюллетень *<TAG>*. посвященный структурированным информационным системам и пользующийся популярностью во всем мире. Книга, над которой он работает сейчас, называется «1001 анекдот про гусей» и будет опубликована, когда число анекдотов в его коллекции достигнет упомянутого в заголовке магического числа 1001. В данный момент в ней 26 перлов.

В настоящее время Брайан постоянно проживает в кресле «2С» самолетов авиакомпании United Airlines, но платит налоги в Колорадо, где является единственным обладателем доступа в Интернет по магистральной линии класса Т1.

Хобби Брайана — лежать в джаккузи и предаваться мечтаниям.

ЛИЦЕНЗИОННОЕ СОГЛАШЕНИЕ MICROSOFT

прилагаемый к книге компакт-диск

ЭТО ВАЖНО — ПРОЧИТАЙТЕ ВНИМАТЕЛЬНО. Настоящее лицензионное соглашение (далее «Соглашение») является юридическим документом, оно заключается между Вами (физическим или юридическим лицом) и Microsoft Corporation (далее «корпорация Microsoft») на указанный выше продукт Microsoft, который включает программное обеспечение и может включать сопутствующие мультимедийные и печатные материалы, а также электронную документацию (далее «Программный Продукт»). Любой компонент, входящий в Программный Продукт, который сопровождается отдельным Соглашением, подпадает под действие именно того Соглашения, а не условий, изложенных ниже. Установка, копирование или иное использование данного Программного Продукта означает принятие Вами данного Соглашения. Если Вы не принимаете его условия, то не имеете права устанавливать, копировать или как-то иначе использовать этот Программный Продукт.

ЛИЦЕНЗИЯ НА ПРОГРАММНЫЙ ПРОДУКТ

Программный Продукт защищен законами Соединенных Штатов по авторскому праву и международными договорами по авторскому праву, а также другими законами и договорами по правам на интеллектуальную собственность.

1. **ОБЪЕМ ЛИЦЕНЗИИ.** Настоящее Соглашение дает Вам право:
   а) **Программный продукт.** Вы можете установить и использовать одну копию Программного Продукта на одном компьютере. Основной пользователь компьютера, на котором установлен данный Программный Продукт, может сделать только для себя вторую копию *а* использовать ее на портативном компьютере.
   б) **Хранение или использование в сети.** Вы можете также скопировать или установить экземпляр Программного Продукта на устройстве хранения, например на сетевом сервере, исключительно для установки или запуска данного Программного Продукта на других компьютерах в своей внутренней сети, но тогда Вы должны приобрести лицензии на каждый такой компьютер. Лицензию на данный Программный продукт нельзя использовать совместно или одновременно на других компьютерах.
   в) License Pak. Если Вы купили эту лицензию в составе Microsoft License Pak, можете сделать ряд дополнительных копии программного обеспечения, входящего в данный Программный Продукт, и использовать каждую копию так, как было описано выше. Кроме того, Вы получаете право сделать соответствующее число вторичных копий для портативного компьютера в целях, также оговоренных выше.
   г) **Примеры кода.** Это относится исключительно к отдельным частям Программного Продукта, заявленным как примеры кода (далее «Примеры»), если таковые входят в состав Программного Продукта.
      i) **Использование и модификация.** Microsoft дает Вам право использовать и модифицировать исходный код Примеров при условии соблюдении пункта (d)(iii) ниже. Вы не имеете права распространять в виде исходного кода ни Примеры, ни их модифицированную версию.
      ii) **Распространяемые файлы.** При соблюдении пункта (d)(iii) Microsoft дает Вам право на свободное от отчислений копирование и распространение в виде объектного кода Примеров или их модифицированной версии, кроме тех частей (или их модифицированных версий), которые оговорены в файле Readme, относящемся к данному Программному Продукту, *как* не подлежащие распространению.
      iii) **Требования к распространению файлов.** Вы можете распространять файлы, разрешенные к распространению, при условии, что: а) распространяете их в виде объектного кода только в сочетании со своим приложением и как его часть; б) не используете название, эмблему или товарные знаки Microsoft для продвижения своего приложения; в) включаете имеющуюся в Программном Продукте ссылку на авторские права в состав этикетки и заставки своего приложения; г) согласны освободить от ответственности и взять на себя защиту корпорации Microsoft от любых претензии или преследований по закону, включая судебные издержки, если таковые возникнут в результате использования или распространения Вашего приложения; и д) не допускаете дальнейшего распространения конечным пользователем своего приложения. По поводу отчислении и других условий лицензии применительно к иным видам использования или распространения распространяемых файлов обращайтесь в Microsoft.

