helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

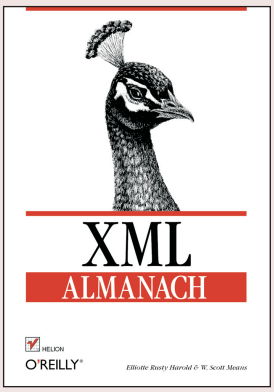
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

XML. Almanach



Autorzy: Elliotte Rusty Harold, W.Scott Means
Tłumaczenie: Jacek Mozdyniewicz
ISBN: 83-7197-594-5
Tytuł oryginału: XML in a Nutshell
Format: B5, stron: 518

Książka skupia się przede wszystkim na samym XML i przedstawia podstawowe reguły, do których muszą stosować się dokumenty i autorzy XML (np. projektant, który wykorzystuje SMIL w celu dodania animacji na stronach WWW lub programista C++, który korzysta z SOAP w celu szeregowania obiektów w zdalnej bazie danych). Książka przedstawia również specjalizowane technologie wspierające, które zostały usytuowane na szczycie XML i są wykorzystywane w wielu aplikacjach napisanych w tym języku. Technologie te obejmują:

- XLinks – Składnia, której podstawą są atrybuty. Służy ona do tworzenia hiperłączy pomiędzy dokumentami XML a innymi dokumentami. Umożliwia też tworzenie prostych, jednokierunkowych łączy, znanych z HTML, łączy wielokierunkowych pomiędzy wieloma dokumentami oraz łączy pomiędzy dokumentami, do których użytkownik nie posiada uprawnień zapisu.
- XSLT – Aplikacja XML, która opisuje transformacje jednego dokumentu na inny, wykonane za pomocą tych samych bądź odmiennych słowników.
- XPointers – Składnia służąca do identyfikacji poszczególnych części składowych dokumentu XML, do których występują odwołania poprzez identyfikatory URI. Często wykorzystywana wraz z XLink.
- XPath – Składnia, nie będąca XML, wykorzystywana przez XPointers i XSLT do identyfikowania określonych części składowych dokumentów XML. XPath może, na przykład, zlokalizować trzeci element adres w dokumencie albo wszystkie elementy z atrybutem email, którego wartością jest elharo@metalab.unc.edu.
- Namespaces (Przestrzenie nazw) – Służy do odróżniania od siebie elementów i atrybutów noszących takie same nazwy, choć pochodzących z różnych słowników XML. Na przykład, tytuł książki i tytuł strony WWW na stronie WWW o książkach.
- SAX – Simple API for XML, sterowany zdarzeniami interfejs programowania aplikacji Java, implementowany przez wiele analizatorów składni.
- DOM – Document Object Model, zorientowany na struktury drzewiaste interfejs programowania aplikacji, który traktuje dokument XML jako zbiór zagnieżdżonych obiektów o różnych właściwościach.

Wszystkie te technologie, niezależnie od tego, czy zostały zdefiniowane w XML (XLinks, XSLT i Namespaces), czy też za pomocą innej składni (XPointers, XPath, SAX i DOM), są wykorzystywane w wielu różnych aplikacjach XML.

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

Spis treści

Przedmowa *9*

CZĘŚĆ I Zadania XML-a *15*

Rozdział 1. *Ogólne wiadomości na temat XML-a* *17*

Co oferuje XML 17
Możliwość przenoszenia danych 20
Jak działa XML 20
Ewolucja XML-a 22

Rozdział 2. *Podstawy XML* *27*

Dokumenty i pliki XML 27
Elementy, znaczniki i dane znakowe 28
Atrybuty 31
Nazwy XML 33
Odwołania do encji 34
Sekcje CDATA 35
Komentarze 36
Instrukcje przetwarzania 36
Deklaracja XML 38
Sprawdzanie, czy konstrukcja dokumentów jest właściwa 39

