helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

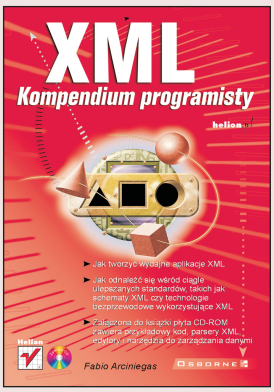
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

XML
Kompendium programisty



Autor: Fabio Arciniegas
Tłumaczenie: Tomasz Żmijewski
ISBN: 83-7197-573-2
Tytuł oryginału: XML Developer
Format: B5, stron: 589
Zawiera CD-ROM

Książka ta pomoże Ci przy tworzeniu wydajnych aplikacji XML oraz przygotowywaniu elastycznych struktur dokumentów. Przedstawiono tu wiele technologii XML — m.in. XML bezprzewodowe (WAP, VoiceXML i inne), a także wiele przykładów oraz język i powiązane z nim technologie. Książka jest przeznaczona dla średniozaawansowanych i zaawansowanych twórców publikacji sieciowych. Można z niej nauczyć się, jak wykorzystać XML do publikacji w Sieci i do wymiany danych; można poznać nowopowstające standardy, w tym schematy XML, XSLT, XPath, XLink i wiele innych.

- Tworzenie elastycznej struktury dokumentów XML.
- Dodawanie informacji źródłowych do dokumentów XML.
- Zrealizowanie relacji w bazie danych XML.
- Określenie sposobu odczytywania dokumentu XML.
- Wysoka jakość rozwiązań.
- Zasady użycia się XML.

Na dołączonej do książki płycie CD znajduje się przykładowy kod, edytory XML i parsery, które umożliwiają zaoszczędzenie wielu godzin pracy. Takiego połączenia specjalistycznej wiedzy z praktycznymi poradami dla projektantów i programistów nie znajdziesz w żadnej innej książce.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

O Autorze **15**

Wstęp **17**

Część I Struktura, składnia i użycie XML **21**

Rozdział 1. Podstawy XML **23**

Wprowadzenie 23
Mity na temat znaczników 23
Czym są znaczniki? 24
Definicja XML 29
Definicja ścisła 29
Definicja z punktu widzenia Sieci 29
Definicja z punktu widzenia danych 30
Album rodzinny XML 30
Składnia i najważniejsze cechy XML 30
Wprowadzenie 30
XML — szczegóły 33
Przykładowy dokument XML 46
Podsumowanie 46

Rozdział 2. DTD: Charakterystyka i techniki **47**

Wprowadzenie 47
Wprowadzenie do DTD 47
DTD a dokumenty 47
Deklarowanie elementów 48
Deklarowanie atrybutów 50
Deklarowanie encji 53
Więcej szczegółów na temat DTD 55
Deklaracje typu elementu 55
Deklaracje list atrybutów 59
Więcej o encjach 67
Rodzaje encji 67
Zewnętrzne encje ogólne 67

Wewnętrzne encje ogólne ... 68
Encje nieparsowane ... 68
Przydatny zestaw encji parametrycznych ... 69
Sekcje warunkowe ... 71
Wewnętrzny i zewnętrzny podzbiór DTD ... 71
Podsumowanie ... 72

Rozdział 3. Przestrzenie nazw. Wprowadzenie do przetwarzania XML ... 73

Wprowadzenie ... 73
Przestrzenie nazw ... 73
Definicja i przykład ... 73
Przykład ... 75
Deklaracja przestrzeni nazw ... 76
Nazwy kwalifikowane ... 77
Struktura — szczegóły ... 78
Typowe nieporozumienia ... 80
Modele przetwarzania ... 82
Paradygmaty ... 82
Możliwości parserów ... 83
Podsumowanie ... 84

Część II Parsowanie i programowe przetwarzanie XML ... 85

Rozdział 4. SAX i SAX2 (wersje 1.0 i 2.0) ... 87

Do czego służą SAX i SAX2? ... 87
Użycie SAX 1.0/SAX2 ... 88
Prosty program SAX ... 88
Definiowanie procedur obsługi zdarzeń ... 89
Zgłaszanie procedur obsługi zdarzeń parserowi, początek analizy XML ... 90
Struktura SAX2 ... 90
Interfejsy SAX ... 91
Zaawansowane zagadnienia związane z SAX2 ... 104
Filtry ... 104
Adaptery ... 105
Typowe błędy w programach korzystających z SAX ... 106
Niespójne procedury obsługi zdarzeń ... 106
Nadużywanie DefaultHandler ... 106
Bezpośrednie modyfikowanie stanu danych (mimo zapotrzebowania na historię) ... 107
Podsumowanie ... 108

Rozdział 5. Zaawansowane techniki programowania w SAX i SAX2 ... 109

Wprowadzenie ... 109
Kilka słów o szablonach ... 109
Wzorzec Builder w aplikacjach SAX2 ... 110
Użycie ... 110
Przykład ... 110
Budowa szablonu ... 113
Przykład ... 114
Konsekwencje użycia szablonu Builder ... 119

Szablon Command w aplikacjach SAX2 ... 119
Użycie ... 119
Przykład ... 119
Budowa szablonu ... 121
Przykład ... 122
Konsekwencje użycia szablonu Command ... 125
Sztafeta wywołań ... 125
Zastosowanie ... 125
Przykład ... 126
Budowa ... 127
Przykład ... 129
Konsekwencje użycia sztafety wywołań ... 130
Podsumowanie ... 130

Rozdział 6. Obiektowy model dokumentu Level 2 (DOM2) ... 131

Wprowadzenie ... 131
Historia DOM ... 132
Czym jest DOM ... 133
Mała aplikacja DOM ... 135
Problem ... 135
Rozwiązanie ... 136
Budowa DOM ... 138
Przewodnik po DOM ... 138
Główne interfejsy DOM ... 139
Bardziej złożony przykład ... 143
Podsumowanie ... 148

Rozdział 7. Zaawansowane techniki DOM2 ... 149

Wprowadzenie ... 149
Aplikacje DOM2 wykorzystujące szablon Visitor ... 149
Użycie szablonu Visitor ... 150
Przykład ... 150
Struktura ... 156
Przykład ... 157
Konsekwencje zastosowania szablonu Visitor ... 158
Aplikacje DOM2 wykorzystujące szablon Iterator ... 159
Użycie szablonu Iterator ... 159
Przykład ... 159
Budowa szablonu Iterator ... 163
Przykład ... 164
Konsekwencje zastosowania szablonu Iterator ... 166
Aplikacje DOM2 wykorzystujące szablon Mediator ... 166
Użycie ... 167
Przykład ... 167
Struktura ... 170
Przykład ... 170
Konsekwencje zastosowania szablonu Mediator ... 172
Podsumowanie ... 173

Część III Technologie pomocnicze XML 175

Rozdział 8. XPath — język ścieżek XML 177

Wprowadzenie 177
XPath — teoria empiryczna 177
Uwagi wstępne 177
Podstawy XPath 178
Narzędzia XPath 187
Tester ścieżek XPath 188
Rozszerzenia XPath w Emacs 189
Biblioteki 189
Programowe użycie XPath 190
Kilka szczegółów dotyczących XPath 191
Ścieżki lokalizacji 191
Etapy 191
Osie 191
Zapis skrócony 192
Podsumowanie 192

Rozdział 9. XPointer 193

Wprowadzenie 193
Podstawy XPointer 193
Zadania XPointer i związane z nim pojęcia 194
Model i język XPointer 195
Krótko o ścieżkach logicznych 195
Postaci przyjmowane przez XPointer 195
Cytowanie w XPointer 197
Cytowanie w URI XPointer 197
Cytowanie XML w XPointer 198
Cytowanie XPointer 199
Rozszerzenia XPath w XPointer 199
Nowe zadania XPointer w XPath 199
Nowe funkcje w XPath 201
Dodatki do XPath — podsumowanie 203
Narzędzia XPointer 203
Podsumowanie 204

Rozdział 10. XLink 205

Wprowadzenie 205
Powiązania między danymi XML 205
Elementy XLink 207
XLink rozszerzone i proste na przykładach 208
Przykładowa struktura 209
Znaczniki XLink 216
Łącza rozszerzone 217
Łącza proste 220
Prezentacja łączy XLink 221
Podsumowanie: atrybuty globalne XLink i zasady ich użycia 222

Przykład łączy XLink nie związanych z prezentacją ... 223
O grafach ... 224
Użycie XLink przy opisywaniu w XML grafów skierowanych ... 224
Modelowanie grafów w Javie ... 226
Tworzenie grafów Javy na bazie grafów XML ... 227
Podsumowanie ... 227