Z. ПРОЧИЕ ПРАВА И ОГРАНИЧЕНИЯ

- Ограничения на **реконструкцию, декомпиляцию** и **дизассемблирование**. Вы не имеете права реконструировать, декомпилировать или дизассемблировать данный Программный Продукт, кроме того случая, когда такая деятельность (только в той мере, которая необходима) явно разрешается соответствующим законом, несмотря на это ограничение.
- Разделение **компонентов**. Данный Программный Продукт лицензируется как единый продукт. Его компоненты нельзя отделять друг от друга для использования более чем на одном компьютере.
- Аренда. Данный Программный Продукт нельзя сдавать в прокат, передавать во временное пользование или уступать для использования ь иных целях.
- Услуги по **технической поддержке**. Microsoft может (но не обязана) предоставить Вам услуги по технической поддержке данного Программного Продукта (далее «Услуги»). Предоставление Услуг регулируется соответствующими правилами и программами Microsoft, описанными в руководстве пользователя, электронной документации и/или других материалах, публикуемых Microsoft. Любой дополнительный программный код, предоставленный в рамках Услуг, следует считать частью данного Программного Продукта и подпадающим под действие настоящего Соглашения. Что касается технической информации, предоставляемой Вами корпорации Microsoft при использовании ее Услуг, то Microsoft может задействовать эту информацию в деловых целях, в том числе для технической поддержки продукта и разработки. Используя такую техническую информацию, Microsoft не будет ссылаться на Вас.
- **Передача** прав на **программное обеспечение**. Вы можете безвозвратно уступить все права, регулируемые настоящим Соглашением, при условии, что не оставите себе никаких копии, передадите асе составные части данного Программного Продукта (включая компоненты, мультимедийные и печатные материалы, любые обновления, Соглашение и сертификат подлинности, если таковои имеется) ч принимающая сторона согласится с условиями настоящего Соглашения.
- **Прекращение** действия Соглашения. Без ущерба для любых других прав Microsoft может прекратить действие настоящего Соглашения, если Вы нарушите его условия. В лом случае Вы должны будете уничтожить все копии данного Программною Продукта вместе со всеми его компонентами,

3. АВТОРСКОЕ ПРАВО. Все авторские права и право собственности на Программный Продукт (в том числе любые изображения, фотографии, анимации, видео, аудио, музыку, текст, примеры кода, распространяемые файлы и апплеты, включенные в состав Программного Продукта) и любые его копии принадлежат корпорации Microsoft или ее поставщикам. Программный Продукт охраняется законодательством об авторских правах и положениями международных договоров. Таким образом, Вы должны обращаться с данным Программным Продуктом, как с любым другим материалом, охраняемым авторскими правами, с тем исключением, что Вы можете установить Программный Продукт на один компьютер при условии, что храните оригинал исключительно как резервную или архивную копию. Копирование печатных материалов, поставляемых вместе с Программным Продуктом, запрещается.