Rozdział 3. *Definicje typu dokumentu* *43*

Kontrola prawidłowości 43
Deklaracje elementów 50
Deklaracje atrybutów 56

Deklaracje encji ogólnych ... 63
Zewnętrzne analizowane encje ogólne ... 65
Zewnętrzne encje nieanalizowane i zapisy ... 66
Encje parametryczne ... 68
Zawieranie warunkowe ... 70
Dwa przykłady DTD ... 71
Wyszukiwanie standardowych definicji DTD ... 73

Rozdział 4. *Przestrzenie nazw* ... *75*
Dlaczego przestrzenie nazw są konieczne ... 75
Składnia przestrzeni nazw ... 78
W jaki sposób analizatory składni obsługują przestrzenie nazw ... 83
Przestrzenie nazw i definicje DTD ... 84

Rozdział 5. *Środowisko międzynarodowe* ... *87*
Deklaracja kodowania ... 88
Deklaracje tekstu ... 88
Zestawy znaków definiowane w XML ... 89
Unicode ... 90
Zestawy znaków ISO ... 92
Zestawy znaków zależne od platformy ... 94
Konwertowanie zestawów znaków ... 95
Domyślny zestaw znaków dla dokumentów XML ... 96
Odwołania do znaków ... 97
xml:lang ... 99

CZĘŚĆ II Dokumenty ukierunkowane narracyjnie ... *103*

Rozdział 6. *XML jako format dokumentów* ... *105*
Dziedzictwo SGML ... 105
Struktury dokumentów narracyjnych ... 106
TEI ... 108
DocBook ... 111
Trwałość dokumentów ... 114
Transformacja i prezentacja ... 115

Rozdział 7. *XML w sieci WWW* *119*
XHTML 120
Bezpośrednie wyświetlanie XML w przeglądarkach 126
Tworzenie złożonych dokumentów przy użyciu Modularnego XHTML 131
Ulepszone metody wyszukiwania w sieci WWW 145

Rozdział 8. *XSL Transformations* *149*
Przykład dokumentu wejściowego 149
xsl:stylesheet i xsl:transform 150
Procesory arkuszy stylów 152
Szablony 153
Wyliczanie wartości elementu za pomocą xsl:value-of 154
Stosowanie szablonów przy użyciu xsl:apply-templates 155
Wbudowane reguły szablonów 158
Tryby 161
Szablony wartości atrybutów 163
XSLT i przestrzenie nazw 164
Inne elementy XSLT 165

Rozdział 9. *XPath* *167*
Drzewiasta struktura dokumentu XML 167
Ścieżki położeń 169
Złożone ścieżki położenia 174
Predykaty 176
Nieskrócone ścieżki położeń 177
Wyrażenia ogólne XPath 179
Funkcje XPath 182

Rozdział 10. *XLinks* *189*
Proste łącza 190
Jak zachowują się łącza 191
Semantyka łącza 194
Łącza rozszerzone 194
Baza łączy 201
Definicje DTD dla XLink 202

Rozdział 11. *Wyrażenia XPointer* *203*
Wyrażenia XPointer w identyfikatorach URL 203
Wyrażenia XPointer w łączach 205
Nazwy pierwotne 206
Sekwencje elementów potomnych 207
Punkty 207
Zakresy 210

Rozdział 12. *Kaskadowe arkusze stylów (CSS)* *213*
Trzy poziomy CSS 215
Składnia CSS 215
Kojarzenie arkuszy stylów z dokumentami XML 217
Selektory 218
Właściwości wyświetlania 222
Piksele, punkty, cycera i inne jednostki długości 223
Właściwości czcionki 225
Właściwości tekstu 226
Kolory 227

Rozdział 13. *XSL Formatting Objects (XSL-FO)* *229*
XSL-Formatting Objects 231
Struktura dokumentu XSL-FO 232
Strony wzorcowe 233
Właściwości XSL-FO 239
CSS czy XSL-FO? 243