Rozdział 11. XSLT: przekształcanie XML ... 229

Wprowadzenie ... 229
Ogólne wiadomości o XSLT ... 229
Podstawy ... 229
Proste przekształcenie XSLT ... 230
Dodatkowe wiadomości o przekształceniach ... 233
Działanie na węzłach ... 233
Wizualizacja sposobu przekształcania ... 234
Procesory XSLT ... 235
Instalacja i wywołanie Xalan ... 236
Instalacja i wywołanie XT ... 236
Wszystko o języku XSLT ... 237
Dane źródłowe ... 237
Wstawianie nowych elementów i atrybutów ... 238
Wstawianie tekstu ... 241
Wstawianie generowanego tekstu ... 242
Kopiowanie ... 243
Numerowanie ... 243
Wyrażenia warunkowe ... 245
Pętle ... 247
Sortowanie ... 247
Zmienne ... 250
Reguły nazwane ... 251
Przekazywanie regułom parametrów ... 252
Podsumowanie ... 254

Rozdział 12. XSLT: zaawansowane techniki i ich zastosowanie ... 255

Wprowadzenie ... 255
Od struktury do prezentacji: wymagania funkcjonalne ... 255
Wymagania ... 256
Modelowanie informacji ... 256
Przekształcanie w XHTML ... 259
Uwagi o przekształcaniu na format PDF ... 264
Graficzna postać danych: generacja SVG ... 264
Dane prezentowane graficznie ... 265
Modelowanie informacji ... 265
Pokazywanie danych w postaci SVG ... 267
Przekształcanie danych statystycznych w obrazki SVG ... 269
Wielokrotne użycie arkuszy stylów ... 271
Dwie drogi do tego samego celu ... 272

Rozszerzanie XSLT ... 272
Kiedy rozszerzać XSLT ... 272
Rozwiązania bez rekursji ... 273
Implementacja instrukcji times ... 273
O innych rodzajach rozszerzeń ... 276
Inne rozszerzenia ... 276
Krótki przewodnik po XSLT ... 276
Podsumowanie ... 280

Rozdział 13. XML Schema ... 281

Wprowadzenie ... 281
XML Schema — krótkie wprowadzenie ... 281
Typy ... 282
Fazy ... 287
Typy anonimowe ... 292
Modele zawartości ... 293
Wielokrotne użycie elementów i atrybutów: grupy nazwane ... 295
Adnotacje ... 295
Podsumowanie ... 296

Rozdział 14. Zaawansowane techniki modelowania danych w XML Schema ... 297

Wprowadzenie ... 297
Zaawansowane zagadnienia związane z XML Schema ... 297
Wyprowadzanie nowych typów i inne relacje między typami ... 298
Niepowtarzalność ... 306
Modularyzacja i wielokrotne użycie ... 307
Porównanie XML Schema z DTD (Rick Jelliffe) ... 308
Podsumowanie ... 310

Część IV Najważniejsze aplikacje XML ... 313

Rozdział 15. XML bezprzewodowe: WAP, VoiceXML i inne ... 315

Wprowadzenie ... 315
WAP ... 316
Zasady działania WAP ... 316
Model WAP ... 317
Struktura WAP ... 318
WAE ... 318
WML ... 319
Środowisko tworzenia WAP ... 320
Użycie WML ... 321
Wszystko o WML ... 322
Architektura ... 332
WMLScript ... 336
Struktura WMLScript ... 336
VoiceXML ... 343
Podsumowanie ... 344

Rozdział 16. XML i bazy danych 345
Wprowadzenie 345
Bazy danych a XML 345
Typy dokumentów 346
Stopień powiązania 348
Specjalne klasy dla poszczególnych słowników XML 349
API rozdzielający bazę danych 352
Narzędzia warstwy pośredniej 356
Szablony a odwzorowanie 357
xml-dbms 358
Podsumowanie: kompletny przykład 360
Przekształcanie metadanych 364
System zarządzania treścią 364
Serwery XML jako bazy danych 365
Lista dostępnych produktów 368
Podsumowanie 369

Rozdział 17. XML między serwerami: XML-RPC i B2B 371
Wprowadzenie 371
XML-RPC 371
Podstawy zdalnego wywoływania procedur 372
Postać wywołania 373
Postać odpowiedzi 373
Postaci informacji o błędzie 374
Testowanie serwisów XML-RPC 375
Zalety i wady XML-RPC 376
Praktyczne wprowadzenie do programowania XML-RPC 378
Szczegóły specyfikacji XML-RPC 381
Aplikacje B2B tworzone przy użyciu XML-RPC — porównanie cen DVD 386
Architektura 387
Serwis DVDTitles 388
Serwer DVDTitles 390
Klient DVDTitle 391
Serwis DVDPrices 392
Podsumowanie 395

Część V Obszerne omówienie poszczególnych przykładów 397

Rozdział 18. XML a wygląd: CSS2, XHTML, SVG i SMIL 399
Wprowadzenie 399
XML i CSS 399
Praktyczne wprowadzenie do pracy z CSS i XML 400
Składnia i dostępne mechanizmy 406
Model ramek prezentacji w CSS 414
Kompletny przykład 415
Podsumowanie 420
Zestawienie właściwości 421

Rozdział 19. Aplikacja przeznaczona do usuwania błędów 429
Wprowadzenie 429
Wymagania 430
Wymagania względem XMLBugTrack (wygenerowane przez req2txt.xsl) 430
Wymagania wobec XMLBugTrack (wygenerowane przez req2xhtml.xsl) 432
Wymagania wobec XMLBugTrack (oryginalny XML) 432
Analiza i projekt 435
Struktura aplikacji 435
Projekt 436
Implementacja 440
Połączenie bazy danych i XML, przetwarzanie XSLT 441
Podsumowanie 455

Rozdział 20. Aplikacja przeznaczona do zarządzania informacjami 457
Wprowadzenie 457
Na czym polega zarządzanie informacjami 458
Wymagania 458
Budowa systemu 462
Wykorzystanie DTD 463
DTD komentarzy 463
Słownik 470
Edytor dokumentów z notatkami 473
Wykorzystanie SMIL/HTML przy tworzeniu prezentacji 475
Rzut oka na wyniki 475
Podsumowanie 482

Dodatki 483

Dodatek A Składnia XML 485
Elementy i atrybuty 485
Deklaracje DTD 486
Odwołania do encji 486

Dodatek B Specyfikacja XML 1.0 487
Rozszerzalny Język Znaczników (XML) 1.0 (wydanie drugie) 487
Rekomendacja W3C z 6 października 2000r. 487
Streszczenie 488
Status tego dokumentu 488
1. Wprowadzenie 489
1.1. Historia dokumentu i jego zadania 489
1.2. Terminologia 490
2. Dokumenty 491
2.1. Poprawne składniowo dokumenty XML 491
2.2. Znaki 492
2.3. Typowe konstrukcje składniowe 492
2.4. Dane znakowe i znaczniki 494
2.5. Komentarze 495

2.6. Instrukcje przetwarzania ... 495
2.7. Sekcje CDATA ... 495
2.8. Prolog i deklaracja typu dokumentu ... 496
2.9. Deklaracja samodzielności dokumentu ... 499
2.10. Jak traktować białe znaki? ... 500
2.11. Obsługa końca wiersza ... 501
2.12. Określanie języka ... 501
3. Struktury logiczne ... 502
3.1. Znaczniki początkowe, końcowe i znaczniki elementów pustych ... 503
3.2. Deklaracje typu elementu ... 504
3.3. Deklaracje list atrybutów ... 506
3.4. Sekcje warunkowe ... 511
4. Struktury fizyczne ... 512
4.1. Znaki i encje ... 512
4.3. Encje parsowane ... 516
4.4. Sposób traktowania encji i odwołań przez procesor XML ... 518
4.5. Konstrukcja tekstu podstawienia encji wewnętrznej ... 520
4.6. Encje predefiniowane ... 521
4.7. Deklaracje notacji ... 521
4.8. Encja dokumentu ... 522
5. Zgodność ... 522
5.1. Procesory z walidacją i bez walidacji ... 522
5.2. Użycie procesorów XML ... 523
6. Wybór notacji ... 523
A. Bibliografia ... 525
A.1. Dokumenty normatywne ... 525
A.2. Inne publikacje ... 526
B. Klasy znaków ... 529
C. XML a SGML (nienormatywne) ... 531
D. Rozwijanie odwołań do encji i znaków (nienormatywne) ... 531
E. Deterministyczne modele zawartości (nienormatywne) ... 533
F. Autodetekcja kodowania znaków (nienormatywne) ... 533
F.1. Detekcja w przypadku braku informacji zewnętrznych ... 534
F.2. Priorytety przyjęte wobec istnienia zewnętrznych informacji o kodowaniu ... 535
G. Grupa robocza XML w W3C (nienormatywne) ... 536
H. Skład grupy W3C XML Core (nienormatywne) ... 536
I. Uwagi o przygotowaniu specyfikacji (nienormatywne) ... 537

Dodatek C Najważniejsze DTD przeznaczone do prezentacji ... 539

WML ... 539
XHTML (wersja ścisła) ... 545
SMIL ... 562

Dodatek D Krótki przewodnik po UML ... 567

Diagram klas ... 567
Diagramy przypadków użycia ... 568

Dodatek E Najważniejsze przykłady wyspecjalizowanych DTD 571
Wprowadzenie 571
Wymagania funkcjonalne 571
Encje pomocnicze 573
DTD komentarzy (wersja prosta) 575
Skorowidz 577

Rozdział 16.