ОГРАНИЧЕНИЕ ГАРАНТИИ

ДАННЫЙ ПРОГРАММНЫЙ ПРОДУКТ (ВКЛЮЧАЯ ИНСТРУКЦИИ ПО ЕГО ИСПОЛЬЗОВАНИЮ) ПРЕДОСТАВЛЯЕТСЯ БЕЗ КАКОЙ-ЛИБО ГАРАНТИИ. КОРПОРАЦИЯ MICROSOFT СНИМАЕТ С СЕБЯ ЛЮБУЮ ВОЗМОЖНУЮ ОТВЕТСТВЕННОСТЬ. В ТОМ ЧИСЛЕ ОТВЕТСТВЕННОСТЬ ЗА КОММЕРЧЕСКУЮ ЦЕННОСТЬ ИЛИ СООТВЕТСТВИЕ ОПРЕДЕЛЕННЫМ ЦЕЛЯМ. ВЕСЬ РИСК ПО ИСПОЛЬЗОВАНИЮ ИЛИ РАБОТЕ С ПРОГРАММНЫМ ПРОДУКТОМ ЛОЖИТСЯ НА ВАС.
НИ ПРИ КАКИХ ОБСТОЯТЕЛЬСТВАХ КОРПОРАЦИЯ MICROSOFT, F.E РАЗРАБОТЧИКИ, А ТАКЖЕ ВСЕ, ЗАНЯТЫЕ В СОЗДАНИИ, ПРОИЗВОДСТВЕ И РАСПРОСТРАНЕНИИ ДАННОГО ПРОГРАММНОГО ПРОДУКТА, НЕ НЕСУТ ОТВЕТСТВЕННОСТИ ЗА КАКОЙ-ЛИБО УЩЕРБ (ВКЛЮЧАЯ ВСЕ, БЕЗ ИСКЛЮЧЕНИЯ, СЛУЧАИ УПУЩЕННОЙ ВЫГОДЫ, НАРУШЕНИЯ ХОЗЯЙСТВЕННОЙ ДЕЯТЕЛЬНОСТИ, ПОТЕРИ ИНФОРМАЦИИ ИЛИ ДРУГИХ УБЫТКОВ) ВСЛЕДСТВИЕ ИСПОЛЬЗОВАНИЯ ИЛИ НЕВОЗМОЖНОСТИ ИСПОЛЬЗОВАНИЯ ДАННОГО ПРОГРАММНОГО ПРОДУКТА ИЛИ ДОКУМЕНТАЦИИ, ДАЖЕ ЕСЛИ КОРПОРАЦИЯ MICROSOFT БЫЛА ИЗВЕЩЕНА О ВОЗМОЖНОСТИ ТАКИХ ПОТЕРЬ. ТАК КАК В НЕКОТОРЫХ СТРАНАХ НЕ РАЗРЕШЕНО ИСКЛЮЧЕНИЕ ИЛИ ОГРАНИЧЕНИЕ ОТВЕТСТВЕННОСТИ ЗА НЕПРЕДНАМЕРЕННЫЙ УЩЕРБ, УКАЗАННОЕ ОГРАНИЧЕНИЕ МОЖЕТ ВАС НЕ КОСНУТЬСЯ.

РАЗНОЕ

Настоящее Соглашение регулируется законодательством штата Вашингтон (США), кроме случаев (и лишь в той мере, насколько это необходимо) исключительной юрисдикции тою государства, на территории которого используется Программный Продукт.
Если у Вас возникли какие-либо вопросы, касающиеся настоящего Соглашения, или если Вы желаете связаться с Microsoft по любой другой причине, пожалуйста, обращайтесь в местное представительство Microsoft или пишите по адресу: Microsoft Sales Information Center, One Microsoft Way, Redmond, WA 98052-6399.

Трэвис Брайан

XML и SOAP: программирование для серверов BizTalk.

Новейшие технологии

Перевод с английского пол общей редакцией А. **В. Кузьмина**

Технический редактор **Н. Г. Тимченко**

Компьютерныйдизайн
и подготовка иллюстраций Д. **В. Петухов**

Дизайнер обложки **Е. В. Козлова**

Оригинал-макет выполнен с использованием
издательской системы Adobe PageMaker 6.0