CZĘŚĆ III Dokumenty zorientowane na dane *245*

Rozdział 14. *XML jako format danych* *247*
Aplikacje programowania XML 247
Opis danych 249
Współpraca z programistami 251

Rozdział 15. *Modele programowania* *253*
Modele sterowane zdarzeniami kontra modele sterowane obiektami 253
Obsługa języków programowania 253
Niestandardowe rozszerzenia 255

Przekształcenia ... 256
Polecenia przetwarzania ... 256
Łącza i odwołania ... 257
Zapisy ... 257
To, co dostajesz nie jest tym, co widziałeś ... 258

Rozdział 16. *Model obiektowy dokumentu (DOM)* ... *259*
Jądro DOM ... 260
Zalety i wady DOM ... 260
Analiza składni dokumentu za pomocą DOM ... 261
Interfejs Node ... 261
Specyficzne typy węzłów ... 262
Interfejs DOMImplementation ... 268
Prosta aplikacja DOM ... 268

Rozdział 17. *SAX* ... *273*
Interfejs ContentHandler ... 275
Właściwości i funkcje SAX ... 281

CZĘŚĆ IV Materiały informacyjne ... *285*

Rozdział 18. *Leksykon XML 1.0* ... *287*
Jak korzystać z leksykonu ... 287
Przykładowe dokumenty z komentarzami ... 287
Klucz do składni XML ... 291
Właściwa konstrukcja ... 291
Prawidłowość ... 295
Globalne struktury składniowe ... 303
DTD (Definicja typu dokumentu) ... 309
Treść dokumentu ... 318
Gramatyka dokumentu XML ... 319

Rozdział 19. *Leksykon XPath* ... *323*
Model danych XPath ... 323
Typ danych ... 324
Ścieżki położeń ... 325
Predykaty ... 329
Funkcje XPath ... 329

Rozdział 20. ***Leksykon XSLT*** ***341***
Przestrzeń nazw XSLT ... 341
Elementy XSLT ... 341
Funkcje XSLT ... 369

Rozdział 21. ***Leksykon DOM*** ***375***
Hierarchia obiektów ... 376
Leksykon obiektów ... 377

Rozdział 22. ***Leksykon SAX*** ***437***
Pakiet org.xml.sax ... 437
Pakiet org.xml.sax.helpers ... 444
Właściwości i funkcje SAX ... 450
Pakiet org.xml.sax.ext ... 452

Rozdział 23. ***Zestawy znaków*** ***455***
Tablice znaków ... 457
Zestawy encji HTML4 ... 462
Inne bloki Unicode ... 471

Skorowidz ***499***

6

XML jako format dokumentów

Kiedy powstał XML, był on pierwszym i najważniejszym formatem dokumentów. Służył do zapisywania stron WWW, książek, artykułów naukowych, poematów, krótkich opowiadań, informatorów, samouczków, podręczników, pism urzędowych, kontraktów, instrukcji i innych dokumentów. Wykorzystanie go jako składni dla danych komputerowych w aplikacjach takich, jak przetwarzanie rozkazów, szeregowanie obiektów, wymiana i tworzenie kopii bezpieczeństwa baz danych i elektroniczna wymiana danych, było przeważnie rezultatem szczęśliwego przypadku.

Większość programistów przygotowana jest do pracy ze sztywnymi strukturami, spotykanymi w aplikacjach zorientowanych na dane, nie jest jednak gotowa na zetknięcie się ze środowiskiem artykułu czy opowiadania; większość pisarzy jest przyzwyczajona do swobodnego formatu książki, opowiadania czy artykułu. XML odpowiada jednak potrzebom obu tych społeczności. Ten rozdział opisuje (na przykładach) struktury spotykane w dokumentach, które przeznaczone są do czytania przez ludzi, nie przez komputery. W kolejnych rozdziałach przyjrzymy się stronom WWW, a następnie technikom adresowania takim, jak XSLT, XLink i arkusze stylów, które pierwotnie były stosowane w dokumentach czytanych przez ludzi. Kiedy już zapoznamy się z tymi technikami, przetestujemy XML jako format dla stosunkowo nietrwałych danych, przeznaczonych do odczytu przez komputery (czyli danych, które nie są wykorzystywane jako półtrwałe dokumenty odczytywane przez ludzi).