XML i bazy danych

Wprowadzenie

Tworzenie dużych witryn sieciowych oraz innych aplikacji zależy od możliwości przechowywania, wyszukiwania i pobierania dużych ilości danych. Mimo że dokumenty XML są elastyczne i niosą w sobie dużo treści, to z uwagi na wydajność wymaganą przy przechowywaniu dużych ilości danych, nie mogą równać się z tradycyjnymi bazami danych, jak np. Oracle czy PostgreSQL.

W tym rozdziale przyjrzymy się, jak wyglądają związki między XML a bazami danych z punktu widzenia twórcy systemów. Szczególnie będą nas interesowały praktyczne ich zastosowania. Nie będziemy zajmować się podstawowymi konstrukcjami baz danych, zakładamy znajomość samego XML i zagadnień omawianych w rozdziałach 4. i 6. (szczególnie interfejsu SAX).

Rozdział ten podzielony został na kilka części: w pierwszej omawiamy podstawowe zagadnienia ogólne, związane z bazami danych i XML oraz sposób ich zapisania w pięciu różnych zastosowaniach. Następnie dokładnie objaśnimy poszczególne zastosowania oraz pokażemy przykłady ich implementacji. Dodatkowo pokażemy zestawienia najlepszych bibliotek i produktów, podzielone według kategorii.

Bazy danych a XML

XML jest szczególnie przydatny przy przenoszeniu danych i opisywaniu ich semantyki. Dzięki niemu otrzymujemy metodę przejrzystego zapisania danych i metadanych w hierarchicznym dokumencie, łatwo będzie nam takie dokumenty przenosić. Z kolei bazy danych są użyteczne wtedy, kiedy trzeba przechowywać duże ilości danych w wydajny sposób.

Podczas tworzenia dużych aplikacji sieciowych XML (przykład w rozdziale 19.) lub jakichkolwiek innych aplikacji, w których mamy do czynienia z dużą ilością danych, przechowywanie tych danych bezpośrednio, w postaci dokumentów XML staje się coraz mniej opłacalne. W takich sytuacjach konieczne staje się powiązanie dwóch technologii: XML i klasycznych baz danych.

Połączenie to może przybierać różne postaci, w zależności od obsługiwanych dokumentów, zadań stawianych przed aplikacją i od tego, jak intensywne będzie przekształcanie danych z dokumentów XML w tabele (obiekty baz danych) i odwrotnie.

Typy dokumentów

Z punktu widzenia trwałości danych, rozróżnia się *XML opisujący dokumenty* i *XML opisujący dane*.

XML opisujący dane ma sztywne struktury, które są standardowymi, powtarzalnymi obiektami, składającymi się z atrybutów. W tego typu dokumentach dane typu `PCDATA` występują zwykle jedynie jako elementy-liście, rzadko zdarza się też zawartość mieszana.

Poniższy kod stanowi przykład XML opisującego dane (kolekcja płyt kompaktowych). Inne przykłady to dokumenty wysyłki i pliki konfiguracyjne.

```
<?xml version="1.0"?>
<!DOCTYPE kolekcja_cd SYSTEM "cd.dtd">
<kolekcja_cd>
  <tytuł xml:space="preserve">Przykładowy fragment
                              kolekcji CD</tytuł>
  <cd id="a9362-43515-2">
    <tytuł xml:space="preserve">In our sleep</tytuł>
    <artysta>Laurie Anderson &amp;amp; Lou Reed</artysta>
    <producent>Brian Eno</producent>
    <rok>1995</rok>
    <utwór numer="1">In our sleep</utwór>
    <utwór numer="2">...</utwór>
  </cd>
  <cd id="a9233-436432-3">
    <tytuł xml:space="preserve">Trans-Europe Express</tytuł>
    <artysta>Kraftwerk</artysta>
    <producent>Kraftwerk</producent>
    <rok>1978</rok>
    <utwór numer="1">Europe Endless</utwór>
  </cd>
  <!-- tutaj może być mnóstwo dalszych płyt -->
</kolekcja_cd>
```

W przypadku XML opisującego dokumenty, gdy prezentowane w nich dane mają być przede wszystkim czytelne, struktura jest bardziej swobodna, często pojawia się zawartość mieszana. Tak jest w przypadku większości witryn sieciowych, listów i innych dokumentów zakodowanych w XML. Poniższy urywek artykułu (zakodowany za pomocą DTD Docbook) należy do takiego właśnie typu dokumentów:

```
<!DOCTYPE article PUBLIC "-//Davenport/DTD DocBook V3.0/EN" [

<!ENTITY email            "fabio@viaduct.com">
<!ENTITY bulletType       "circle">
<!ENTITY version          "$Id: main.xml,v 1.5 2000/08/18 13:51:12 faa Exp $">
<!ENTITY status           "draft 1">
<!ENTITY project          "Wprowadzenie do XLink">
<!ENTITY abstract         SYSTEM "abstract.xml">
<!ENTITY what             SYSTEM "what.xml">
<!ENTITY intro            SYSTEM "intro.xml">
<!ENTITY why              SYSTEM "why.xml">
<!ENTITY how              SYSTEM "how.xml">
<!ENTITY references       SYSTEM "glossary.xml">
]>

<article>
  <artheader>
    <title>XLink objaśnione</title>
    <title>Wprowadzenie do XLink</title>
    <author>
      <surname>Arciniegas A.</surname>
      <firstname>Fabio</firstname>
    </author>
    <!-- &abstract; -->
  </artheader>
  &intro;
  &what;
  &why;
  &how;
  &references;
</article>
```

Oto jeden z plików z danymi, *abstract.xml*:

```
<part id="intro-part">
<docinfo>
  <title>Streszczenie</title>
</docinfo>
<title>Streszczenie</title>

<chapter id="Main-Introduction">
  <title>Wprowadzenie do Heqet:
         ogólnego edytora XML dla Gnome</title>
<para>
  Niewiele technologii z <a href="family">rodziny XML</a>
  było oczekiwanych tak powszechnie i z taką niecierpliwością,
  jak <emphasis>XLink</emphasis>...
</para>
<!-- ... tutaj ciąg dalszy -->
```

W tej chwili różnica między XML opisującym dane a służącym do zapisu dokumentów będzie miała dla nas kapitalne znaczenie, gdyż wiąże się z nią zasadnicze pytanie: czy interesują nas *dane* zapisane w XML, czy sam XML?

W przypadku dokumentów interesują nas raczej same dokumenty: budowa dokumentu, zawarte w nim komentarze, użycie encji i tak dalej — wszystko, co dotyczy opisywanego problemu.

W przypadku XML opisującego dane interesują nas głównie (lub wyłącznie) dane znajdujące się w dokumencie. To, czy dokument składa się z pięciu zewnętrznych encji, czy też jest zapisany w całości zapisany w jednym pliku, nie ma znaczenia[1]. Tak samo nie mają znaczenia komentarze, wcięcia, a czasem nawet kolejność elementów (kolejność akapitów w liście jest bardzo ważna, natomiast kolejność pozycji faktury nie ma już takiego znaczenia).

W większości aplikacji związanych z bazami danych XML traktuje się jako źródło danych, zaś kluczowe aspekty to: wydajne przetwarzanie dużych ilości dokładnych danych (przykładami mogą być spis towarów w hurtowni, zbiór danych geograficznych i tak dalej). Dlatego w tym rozdziale poświęcimy dużo uwagi narzędziom i technikom potrzebnym do przekazywania danych między XML a bazami danych.

Stopień powiązania

Natkniemy się też na problem z określeniem, ile aplikacja „wie" o bazie danych, czyli na ile jest z nią powiązana. Tak naprawdę jest to pytanie o to, na ile warstwa aplikacji odpowiada warstwie bazy danych — może być to związek od bardzo ścisłego aż po całkowitą niezależność.

Z bardzo ścisłym związkiem mamy do czynienia wtedy, gdy *klasy specjalizowane* znają strukturę bazy danych, potrafią ustanawiać połączenia i tworzyć zapytania oraz generować kod XML na podstawie wyników.

Kiedy związek ten jest bardzo luźny, aplikacja nie wie nic o różnych sposobach zapisu danych i po prostu wywołując odpowiednie funkcje API serwera XML, otrzymuje dokument XML.