Pora(-)Type

TypeMarketFon tLibrary
легальный пользователь

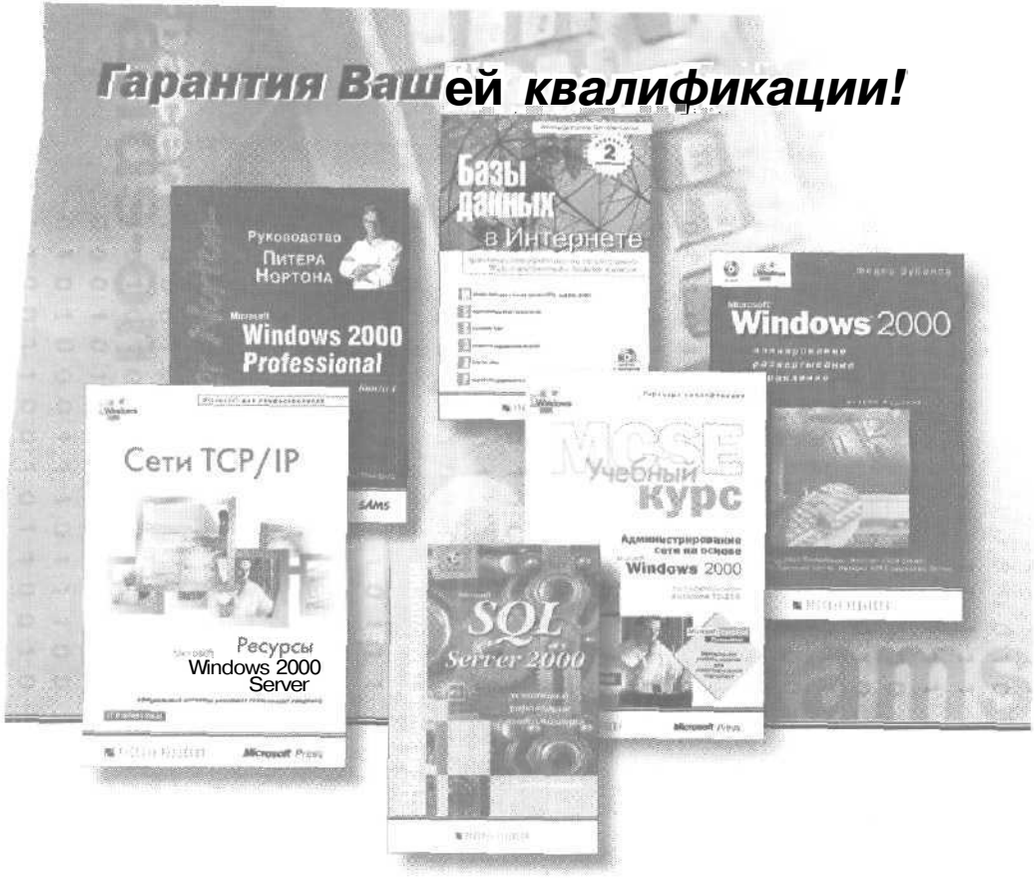
Главный редактор **А. И. Козлов**

Подготовлено к печати издательско-торговым домом
«Русская Редакция»

РУССКАЯ РЕДАКЦИЯ

Лицензия ЛР № 066 422 от 19.03.99 г.
Подписано в печать 16.02.2001 г. Физ. п. л. 31.
Формат 60x901/16. Тираж 5000 экз.

Отпечатано в ОАО «Типография "Новости"»
107005, Москва, ул. Фр. Энгельса, 46



Издательство «Русская Редакция»

— партнер Microsoft Press в **России** —

предлагает широкий выбор литературы

по современным информационным технологиям.

Мы переводим на русский язык бестселлеры ведущих издательств мира,

а также сотрудничаем с компетентными российскими авторами.

Наши книги Вы можете приобрести

- в Москве:
  - «Библио-Глобус» ул. Мясницкая, 6, тел.: (095) 928-3567
  - «Московский дом книги» ул. Новый Арбат, 8, тел.: (095) 290-4507
  - «Дом технической книги» Ленинский пр-т, 40, тел.: (095) 137-6019
  - «Молодая гвардия» ул. Большая Полянка, 28, тел.: (095) 238-5001
  - «Мир» Ленинградский пр-т, 7В тел.: (095) 152-4511
  - «Мир печати» ул. 2-я Тверская-Ямская, 54, ТЕЛ. (095) 978-5047
  - Торговый дом «Москва» ул. Тверская 8, тел. (095) 229-6483
  - «Алекс и К°» Магазин «Книги» г. Зеленоград, Панфиловский пр-т, 1106Б, тел.: (095) 532-5669
- в Санкт-Петербурге: ЗАО «Диалект», тел. (812) 247-1483
- в Новосибирске: ООО «Топ-книга», тел.: (3832) 36-1026
- в Набережных Челнах: ООО «Аспект-С», тел. (8552) 58-8013
- в Алма-Ате (Казахстан): ЧП Болат Амреев, тел.: 8-327-290-191-25, (3272) 26-1404
- в Киеве (Украина): ООО Издательство «Ирина» тел.: (044) 289-0423; «Техническая книга на Петровке», тел.: (044) 464-6895