Dziedzictwo SGML

XML jest uproszczoną formą SGML-a (Standardized General Markup Language — standardowy uogólniony język znaczników). Język, który ostatecznie stał się SGML-em, został wynaleziony przez Charlesa F. Goldfarba, Eda Moshera i Raya Lorie z IBM w latach siedemdziesiątych. Następnie pracowało nad nim kilkaset osób na całym świecie, aż do czasu ostatecznego przyjęcia go w 1986 roku jako standard ISO 8879. Celem twórców SGML było rozwiązywanie problemów, które rozwiązuje XML, w taki sam sposób, w jaki czyni to XML. SGML to język oznaczeń semantycznych i strukturalnych dla dokumentów tekstowych. Posiada on ogromne możliwości i odniósł już kilka sukcesów w kręgach wojskowych i rządowych Stanów Zjednoczonych, sektorze lotni-

czo-kosmicznym oraz innych dziedzinach, w których potrzebne jest efektywne zarządzanie dokumentami technicznymi o objętości dziesiątek tysięcy stron.

Największym sukcesem SGML było stworzenie własnej aplikacji, HTML. Jednakże HTML, ponieważ jest tylko jedną z aplikacji, nie posiada takich możliwości, jak SGML. SGML został również wykorzystany do zdefiniowania innych formatów dokumentów, m.in. DocBook i Text Encoding Initiative (TEI), które zostaną tutaj pokrótce omówione.

SGML jest bardzo skomplikowany. Oficjalna specyfikacja zawiera ponad 150 stron i obejmuje wiele przypadków szczególnych i mało prawdopodobnych sytuacji. Jest tak skomplikowana, że niewiele programów komputerowych posiada jej pełną implementację, a programy, które implementują lub opierają się na różnych fragmentach specyfikacji SGML, są często niezgodne. Szczególne funkcje, które w jednym programie zostały uznane za zasadnicze, są często postrzegane jako niepotrzebne dodatki i pomijane w innym programie. Mimo to, doświadczenia z SGML nauczyły programistów prawidłowego projektowania, implementacji i użytkowania języków oznaczeń w różnorakich dokumentach. Wiele z tych doświadczeń obejmuje również XML.

Oczywiście, dokumenty XML znajdują zastosowanie nie tylko w sieci WWW; XML może z łatwością sprostać wymaganiom różnych mediów, takich jak książki, magazyny, dzienniki, gazety i broszury. XML jest szczególnie użyteczny, gdy istnieje potrzeba wydania dokumentu we wszystkich mediach jednocześnie. Dzięki zastosowaniu dla tego samego dokumentu źródłowego różnych arkuszy stylów, można utworzyć strony WWW, notatki prelegenta i gotowe do druku matryce.

Struktury dokumentów narracyjnych

Wszystkie dokumenty XML mają strukturę drzewa, które jest jednak strukturą danych ogólnego przeznaczenia. Każdy, kto uczył się informatyki, zetknął się z drzewami binarnymi, drzewami czerwono-czarnymi, drzewami zrównoważonymi, drzewami typu B czy drzewami uporządkowanymi. Jednakże XML rzadko pasuje do którejś z tych struktur. Dokumenty XML stanowią najbardziej ogólną odmianę drzewa, w której nie ma szczególnych ograniczeń dotyczących uporządkowania węzłów, sposobu ich połączenia, czy zasad łączenia różnych typów węzłów. Narracyjne dokumenty XML mają strukturę mniej identyfikowalną niż dokumenty XML zorientowane na dane.