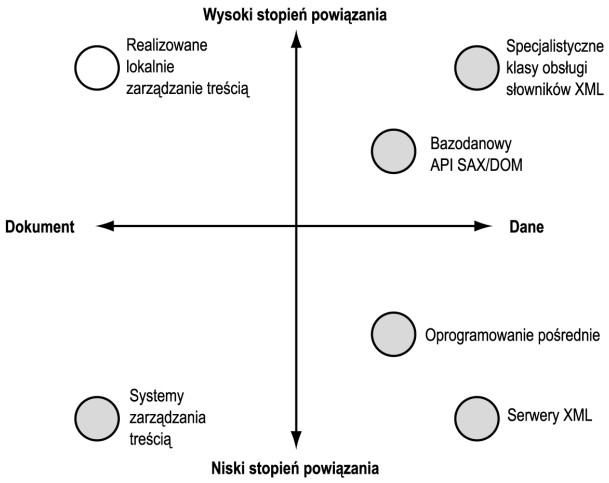
W przypadku powiązania pośredniego stopnia, do połączenia tradycyjnej bazy danych (zwykle relacyjnej, jak PostreSQL czy MS SQL) z dokumentami XML używa się oprogramowania pośredniego.

Z analogiczną warstwą oprogramowania pośredniego mamy do czynienia, jeśli używany jest parser, pobierający tabele i generujący zdarzenia SAX. Parser taki (będący też pewnego rodzaju oprogramowaniem pośrednim) umieszczony jest między aplikacją a bazą danych i tworzy w ten sposób interfejs do bazy relacyjno-obiektowej.

Jeśli uwzględnimy wspomniane dwa czynniki, otrzymamy diagram taki, jak na rysunku 16.1.

Wybranie odpowiedniej technologii z powyższego diagramu zależy od trzeciego czynnika, o którym wspomniano na początku tego rozdziału: wymagań wobec aplikacji. Omówione w następnych punktach przykłady i teoria zilustrują zastosowanie omawianych zasad w praktyce i pokażą, jak podejmować decyzję o doborze technologii właściwej dla danego projektu.

[1] Stoi to w jawnej sprzeczności z XML opisującym dokumenty, w którym podział na encje jest podstawowym mechanizmem modularyzacji i możliwości wielokrotnego użycia.



Rysunek 16.1. *Diagram, przedstawiający sposoby powiązania XML i baz danych*

Specjalne klasy dla poszczególnych słowników XML

Pierwszym, najprostszym sposobem tworzenia dokumentu XML na podstawie bazy danych jest jawne zapisanie logiki dla konkretnego słownika XML w specjalizowanej klasie. Klasa taka jest ściśle powiązana z bazą danych i XML. Mimo że takie rozwiązanie bardzo poprawia wydajność, to jest ono ograniczone i trudne do rozszerzania i utrzymania.

Niezależnie od wspomnianych ograniczeń, takie „domowe" rozwiązanie jest czasem przydatne. Poniższy kod to implementacja specjalizowanej klasy, pobierającej z tabeli zbiór rekordów dotyczących filmów:

```
import java.sql.  Connection;
import java.sql.  DriverManager;
import java.sql.  Statement;
import java.sql.  ResultSet;
import java.util. Properties;
import java.util. Properties;
import java.io.   FileInputStream;


public class CustomTable2XML {
    public static void main(String[] argv) {
        if (argv.length != 1) {
            System.out.println("Użycie: java
                                CustomTable2XML  PlikWłaściwości");
            System.exit(-1);
        }

        Connection conn = null;
        Statement stmt = null;
```

```
    ResultSet rs = null;
    Properties myDB = new Properties();
    try
    {
     myDB.load(new FileInputStream(argv[0]));
     Class.forName(myDB.getProperty("driver"));
     conn = DriverManager.getConnection(
                          myDB.getProperty("URL"),
                          myDB.getProperty("user"),
                          myDB.getProperty("password"));
     stmt = conn.createStatement();
     rs = stmt.executeQuery("SELECT * FROM " +
               myDB.getProperty("table"));
     createXMLDocument(rs);
    }
    catch(java.lang.ClassNotFoundException cnfe)
    {
        System.out.println("Nie znaleziono sterownika");
        System.exit(-1);
    }
   catch (java.sql.SQLException sqle)
   {
        System.out.println("Nie powiodło się połączenie
                            z bazą danych (SQL) " + sqle);
        System.exit(-1);
    }
   catch (java.lang.Exception e)
    {
        System.out.println("Nie powiodło się połączenie
                            z bazą danych " + e);
        System.exit(-1);
    }
}

// Dane z tabeli kierowane są do STDOUT jako XML
public static void createXMLDocument(ResultSet rs)
                  throws java.sql.SQLException,
                  java.io.IOException
{
    System.out.println("<?xml version=\"1.0\"?>");
    System.out.println("<!-- Plik wygenerowany
                        na podstawie bazy danych Movies-->");
    System.out.println("<movies>");

    // Program "zna" strukturę tabeli, struktura ta została
    // zakodowana na stałe
    while(rs.next())
    {
        System.out.println("<dvd id='" +
                  rs.getInt("DVDID") + "'");
        System.out.println("     releasedDate='" +
                  rs.getDate("Released").toString() + "'>");
        System.out.println("   <title> "   +
                  rs.getString("DVDTitle") + "</title>");
        System.out.println("   <tagline> " +
                  rs.getString("Tagline") + "</tagline>");
        System.out.println("   <category> " +
                  rs.getString("Category") + "</category>");
```

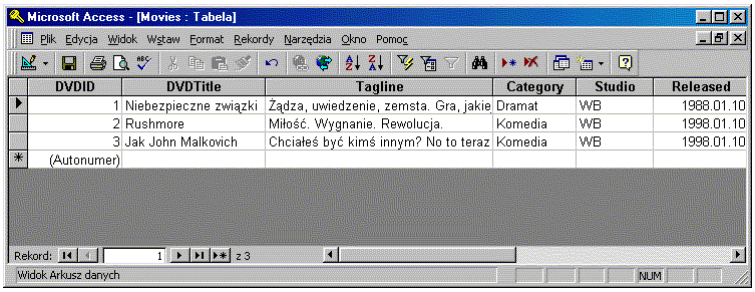
```
            System.out.println("   <studio> " +
                               rs.getString("Studio") + "</studio>");
            System.out.println("</dvd>");
        }
        System.out.println("</movies>");

    }
}
```

Jeśli tabela zawiera dane takie, jak na rysunku 16.2, a plik właściwości ma postać:

```
# Plik właściwości CustomTable2XML
URL=jdbc:odbc:DTDs
user=
password=
table=Movies
driver=sun.jdbc.odbc.JdbcOdbcDriver
```

Rysunek 16.2.
Tabela Movies

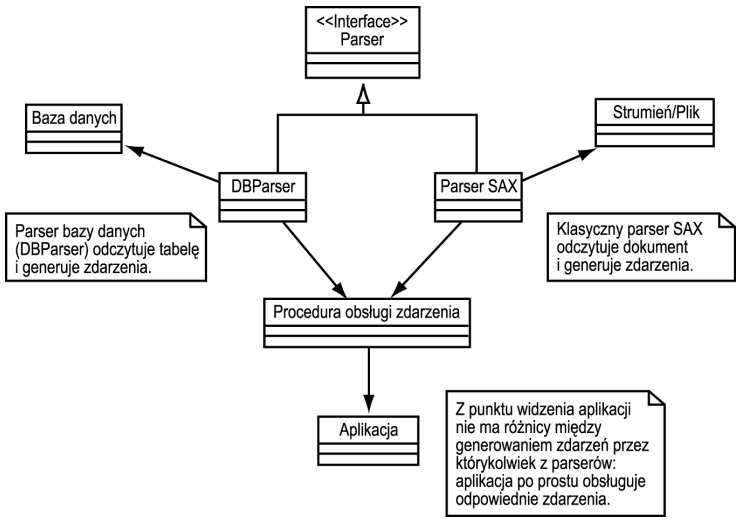


To wygenerowany zostanie następujący dokument:

```
<?xml version="1.0"?>
<!-- Plik wygenerowany na podstawie bazy danych Movies -->
<movies>
<dvd id='1'
     releasedDate='1988-10-01'>
   <title> Niebezpieczne związki</title>
   <tagline> Żądza, uwiedzenie, zemsta. Gra, jakiej nie znasz.</tagline>
   <category> Dramat</category>
   <studio> WB</studio>
</dvd>
<dvd id='2'
     releasedDate='1998-10-01'>
   <title> Rushmore</title>
   <tagline> Miłość. Wygnanie. Rewolucja.</tagline>
   <category> Komedia</category>
   <studio> WB</studio>
</dvd>
<dvd id='3'
     releasedDate='1998-10-01'>
   <title> Być jak John Malkovich</title>
   <tagline> Chciałeś być kimś innym? No to już możesz.</tagline>
   <category> Komedia</category>
   <studio> WB</studio>
</dvd>
</movies>
```

API rozdzielający bazę danych

Kiedy zmniejszymy stopień powiązania oprogramowania z danymi, natkniemy się na ciekawe rozwiązanie: parsery, które pobierają dowolne tabele[2] jako dane wejściowe, a jako dane wyjściowe generują standardowe zdarzenia SAX lub DOM. Z punktu widzenia reszty aplikacji, nie ma różnicy między takim parserem a zwykłym parserem XML. Całe przetwarzanie bazy danych zostaje oddzielone w klasie parsera, zaś aplikacja może zająć się jedynie przetwarzaniem procedur obsługi, potrzebnych do obsługi zdarzeń SAX (rysunek 16.3).