Фирменный магазин «Компьютерная и деловая книга» Москва, Ленинский проспект, строение 38, тел.: (095) 773-7269

Интернет-магазин www.tfbook.ru

РУССКАЯ РЕДАКЦИЯ

тел.: (095) 142-0571; тел./факс: (095) 145-4519

e-mail: info@rusedit.ru; http://www.rusedit.ru

КАТАЛОГ
ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

SoftLine direct

ПРЕДНАЗНАЧЕН ДЛЯ:

- Руководителей отделов информатизации
- Системных администраторов
- Разработчиков программного обеспечения
- Разработчиков решений для Internet
- Системных интеграторов
- Профессионалов рынка информационных технологий

САМЫЙ УДОБНЫЙ
СПОСОБ ПОКУПКИ
ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

БЕСПЛАТНАЯ
подписки!

(095) 232-0023
www.softline.ru

БЛАНК ПОДПИСКИ

ФАМИЛИЯ		ИМЯ
ОРГАНИЗАЦИЯ		ДОЛЖНОСТЬ
СТРАНА	ИНДЕКС	ГОРОД
УЛИЦА/ДОМ		
ТЕЛЕФОН		ФАКС
E-MAIL		WWW

XML SOAP
программирование для серверов BizTalk

Новейшие технологии

XML, SOAP и BizTalk — язык Интернет-бизнеса

Эта книга посвящена технологиям, лежащим в основе электронной коммерции и современного Интернет-бизнеса. Брайан Трэвис познакомит вас с разработкой приложений для сферы «бизнес-бизнес» (B2B) на основе языка XML. Начав с рассмотрения причин популярности XML, автор детально описывает существующие промышленные стандарты, схемы представления данных и идеологию BizTalk, освещает вопросы интеграции приложений и показывает, как создавать приложения электронной коммерции и документооборота на платформе BizTalk. Материал представлен на конкретных примерах, включая пример разработки BizTalk-сервера для обработки XML-документов.

На прилагаемом компакт-диске:

- электронная версия книги;
- пакет разработчика SOAP для Visual Studio 6.0;
- XML-редактор компании ArchiTag;
- исполняющая система C/VM и интегрированная среда разработки для языка сценариев OmniMark;
- электронная версия 1-го тома книги Брайана Трэвиса «OmniMark At Work»;
- коды всех примеров и упражнений;
- пробная 120-дневная версия Microsoft BizTalk Server 2000;
- пробная 120-дневная версия Microsoft Visio 2000.

Отзывы о книге присылайте по адресу:
info@rusedit.ru

Издательство «Русская Редакция» в 2001 году выпускает следующие книги серии «Новейшие технологии»:

Книги этой серии познакомят Вас с последним словом в мире информационных технологий.

ISBN 5-7502-0129-5
9 785750 201297

Web-узел издательства: **www.rusedit.ru**
Интернет-магазин: **www.ITbook.ru**

Отзывы читателей, присланные авторам на адрес alexandre@frolov.pp.ru

«Я в восторге от Вашей книжки "Базы данных в Интернете", особенно от разделов, посвященных страницам ASP и созданию магазина на ASP».

Надежда (web-мастер), nad@mail.ur.ru

«...спасибо за хорошую книгу "Базы данных в Интернете"».

Чернов Николай Николаевич, Украина, Кировоград, ОАО «Кировоградоблэнерго», nick_chernov@MailAndNews.com

«Мне очень понравилась Ваша книга "Базы данных в Интернете". Спасибо Вам большое...Именно эта книга мне необходима для работы».

Студент Университета Дружбы народов Шил Меликов, noil_melikov@mail.ru

«Я хотел бы выразить Вам свое восхищение: Вы проделали громадную и очень нужную работу!»

А. С. Цветков, доц. мат-мех. факультета СПб Университета, alex_ tsvetkov@mail.ru

Присылайте нам свои отзывы!