Jak zatem wyglądają dokumenty XML zorientowane narracyjnie? Oczywiście, mają one element bazowy, który posiadają wszystkie dokumenty XML. Ogólnie rzecz biorąc, ten element bazowy reprezentuje dokument. Na przykład, jeżeli dokument jest książką, to elementem bazowym jest `book` (książka). Jeżeli dokument jest artykułem, to elementem bazowym jest `article` (artykuł).

Duże dokumenty są podzielone na części, w przypadku książek na rozdziały, w przypadku artykułów na sekcje, a w dokumentach prawniczych na paragrafy. Większość dokumentu znajduje się w tych podstawowych częściach. W niektórych przypadkach istnieje kilka różnych rodzajów części, na przykład, jedna dla spisu treści, inna dla indeksu i po jednej na każdy rozdział książki.

Element bazowy z reguły zawiera również elementy dostarczające metainformacji o dokumencie, takich jak tytuł roboczy, autor dokumentu, czy data utworzenia dokumentu i ostatniej jego modyfikacji.

Umieszczenie metainformacji w jednym elemencie potomnym elementu bazowego, a głównej treści w innym, stanowi regułę. W taki właśnie sposób pisane są dokumenty HTML. Elementem bazowym jest `html`. Metainformacje zawarte są w elemencie `head` (nagłówek), a główna treść znajduje się w elemencie `body` (treść dokumentu). Tej regule podporządkowane są dokumenty TEI, a także DocBook.

Sekcje dokumentów mogą dzielić się na podsekcje, które mogą podlegać dalszemu podziałowi. Ilość poziomów podsekcji zależy głównie od wielkości dokumentu. Encyklopedia ma bardzo wiele poziomów podsekcji, podczas gdy broszura czy ulotka reklamowa nie ma ich prawie w ogóle. Każda sekcja lub podsekcja z reguły ma tytuł. Może również posiadać atrybuty lub elementy wskazujące na metainformacje o sekcji, takie jak autor lub data ostatniej modyfikacji.

Do tej pory unikano zawartości mieszanej. Elementy zwykle zawierają elementy potomne i pomijalne znaki niewidoczne. Jednakże w niektórych przypadkach konieczne jest wstawienie aktualnego tekstu dokumentu — słów czytelnych dla ludzi. W większości języków zachodnich tekst byłby podzielony na akapity i inne elementy na poziomie bloków, takie jak nagłówki, rysunki, paski boczne (sidebar) i przypisy. Definicje DTD ogólnego dokumentu, na przykład dokumentu DocBook, nie będą w stanie powiedzieć o nich nic więcej.

Akapity i inne elementy blokowe z reguły zawierają słowa zapisane w wierszu, czyli tekst. Część tego tekstu może być oznaczona jako elementy wierszowe. Na przykład, można wskazać, że konkretny ciąg znaków w tekście elementu blokowego jest datą lub osobą albo że jest po prostu ważny. Jednakże większość tekstu nie będzie oznaczona w taki sposób.

Jednym z obszarów, w jakim różne aplikacje XML działają rozbieżnie, jest problem, czy element blokowy może zawierać w sobie inne elementy blokowe. Na przykład, czy akapit może zawierać listę albo czy lista może zawierać akapit. Przypuszczalnie praca z bardziej strukturalnymi dokumentami, w których bloki nie mogą zawierać innych bloków (szczególnie bloków tego samego typu), jest łatwiejsza. Jednakże czasami istnieją powody, dla których bloki mogą zawierać w sobie inne bloki. Na przykład, długie wykazy pozycji lub cytatów mogą zawierać kilka akapitów.

Cała ta struktura, od elementu bazowego do najgłębiej zagnieżdżonego elementu wierszowego, wydaje się być strukturą liniową; to znaczy, że czytelnik będzie czytać wyrazy w kolejności prawie takiej samej, w jakiej pojawiają się one w dokumencie. Gdyby całe oznaczenie zostało nagle usunięte i czytelnikowi zostałby pozostawiony pierwotny tekst, rezultat byłby nadal czytelny. Oznaczenie można wykorzystywać do indeksowania lub formatowania dokumentów, ale nie stanowi ono zasadniczej części ich treści.