Rysunek 16.3. *API SAX do baz danych*

Poniższy kod to klasa `DBParser`, implementująca interfejs parsera SAX:

```
import org.xml.sax.         *;
import org.xml.sax.helpers. LocatorImpl;
import org.xml.sax.helpers. AttributeListImpl;
import java.lang.           Integer;

import java.sql.            ResultSet;
import java.sql.            ResultSetMetaData;
import java.sql.            SQLException;
import java.io.             IOException;
import java.util.           Locale;
import java.util.           Vector;


public class DBParser implements Parser
```

[2] W tym rozdziale częściej będziemy mówić o tabelach i relacjach niż o obiektach. Przyczyna tego jest prosta: w przypadku baz danych model relacyjny jest stosowany znacznie częściej niż model obiektowy. Jednak te same zasady działania i te same rozwiązania można bez trudności zastosować w przypadku obiektowych baz danych.

```
{

    static HandlerBase s_defaulthandler =  new HandlerBase();
    EntityResolver  entityHandler   =   s_defaulthandler;
    DTDHandler      dtdHandler      =   s_defaulthandler;
    DocumentHandler documentHandler =   s_defaulthandler;
    ErrorHandler    errorHandler    =   s_defaulthandler;
```

Najpierw muszą być zaimplementowane wszystkie metody interfejsu parsera:

```
public void setEntityResolver(EntityResolver handler) {
        entityHandler = handler;
    }

    public void setDocumentHandler(DocumentHandler handler) {
        documentHandler = handler;
    }

    public void setErrorHandler(ErrorHandler handler) {
        errorHandler = handler;
    }

    public  void setDTDHandler (DTDHandler handler) {
        dtdHandler = handler;
    }

  public void parse (InputSource source)
    throws SAXException, IOException
    {
        throw new SAXException("Jako dane wejściowe DBParser
                       musi otrzymać obiekt typu ResultSet");
    }


  public void parse (String systemId)
    throws SAXException, IOException
    {
        throw new SAXException("Jako dane wejściowe DBParser
                        musi otrzymać obiekt typu ResultSet");
    }

  public void setLocale (Locale locale)
    throws SAXException
    {
        // w tej wersji nie zaimplementowano ustawień lokalnych.
        // kod jest zgodny z SAX2, generowany jest wyjątek.
        throw new SAXException("Ustawienia lokalne
                                nie są obsługiwane");
    }
```

Zaimplementowane są już zatem wszystkie metody obowiązkowe i możemy zająć się podstawową częścią przykładu: metodą `parse`, umożliwiającą generowanie zdarzeń SAX na podstawie zbioru wynikowego.[3]

[3] Klasę tę można bardzo łatwo zmodyfikować tak, aby móc pobierać inne obiekty danych, jak np. połączenie czy zapytanie.

Nasza metoda `parse` otrzymuje cztery parametry.

- `Resultset rs` — obiekt `ResultSet`, na podstawie którego generowany będzie kod XML.
- `String rootName` — nazwa znacznika używanego jako element główny.
- `String rowTagName` — każdemu wierszowi danych odpowiada element o nazwie `rowTagName` (jego podelementy są polami w wierszu). W tym parametrze podać należy nazwę elementu zawierającego wiersz danych.
- `Vector ra` — wektor ten zawiera indeksy kolumn, które muszą przyjąć postać atrybutów. Jeśli nie zostanie w tym wektorze umieszczony indeks kolumny, dane tej kolumny będą wstawiane jako elementy.

```
//*****************************************************
// Metody specyficzne dla DBParser
//*****************************************************
public void parse(ResultSet rs,String rootName,
                  String rowTagName, Vector ra)
    throws SAXException,SQLException
{
    AttributeList emptyAL = new AttributeListImpl();
    ResultSetMetaData md = rs.getMetaData();
    int cols = md.getColumnCount();

    documentHandler.startDocument();
    documentHandler.startElement(rootName,emptyAL);

    while(rs.next())
    {
        AttributeListImpl atts = new AttributeListImpl();

        // pobierz pierwszą kolumnę zapisywaną jako atrybut
        for(int i = 0; i < ra.size(); i++)
        {
            int col = ((Integer)ra.elementAt(i)).intValue();
            String atName =  md.getColumnLabel(col);
            String atValue = rs.getString(col);
            atts.addAttribute(atName,"CDATA",atValue);
        }

        documentHandler.startElement(rowTagName,atts);

        // pobierz kolumny, które muszą być opisywane za pomocą
        //     podelementów
        for(int i = 1; i <= cols; i++)
        {
            if(!ra.contains(new Integer(i)))
            {
                String eleName =  md.getColumnLabel(i);
                String eleContent = rs.getString(i);
                if (eleContent == null)
                    break;
                char[] eleContentCA = eleContent.toCharArray();

                documentHandler.startElement(eleName,emptyAL);
```

```
                    documentHandler.characters(eleContentCA,0,
                                               eleContentCA.length);
                    documentHandler.endElement(eleName);
                }
            }
            documentHandler.endElement(rowTagName);
        }

        documentHandler.endElement(rootName);
        documentHandler.endDocument();
        }
    }
```

Poniższy program testowy nakazuje parserowi odczytanie tej samej tabeli, co w poprzednim przykładzie. Tym razem jednak logika struktury XML nie jest już zakodowana na stałe, zadanie to zlecane jest parserowi SAX, przy którym można zarejestrować dowolne procedury obsługi zdarzeń[4] tak, jakby czytanie odbywało się z pliku XML:

```
import java.sql.  Connection;
import java.sql.  DriverManager;
import java.sql.  Statement;
import java.sql.  ResultSet;
import java.util. Properties;
import java.util. Vector;
import java.io.   FileInputStream;


public class TestSAXAPI {
    public static void main(String[] argv) {
        if (argv.length != 1) {
            System.out.println("Użycie: java CustomTable2XML
                                PlikWłaściwości");
            System.exit(-1);
        }

        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        Properties myDB = new Properties();
        try
        {
         myDB.load(new FileInputStream(argv[0]));
         Class.forName(myDB.getProperty("driver"));
         conn = DriverManager.getConnection(
                     myDB.getProperty("URL"),
                     myDB.getProperty("user"),
                     myDB.getProperty("password"));
         stmt = conn.createStatement();
         rs = stmt.executeQuery("SELECT * FROM " +
                                myDB.getProperty("table"));
```

[4] W programie testowym użyto klasy `SAXPrintHandler` (aby wypisać zdarzenia otrzymywane tak, jakby zostały one odczytane z dokumentu XML). Klasa ta była omówiona w rozdziałach 4. i 5., jej kod znajduje się na dołączonej do książki płycie.

Ponieważ nasza procedura `SAXPrintHandler` zwraca otrzymane zdarzenia w postaci XML, po wywołaniu:

```
C:\xmldb\SAXApi>java TestSAXAPI properties.prop
```

otrzymamy następujący dokument XML:

```
<movies>
<dvd DVDID="1" Studio="WB">
<DVDTitle>
Niebezpieczne związki</DVDTitle>
<Tagline>
Żądza, uwiedzenie, zemsta. Gra, jakiej nie znasz.</Tagline>
<Category>
Dramat</Category>
<Released>
1988-10-01 00:00:00</Released>
</dvd>
<dvd DVDID="2" Studio="WB">
<DVDTitle>
Rushmore</DVDTitle>
<Tagline>
Miłość. Wygnanie. Rewolucja.</Tagline>
<Category>
Komedia</Category>
<Released>
1998-10-01 00:00:00</Released>
</dvd>
<dvd DVDID="3" Studio="WB">
<DVDTitle>
Być jak John Malkovich</DVDTitle>
<Tagline>
Chciałeś być kimś innym? No to już możesz.</Tagline>
<Category>
Komedia</Category>
<Released>
1998-10-01 00:00:00</Released>
</dvd>
</movies>
```

Użycie API SAX jest zdecydowanie wygodniejsze od tworzenia klas specjalizowanych. Logika samej aplikacja jest oddzielona od tworzenia kodu XML, dzięki czemu otrzymujemy konstrukcję, której można użyć do wielu baz danych.

Pokazane tutaj rozwiązanie jest uniwersalne, ale ma też poważne ograniczenia — m.in. brak możliwości przejścia od XML do bazy danych (nie wspominając już o tym, że ograniczamy się na razie do jednej tabeli). W następnym punkcie pokażemy pełne rozwiązanie z oddzieloną warstwą pośrednią, która pozwoli nam takich problemów uniknąć.

Narzędzia warstwy pośredniej

Implementacja pełnego API, przekształcającego XML w bazę danych (i odwrotnie) przyjmuje postać oprogramowania warstwy pośredniej, które jest dostępne jako *open source* i jako produkt komercyjny.

Jeśli oprogramowanie to jest dostarczane wraz z komercyjną bazą danych, jest ono reklamowane jako część samej bazy. Baza ta z kolei reklamowana jest jako baza danych, obsługująca XML (*XML-enabled*).