Inną ważną cechą narracyjnych dokumentów XML tego rodzaju jest fakt, że nie tylko składają się one z wyrazów zapisanych w wierszach, ale są zbudowane z *wyrazów*. Zawierają tekst przeznaczony do czytania przez ludzi. Nie zawierają liczb, dat czy wartości pieniężnych (poza sytuacjami, kiedy występują one jako część normalnego strumienia narracyjnego). Zawartość `#PCDATA` jest najniższym elementem drzewa, zazwyczaj jednego typu: ciągu znaków. Jeżeli jakiś element należy do typu innego niż ciąg znaków, zwykle stanowi metainformację o dokumencie (numer rysunku, data ostatniej modyfikacji itp.), a nie treść samego dokumentu.

Dużo czasu zajęłoby nam wyjaśnianie, dlaczego definicje DTD nie zapewniają silnego, a w zasadzie nie zapewniają żadnego, wspomagania dla wprowadzania danych. Nie wymagały go dokumenty, dla których został zaprojektowany SGML. Dokumenty XML wykorzystywane w zadaniach, dla których nie został zaprojektowany SGML, takich jak śledzenie spisu inwentarza czy

spis ludności, wymagają wprowadzania danych; dlatego różne osoby i organizacje wynalazły tak liczne języki schematyczne. Jednakże schematy nie wzbogacają zbytnio definicji DTD dokumentów narracyjnych tego rodzaju.

Nie wszystkie dokumenty XML przypominają dokumenty opisane tutaj; do opisu tego nie pasują nawet wszystkie dokumenty zorientowane narracyjnie. Jednak zaskakująco duża liczba aplikacji zorientowanych narracyjnie odpowiada wzorowi podstawowemu, prawdopodobnie trochę do niego dodając lub od niego ujmując. Powodem, dla którego dokumenty narracyjne przyjmują tę podstawową strukturę, jest jej użyteczność. Gdybyśmy mieli definiować własne definicje DTD dla ogólnych dokumentów zorientowanych narracyjnie, miałyby one wiele wspólnego z tą strukturą. Gdybyśmy definiowali własne definicje DTD dla bardziej specjalizowanych dokumentów zorientowanych narracyjnie, wtedy nazwy elementów mogłyby się różnić, w zależności od dziedziny. Pisząc na przykład podręcznik Młodego Harcerza, jedną z sekcji moglibyśmy nazwać `Zdobycie-Sprawności`. Jednakże podstawowa hierarchia dokumentu, metainformacje, sekcje i podsekcje, elementy blokowe i tekst zaopatrzony w oznaczenie pozostałyby najprawdopodobniej bez zmian.

TEI

Text Encoding Initiative (TEI), (*http://www.tei-c.org/*), jest aplikacją SGML przeznaczoną do oznaczania tekstów literatury klasycznej, takich jak „Eneida" Wergiliusza czy dzieła zebrane Tomasza Jeffersona. TEI jest świetnym przykładem definicji DTD zorientowanej narracyjnie. Ponieważ została ona przeznaczona do analiz naukowych, a nie do swobodnego czytania czy działalności wydawniczej, zawiera elementy zarówno typowych struktur dokumentów (rozdział, scena, zwrotka itp.), jak i typografii, struktury gramatycznej, położeń ilustracji na stronie i tak dalej. Te struktury i elementy nie są dla większości czytelników istotne, są jednak kluczowe dla naukowców, dla których TEI została stworzona. W przypadku wielu prac akademickich, jeden rękopis „Eneidy" nie jest dokładnie taki sam jak następny. Istotne mogą być błędy przekładu i poprawki wnoszone w średniowieczu przez różnych mnichów.

TEI jest aplikacją SGML. Korzysta z kilku funkcji SGML, jakich nie znajdziemy w XML, m.in. łącznika `&` i znacznika minimalizacji. Jednakże XML jest zwiastunem przyszłości. Tak jak większość ewoluujących aplikacji SGML, również TEI przekształca się w kierunku XML. Dostępna jest już uproszczona wersja DTD TEI dla autorów, którzy preferują pracę w czystym XML-u. Nie jest ona tak kompletna jak wersja SGML, ale wiele jej praktycznych zastosowań jest bardzo do tej wersji zbliżone.

Przykład 6.1 pokazuje dość prosty dokument TEI Lite, który wykorzystuje XML-ową wersję definicji DTD TEI. Jego zawartość pochodzi z książki, którą właśnie czytamy. Mimo że rękopis zakodowany za pomocą pełnej wersji byłby o wiele dłuższy, ten prosty przykład demonstruje podstawowe cechy większości dokumentów TEI, które reprezentują książki. (TEI, poza prozą, można również zastosować w przypadku gier, poematów, mszałów i dowolnej formy literatury pisanej).

Przykład 6.1. Dokument TEI

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE TEI.2 SYSTEM "xteilite.dtd">
<TEI.2>

  <teiHeader>
    <fileDesc>
```

```
    <titleStmt>
      <title>XML. Almanach</title>
      <author>Harold, Elliotte Rusty</author>
      <author>Means, W. Scott</author>
    </titleStmt>
    <publicationStmt><p></p></publicationStmt>
    <sourceDesc><p>Wczesny projekt rękopisu</p></sourceDesc>
  </fileDesc>
</teiHeader>

<text id="HarXMLi">

  <front>
    <div type='toc'>
      <head>Spis treści</head>
      <list>
        <item>Prezentacja XML</item>
        <item>XML jako format dokumentów</item>
        <item>XML jako "lepszy" HTML</item>
      </list>
    </div>

  </front>

  <body>

    <div1 type="chapter">
      <head>Prezentacja XML</head>
      <p></p>
    </div1>

    <div1 type="chapter">
      <head>XML jako format dokumentów</head>
      <p>
        Kiedy powstał XML, był on pierwszym i najważniejszym formatem
        dokumentów. Służył do zapisywania stron WWW, książek,
        artykułów naukowych, poematów, krótkich opowiadań,
        informatorów, samouczków, podręczników, pism urzędowych,
        kontraktów, instrukcji i innych dokumentów, czytanych
        przez ludzi. Wykorzystanie go jako składni dla danych
        komputerowych w takich aplikacjach, jak przetwarzanie
        rozkazów, szeregowanie obiektów, wymiana i tworzenie kopii
        bezpieczeństwa baz danych i elektroniczna wymiana danych, było
        przeważnie rezultatem szczęśliwego przypadku.
     </p>

     <div2 type="section">
       <head>Spuścizna SGML</head>
       <p></p>
     </div2>

     <div2 type="section">
       <head>TEI</head>
       <p></p>
     </div2>

     <div2 type="section">
       <head>DocBook</head>
       <p>
         DocBook (http://www.oasis-open.org/docbook/) jest
         aplikacją SGML przeznaczoną wyłącznie dla
         nowych dokumentów. Typowe jej użycie ma miejsce
```

```
            w przypadku dokumentacji komputerowej. Kilka książek
            wydawnictwa O'Reilly zostało napisanych przy użyciu
            DocBook (m.in. książka Normana Walsha
            i Leonarda Muellnersa „DocBook: Definitive Guide"). Zostało
            w nim napisanych także wiele tekstów projektu dokumentacji Linuxa
            (http://www.linuxdoc.org/) w DocBook.
          </p>
        </div2>

      </div1>

      <div1 type="chapter">
        <head>XML jako "lepszy" HTML</head>
        <p></p>
      </div1>
    </body>