Istnieje też szereg niekomercyjnych, niezależnych pakietów, zawierających API wykorzystywany do przekazywania danych między dokumentami XML a relacyjnymi bazami danych (zwykle przez ODBC lub JDBC). W tym punkcie pokażemy zasady rządzące większością rozwiązań komercyjnych i darmowych, a także pokażemy praktyczną implementację oprogramowania *open source* o nazwie *xm-dbms*, autorstwa Ronalda Bourreta.

Szablony a odwzorowanie

Kiedy stajemy przed koniecznością pobrania informacji XML z bazy danych, oprogramowanie pośrednie ogólnego zastosowania może działać dwojako:

- może umożliwiać włączanie instrukcji SQL do szablonów XML;
- może udostępniać jawne odwzorowanie między polami bazy danych a elementami XML.

W pierwszym wypadku dane wejściowe systemu byłyby dokumentem XML, zawierającym pewne elementy, które byłyby przez owo oprogramowanie pośrednie przekształcane na kod XML, który z kolei odpowiadający wynikom zapytania. Na przykład, poniższy dokument definiuje zapytanie do bazy danych w elemencie `select`:

```
<?xml version="1.0"?>
<piosenki>
   <opis>Opis pewnych melodii dżungli</opis>
   <select name="styl">SELECT name, opis
                       FROM StyleMuzyczne
                       WHERE type="dżungla"
   </select>
</piosenki>
```

Kiedy dokument ten zostanie przetworzony przez oprogramowanie pośrednie, element `select` zostanie zastąpiony danymi i uzyskamy:

```
<?xml version="1.0"?>
<piosenki>
   <opis>Opis pewnych melodii dżungli</opis>
   <styl>
      <tytuł>Jump-up</tytuł>
      <opis>Taneczne rytmy dżungli oparte na próbkach
            hip-hop.
      </opis>
   </styl>
   <styl>
      <tytuł>Liquid Funk</tytuł>
      <opis>Muzyka taneczna grana na bębnach i basie.
      </opis>
   </styl>
</piosenki>
```

W przypadku wyboru drugiej możliwości odwzorowania, danymi wejściowymi systemu są:

1. Odwzorowanie określające, w jaki sposób elementy i atrybuty są zapisywane w bazie danych.
2. Dokument XML lub dane z bazy danych.

Wynikiem generowanym przez oprogramowanie pośrednie są dane w bazie danych lub dane wejściowe zapisane jako XML.

Oprogramowanie oparte na szablonach przekształca tylko dane z bazy na XML. Poza tym, rozwiązania oparte na odwzorowaniach są łatwiej dostępne i lepiej udokumentowane. W pokazanym dalej przykładzie oprogramowania pośredniego skoncentrujemy się właśnie na API opartym na odwzorowaniach.

xml-dbms

Autorem pakietu xml-dbms jest Ronald Bourret, on też się tym pakietem opiekuje. Jest to doskonały przykład opartego na odwzorowaniach API, które służy do przekształcania danych XML w tabele i odwrotnie. Model xml-dbms oparty jest na języku odwzorowań, pozwalającym na jednoznaczne przekształcanie danych (na dwa sposoby). Tworzenie typowej aplikacji xml-dbms można zapisać w postaci trzech punktów:

1. Utworzenie obiektu odwzorowania na podstawie dokumentu odwzorowania.
2. Przekazanie obiektu odwzorowania obiektowi `domToDBMS` lub `DBMSToDom`.
3. Odebranie i przetwarzanie wyników przekształcenia.

W następnych punktach przekształcimy ten krótki schemat na konkretny kod.

Dokumenty odwzorowań

Dokument odwzorowania to dokument XML zgodny z DTD xml-dbms Map[5]. Dokumenty odwzorowań zawierają opis sposobu przekształcania, mającego miejsce między konstrukcjami bazodanowymi (jak tabele i kolumny) a XML (elementami i atrybutami). Pierwszym i najbardziej typowym sposobem odwzorowywania jest odwzorowanie typów elementów na tabele.

Zanim pokażemy takie odwzorowanie, trzeba wspomnieć, że język odwzorowań xml-dbms opiera się na założeniu, że w XML trzeba najpierw zamodelować jako klasy i atrybuty, a dopiero potem można przekształcać ten model na konstrukcje bazodanowe. Na szczęście obie te czynności wykonuje się w dokumencie odwzorowania:

```
<ClassMap>
   <ElementType Name="DaneGeograficzne"/>
   <ToClassTable>
      <Table Name="Miejsca"/>
```

[5] W innych rodzajach oprogramowania pośredniego stosowane są inne języki odwzorowań.

```
    </ToClassTable>
    ... odwzorowania właściwości ...
    ... powiązane odwzorowania klas ...
</ClassMap>
```

Istnieje już zatem relacja między typem elementu złożonego a tabelą, więc można powiązać atrybuty i elementy proste dokumentu z kolumnami tabeli:

```
<PropertyMap>
   <Attribute Name="Szerokość"/>
   <ToColumn>
      <Column Name="Szerokosc">
   </ToColumn>
</PropertyMap>
<PropertyMap>
   <Attribute Name="Długość"/>
   <ToColumn>
      <Column Name="Dlugosc">
   </ToColumn>
</PropertyMap>

<PropertyMap>
   <Attribute Name="miejsce"/>
   <ToColumn>
      <Column Name="NazwaMiejsca">
   </ToColumn>
</PropertyMap>
```

Inną możliwością języka xml-dbms jest odwzorowywanie relacji międzyklasowych (czyli modelowanie kluczy wiążących tabele) na hierarchię elementów:

```
<RelatedClass KeyInParentTable="placeID">
   <ElementType Name="OpisMiejsca"/>
   <CandidateKey Generate="No">
      <Column Name="placeID"/>
   </CandidateKey>
   <ForeignKey>
      <Column name="Czesc"/>
   </ForeignKey>
</RelatedClass>
```

Pokazane tutaj trzy konstrukcje wyczerpują zakres języka potrzebny przy większości zastosowań. Pełny opis wszystkich opcji języka odwzorowań xml-dbms znajduje się w DTD tego języka i w dokumentacji pakietu.

Tworzenie obiektu mapy

W programie Javy dokument odwzorowania musi zostać przekształcony w obiekt odwzorowania. Dokonuje się tego za pomocą specjalnych klas Javy, *map factory*. Klasy te mogą tworzyć obiekty odwzorowań na podstawie dokumentów języka xml-dbms lub na podstawie DTD (tworzenie odwzorowania na podstawie DTD omówimy w punkcie „Przekształcanie metadanych”).

Poniższy fragment kodu pokazuje tworzenie obiektu `Map` na podstawie pliku odwzorowania:

```
// Odwzorowanie łączy dokument z bazą danych, zatem
// konstruktor map factory otrzymuje jedno i drugie
factory = new MapFactory_MapDocument(conn, parser);

// Tworzenie obiektu odwzorowania na podstawie geo.map
map = factory.createMap(new InputSource(
              new FileReader("geo.map")));
```

Przekazywanie odwzorowania do dbsmToDOM i pobieranie danych

Tworzenie danych w postaci XML na podstawie bazy lub odwrotnie, często wykorzystywane jest w tylko jednym wierszu. Poniższy kod to przekształcanie bazy danych w XML; w pełnym przykładzie pokażemy też działanie w kierunku odwrotnym.

```
// Utworzenie nowego obiektu DBMSToDOM
   dbmsToDOM = new DBMSToDOM(map, new DF_Oracle());

   // utworzenie nowego klucza i pobranie związanych z nim danych
   key = {new Integer(123)};
   doc = dbmsToDOM.retrieveDocument("Sales", key);
```

Podsumowanie: kompletny przykład

Sprawdziliśmy już, jak pobrać dane XML z bazy (*vide* klasy specjalizowane i API SAX do baz danych). W tym przykładzie wstawimy dane z XML do bazy danych za pomocą klasy `DOMtoDBMS` pakietu xml-dbms.

Najpierw przyjrzyjmy się dokumentowi, jaki chcemy przetwarzać:

```
<?xml version="1.0"?>
<movies>
<dvd id='1'
     releasedDate='1988-10-01'>
   <title> Niebezpieczne związki</title>
   <tagline> Żądza, uwiedzenie, zemsta. Gra, jakiej nie znasz.</tagline>
   <category> Dramat</category>
   <studio> WB</studio>
</dvd>
<dvd id='2'
     releasedDate='1998-10-01'>
   <title> Rushmore</title>
   <tagline> Miłość. Wygnanie. Rewolucja.</tagline>
   <category> Komedia</category>
   <studio> WB</studio>
</dvd>
<dvd id='3'
     releasedDate='1998-10-01'>
   <title>Być jak John Malkovich</title>
   <tagline> Chciałeś być kimś innym? No to już możesz.</tagline>
   <category> Komedia</category>
   <studio> WB</studio>
</dvd>
</movies>
```

Oto plik *dvd.map*, zawierający reguły przekształcania danych:

```
<?xml version='1.0'?>
<!DOCTYPE XMLToDBMS SYSTEM "xmldbms.dtd" >

<XMLToDBMS Version="1.0">
   <Options>
   </Options>
   <Maps>
      <ClassMap>
         <ElementType Name="movies"/>
         <ToClassTable>
            <Table Name="dvd"/>
         </ToClassTable>
         <PropertyMap>
            <Attribute Name="id"/>
            <ToColumn>
               <Column Name="dvdid"/>
            </ToColumn>
         </PropertyMap>
         <PropertyMap>
            <Attribute Name="releaseDate"/>
            <ToColumn>
               <Column Name="release"/>
            </ToColumn>
         </PropertyMap>
         <PropertyMap>
            <ElementType Name="title"/>
            <ToColumn>
               <Column Name="title"/>
            </ToColumn>
         </PropertyMap>
         <PropertyMap>
            <ElementType Name="tagline"/>
            <ToColumn>
               <Column Name="tagline"/>
            </ToColumn>
         </PropertyMap>
         <PropertyMap>
            <ElementType Name="category"/>
            <ToColumn>
               <Column Name="category"/>
            </ToColumn>
         </PropertyMap>
         <PropertyMap>
            <ElementType Name="studio"/>
            <ToColumn>
               <Column Name="studio"/>
            </ToColumn>
         </PropertyMap>
      </ClassMap>
   </Maps>
</XMLToDBMS>
```

Nadszedł już czas na wykorzystanie programu *PutDVDs.java*, w którym w celu wstawienia danych z XML do bazy użyjemy oprogramowania warstwy pośredniej xml-dbms:

```
import de.tudarmstadt.ito.xmldbms.DBMSToDOM;
import de.tudarmstadt.ito.xmldbms.DOMToDBMS;
import de.tudarmstadt.ito.xmldbms.Map;
import de.tudarmstadt.ito.xmldbms.helpers.KeyGeneratorImpl;
import de.tudarmstadt.ito.xmldbms.mapfactories.MapFactory_MapDocu-
ment;

// Pominięte pozostałe importy (java.lang i org.apache)

public class PutDVDs {
 public static void main (String[] argv) throws java.sql.SQLException
 {
      Connection       conn1 = null, conn2 = null;
      Map              map;
      Document         doc;
      DOMToDBMS        domToDBMS;
      KeyGeneratorImpl keyGenerator = null;

      String mapFilename = argv[0];
      String xmlFilename = argv[1];
      try
      {
        if (argv.length != 2) {
            System.out.println("Użycie: java PutDVDs
                               odwzorowania plikXML");
            System.exit(-1);
        }

         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");


         // Podłączenie do bazy danych.
         conn1 = DriverManager.getConnection("jdbc:odbc:DTDs");
         conn2 = DriverManager.getConnection("jdbc:odbc:DTDs");

         // Utworzenie i inicjalizacja generatora kluczy.
         keyGenerator = new KeyGeneratorImpl(conn1);
         keyGenerator.initialize();

         // Utworzenie obiektu odwzorowań i otwarcie dokumentu XML.
         map = createMap(mapFilename, conn2);
         doc = openDocument(xmlFilename);

         // Utworzenie nowego obiektu DOMToDBMS
         // i przekształcenie danych.

         domToDBMS = new DOMToDBMS(map, keyGenerator,
                                   new NQ_DOM2());
         domToDBMS.storeDocument(doc);
      }
      catch(Exception e)
      {
          System.out.println(e);
      }
      finally
      {
         if (conn1 != null) conn1.close();
```

```
            if (conn2 != null) conn2.close();
         }

      }

      static Map createMap(String mapFilename, Connection conn)
             throws Exception
      {
         MapFactory_MapDocument    factory;

         // Create a new map factory and create the Map.
         factory = new MapFactory_MapDocument(conn, getSAXParser());
         return factory.createMap(new
                        InputSource(getFileURL(mapFilename)));
      }

      static String getFileURL(String fileName)
      {
         File   file;

         file = new File(fileName);
         return "file:///" + file.getAbsolutePath();
      }

      static Parser getSAXParser()
      {
         // UWAGA!  Kod specyfikczny dla parsera Xerces

         return new SAXParser();
      }

      static Document openDocument(String xmlFilename)
             throws Exception
      {

         DOMParser parser;

         // Uruchomienie parsera i ustawienie różnych opcji.
         parser = new DOMParser();
         parser.setFeature("http://xml.org/sax/features/namespaces",
                           true);

         // Analiza pliku wejściowego.
         parser.parse(new InputSource(getFileURL(xmlFilename)));

         // Zwrócenie drzewa DOM.
         return parser.getDocument();
      }
   }
```

Wynikiem uruchomienia tego programu z parametrami *dvd.map* i *dvd.xml* jest wypełnienie tabeli treścią dokumentu XML, zgodnie ze specyfikacjami w pliku odwzorowań. Bardziej złożone przykłady użycia xml-dbms znajdują się w dokumentacji tego pakietu (szczególnie wart uwagi jest przykład *transfer.java*, w którym zaprezentowano zaawansowane zastosowanie wszystkich omówionych tutaj technik).

Przekształcanie metadanych

Z poprzedniego punktu, dotyczącego oprogramowania pośredniego jasno wynika, że w większości wypadków do połączenia XML z bazą danych potrzebne jest odwzorowanie danej DTD na strukturę bazy danych.

Oczywiście, w tej sytuacji pojawia się następujące pytanie: jak określić strukturę bazy danych na podstawie DTD? Poniżej zestawiono kroki, jakie należy podjąć w celu zrealizowania tego zadania[6]:

1. Dla każdego typu elementu zawierającego elementy lub treść mieszaną, należy utworzyć tabelę i kolumnę klucza głównego.
2. Dla każdego elementu o zawartości mieszanej należy utworzyć odrębną tabelę na dane znakowe, powiązaną z tabelą nadrzędną za pomocą klucza głównego pierwszej tabeli.
3. Dla każdego jednowartościowego atrybutu danego elementu i dla każdego pojedynczo występującego elementu potomnego, zawierającego jedynie dane znakowe, należy utworzyć kolumnę. Jeśli element lub atrybut jest opcjonalny, należy zdefiniować kolumnę jako mogącą mieć wartość pustą.
4. Dla każdego wielowartościowego atrybutu i dla każdego powtarzającego się elementu potomnego należy utworzyć odrębną tabelę przeznaczoną na te wartości, a następnie połączyć tę tabelę z tabelą nadrzędną za pośrednictwem klucza głównego.
5. Dla każdego elementu potomnego, zawierającego elementy lub treść mieszaną, należy powiązać tabelę elementu „rodzica" z tabelą elementu „dziecka" za pomocą klucza głównego.

Z kolei, aby wygenerować DTD na podstawie bazy danych, trzeba po kolei:

1. Dla każdej tabeli utworzyć element.
2. Dla każdej kolumny z tabeli utworzyć atrybut lub element potomny, zawierający jedynie dane PCDATA.
3. Dla każdej relacji klucz główny/obcy w tabeli z kluczem głównym, należy utworzyć element potomny.

System zarządzania treścią

Objaśniliśmy już niemal wszystkie rozwiązania z diagramu 16.1. W tym punkcie omówimy rozwiązanie ukierunkowane na obsługę dokumentów: Systemy Zarządzania

[6] W przypadku implementacji automatycznej, kroki te mogą w pewnych szczegółach różnić się, w zależności od źródła. Podana tutaj lista została przepisana z doskonałego artykułu Rolanda Bourreta „XML i bazy danych".

Treścią (CMS, *Content Management System*). W następnym punkcie z kolei omówimy rozwiązanie ukierunkowane na dane, serwery XML.

CMS to szereg produktów, służących do tworzenia, edytowania i zarządzania różnymi wersjami dużej liczby dokumentów. Podstawowymi zadaniami CMS są:

1. Udostępnienie użytkownikowi sprawnego i prostego interfejsu, służącego do obsługi grup dokumentów oraz
2. Możliwość skalowania rozwiązania (w przypadku zwiększania się liczby obsługiwanych dokumentów).

CMS zwykle są ukierunkowane przede wszystkim na dostęp do nich przez interfejs użytkownika, mimo że niektóre z nich mają też API. Sposób traktowania treści w bardzo dużej mierze zależy od producenta[7], charakterystyczne jest położenie największego nacisku na obsługę dokumentu.

Z uwagi na opisane cechy CMS nie będziemy tutaj omawiać tych systemów; ograniczymy się do przedstawienia na końcu rozdziału produktów, które można zakwalifikować jako XML CMS.

Należy jeszcze wspomnieć, że pole w diagramie 16.1, odpowiadające tworzonym lokalnie systemom zarządzania treścią, jest puste, ponieważ tworzenie własnymi siłami rozwiązań tak dużych jak CMS jest stratą czasu i zasobów.

Serwery XML jako bazy danych

Serwery XML to kompletne platformy aplikacji, które dostarczają i odbierają dane w formacie XML jako podstawowym formacie wymiany danych. Zawierają API, służące do bezpośredniego przekształcania danych XML, które można je sklasyfikować jako najbardziej zwarte rozwiązanie przeznaczone do obsługi XML[8].

Mimo że trudno tu o zarysowanie wyraźnych granic, serwery XML różnią się od CMS dwoma zasadniczymi cechami:

- Serwery XML obsługują przede wszystkim dane. XML używany jest z uwagi na swoją przenośność i rozszerzalność, ale podstawowym jego problemem jest obsługa danych.
- Serwery XML nie są zorientowane na obsługę użytkownika. Mimo że niektóre z nich zawierają interfejs użytkownika, to głównym ich zadaniem jest tworzenie aplikacji i udostępnianie odpowiednich API.

[7] Może to się zmienić po pojawianiu się produktów takich, jak IM firmy Interleaf, który zwiększa zgodność ze standardami XML.

[8] Nie znaczy to oczywiście, że jest to najlepsze rozwiązanie dla wszystkich baz danych XML. Doświadczenie pokazuje nawet, że w przypadku projektów średniej wielkości lepiej skorzystać z oprogramowania warstwy pośredniej.

Interfejsy oferowane przez serwery XML są bardzo różne, więc najlepiej po prostu zająć się API konkretnego serwera. Aby jednak przedstawić przynajmniej podstawowe zasady użycia serwerów XML, pokazano przykład kodu API serwera dbxml[9]:

```
import com.dbxml.core.corba.*;
import com.dbxml.core.corba.database.*;
import com.dbxml.orblet.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;

import java.util.Properties;

public class APIExample {
   public static void main(String[] args) {
      try {
         Properties orbConfig = new Properties();

         /* Wskazanie orb serwisu nazewniczego. W przypadku
          * korzystania z innej konfiguracji parametry
          * domyślne mogą wymagać zmiany.
          */
         orbConfig.put("org.omg.CORBA.ORBInitialPort", "1997");
         orbConfig.put("org.omg.CORBA.ORBInitialHost", "localhost");

         // utworzenie i inicjalizacja ORB
         ORB orb = ORB.init(new String[0], orbConfig);

         // pobranie głównego kontekstu nazewnictwa
         org.omg.CORBA.Object objRef =
         orb.resolve_initial_references("NameService");
         NamingContext ncRef = NamingContextHelper.narrow(objRef);

         // rozwinięcie Object Reference w Naming
         NameComponent nc = new NameComponent("Application", "");
         NameComponent path[] = {nc};
         Application app = ApplicationHelper.narrow(ncRef.resolve(path));

         System.out.println("Application Name " + app.getName());

         // uruchomienie przykładów
         listCollections(app);
         String oid = insertDocument(app);
         retrieveDocument(app, oid);
      } catch (Exception e) {
         System.out.println("ERROR : " + e) ;
         e.printStackTrace(System.out);
      }
   }

   /**
     * Wyliczenie baz danych i ich zbiorów na serwerze
     */
```

[9] Aby ułatwić polskiemu czytelnikowi lekturę, w książce przetłumaczone zostały komentarze. Na dołączonej do książki płycie znajduje się oczywiście wersja oryginalna — *przyp. tłum.*

```
    public static void listCollections(Application app) throws Exception {
       // pobranie listy baz danych
       String[] databases = app.listDatabases();
       int i = 0;
       while (i < databases.length) {
          System.out.println(databases[i] + ":");

          // pobranie wszystkich zbiorów danej bazy
          Database database = app.getDatabase(databases[i]);
          String [] collections = database.listCollections();
          int j = 0;
          while (j < collections.length) {
             System.out.println("\t" + collections[j]);
             j++;
          }

          i++;
       }
    }

    /**
       * wstawienie do kolekcji nowego dokumentu
       */
    public static String insertDocument(Application app) throws Exception {
       // dokument XML musi być poprawny składniowo
       String document = "<?xml version=\"1.0\"?>\n <test>Hello</test>";

       // instalacja domyślna włącza bazę danych test ze zbiorem ocs
      Collection col = app.getDatabase("test").getCollection("ocs");

       // CORBA wymaga, aby dokument był EncodedBuffer
       EncodedBuffer buf = new EncodedBuffer("", document.getBytes());
       String oid = col.insertDocument(buf);
       System.out.println("Document OID: " + oid);
       return oid;
    }

    /**
       * pobranie i wydruk zapisanego dokumentu
       */
    public static void retrieveDocument(Application app, String oid)
          throws Exception {
       // instalacja domyślna włącza bazę danych test ze zbiorem ocs
       Collection col = app.getDatabase("test").getCollection("ocs");
       System.out.println(oid);
       EncodedBuffer buffer = col.getDocument(oid);

       // dane XML są zapisywane w buffer.buf
       System.out.println(new String(buffer.buf));
    }
}
```

Lista dostępnych produktów

Omawianie technik łączenia baz danych i XML zakończymy przedstawieniem listy dostępnych na rynku produktów.

Tabele 16.1 i 16.2 stanowią krótki indeks, pozwalający zorientować się, jakie produkty są dostępne na rynku.

Tabela 16.1. *Oprogramowanie pośrednie*

4ODS	FourThought	*http://opentechnology.org/4Suite/*
ASP2XML	Stonebroom	*http://www.stonebroom.com/asp2xml.htm*
Beanstalk	Transparency	*http://www.transparency.com*
Castor	exolab.org	*http://castor.exolab.org/index.html*
DatabaseDom	IBM	*http://www.alphaworks.ibm.com/tech/databasedom*
DataCraft	IBM	*http://www.alphaworks.ibm.com/formula/datacraft*
DB2XML	Volker Turau	*http://www.informatik.fh-wiesbaden.de/~turau*
DBIx::XML_RDB	Matt Sergeant	*http://theory.uwinnipeg.ca/CPAN/*
InterAccess	XML Software Corporation	*http://www.xmlsoft.com.au/iaccess.html*
Net.Data	IBM	*http://www-4.ibm.com/software/data/net.data/*
ODBC2XML	Intelligent Systems Research	*http://members.xoom.com/gvaughan/odbc2xml.htm*
XML-DB Link	Rogue Wave Software	*http://www.roguewave.com/*
XML-DBMS	Ronald Bourret	*http://www.rpbourret.com/xmldbms/index.htm*
XML Junction	Data Junction, Inc.	*http://www.xmljunction.net*
XLE	IBM	*http://www.alphaworks.ibm.com/tech/xle*

Tabela 16.2. *Serwery XML i CMS*

Astoria	Chrystal Software	*http://www.chrystal.com/products/astoria/astoria.htm*
BladeRunner	Interleaf	*http://www.xmlecontent.com/products/brintro.htm*
Bluestone XML-Server	Bluestone	*http://www.bluestone.com/*
Cocoon	Apache.org	*http://xml.apache.org/cocoon/sql.html*
DataChannel Server	DataChannel	*http://www.datachannel.com/*
Documentum	Documentum, Inc.	*http://www.documentum.com/*
Dynabase	Inso	*http://www.ebt.com/dynabase/*
Epic	Arbortext	*http://www.arbortext.com/Products/Epic/epic.html*
Excelon	eXcelon Corp.	*http://www.exceloncorp.com/products/excelon.html*
Frontier	UserLand Software	*http://frontier.userland.com*
GroveMinder	TechnoTeacher	*http://www.techno.com/*

Tabela 16.2. *Serwery XML i CMS (ciąg dalszy)*

Hynet Directive	Hynet Technologies	*http://www.hynet.com/Products/main.html*
Information	Interleaf Manager	*http://www.xmlecontent.com/products/im.htm*
Lasso	Blue World Communications	*http://www.blueworld.com/blueworld/products/*
LivePage Enterprise	Janna Systems	*http://www.janna.com*
POET Suite	POET	*http://www.poet.com/products/cms/cms.html*
Rhythmyx	Percussion Software	*http://www.percussion.com/*
SigmaLink	STEP	*http://www.step.de/sigmalink.htm*
SIM	Progressive Information Technologies	*http://www.simdb.com*
Tamino	Software AG	*http://www.softwareag.com/tamino/*
Target 2000	Magnus	*http://www.target2000.com/main.html*
WSDOM XML-Portal	Radian Systems	*http://www.radsys.com/products/portal.htm*
XML Portal Server (XPS)	Sequoia Software Corp.	*http://www.sequoiasw.com/xps/index.asp*
XMLBase	eidon	*http://www.eidon-products.com/sgmlxmlbase.htm*

Podsumowanie

W tym rozdziale omówiliśmy różne sposoby integrowania XML i baz danych; podzieliliśmy je według poziomu powiązania oprogramowania z bazą danych i danymi XML.

Omówiliśmy część teoretyczną i praktyczną wielu rozwiązań, od specjalizowanych klas, obsługujących konkretne słowniki XML, przez API SAX do baz danych i oprogramowanie pośrednie, po serwery XML.

Omówione w tym rozdziale pomysły i klasy będą używane w następnych rozdziałach jako elementy bardziej złożonych aplikacji.