    <back>
      <div1 type="index">
        <list>
          <head>Skorowidz</head>
          <item>SGML, 46, 145-148</item>
          <item>DocBook, 147</item>
          <item>TEI, 149</item>
          <item>Text Encoding Initiative, Patrz TEI</item>
        </list>
      </div1>
    </back>

  </text>
</TEI.2>
```

Elementem bazowym wszystkich dokumentów TEI jest `TEI.2`. Jest on zawsze podzielony na dwie części: nagłówek, reprezentowany przez element `teiHeader`, i główną treść dokumentu, reprezentowaną przez element `text`. Nagłówek zawiera informacje o źródle dokumentu, na przykład, z którego średniowiecznego rękopisu został przepisany tekst, informację o kodowaniu dokumentu lub słowa kluczowe opisujące dokument.
Element `text` dzieli się na trzy części:

Początek w elemencie front

Początek zawiera przedmowę, spis treści, stronę z dedykacją i okładkę. Każdy z tych elementów reprezentowany jest przez element `div` z atrybutem `type`, którego wartość identyfikuje daną część jako spis treści, przedmowę, stronę tytułową itd. Każda część zawiera inne elementy, które rozmieszczają jej zawartość.

Treść w elemencie body

Treść zawiera poszczególne rozdziały, z jakich składa się dokument. Każdy z tych elementów reprezentowany jest przez element `div1` z atrybutem `type`, który identyfikuje poszczególne części jako tomy, książki, części, rozdziały, poematy, akty itd. Każdy element `div1` ma element potomny `header`, który nadaje tytuł tomowi, książce, części lub rozdziałowi.

Zakończenie w elemencie back

Zakończenie zawiera indeks i słowniczek.

Ten podział może wyglądać dalej następująco: `div1` może zawierać elementy `div2`, `div 2` może zawierać elementy `div3`, `div3` może zawierać elementy `div4` i tak dalej, aż do `div7`. Jednakże w każdej pracy istnieje pewna najmniejsza jej część. Te najmniejsze części, w przypadku prozy zawierają akapity i są reprezentowane przez elementy `p`, a w przypadku poezji zawierają zwrotki, reprezentowane przez elementy `lg`. Zwrotki dzielą się dalej na poszczególne wiersze, reprezentowane przez elementy `l`.

Zarówno wiersze, jak i akapity posiadają zawartość mieszaną. Oznacza to, że zawierają zwykły test. Jednakże części tego tekstu mogą być dodatkowo oznaczone przez elementy wskazujące, że konkretne słowa lub znaki są nazwiskami osób (`name`), poprawkami (`corr`), nieczytelne (`unclear`), błędnie napisane (`sic`) itd.

Ta struktura dokładnie odzwierciedla strukturę rzeczywistych dokumentów kodowanych w TEI i sprawdza się w przypadku większości aplikacji XML zorientowanych narracyjnie (aplikacji, które muszą obsługiwać dość ogólne dokumenty). TEI jest bardzo reprezentatywnym przykładem struktury typowego dokumentu XML.

DocBook

DocBook (*http://www.oasis-open.org/docbook/*) jest aplikacją SGML stworzoną wyłącznie dla nowych dokumentów. Typowe jej użycie ma miejsce w przypadku dokumentacji komputerowej. Kilka książek wydawnictwa O'Reilly zostało napisanych przy użyciu DocBook (m.in. książka Normana Walsha i Leonarda Muellnersa „DocBook: Definitive Guide"). Zostało w nim napisanych także wiele tekstów projektu dokumentacji Linuxa (*http://www.linuxdoc.org/*).

Obecna wersja DocBook 4.1.2 jest dostępna zarówno jako aplikacja SGML, jak i aplikacja XML. Wersja XML nie jest dokładnie taka sama jak wersja SGML, ale wiele ich zastosowań praktycznych jest do siebie zbliżone. Zespół pracujący nad rozwojem DocBook zapowiedział, że w wersji 5.0 nastąpi przejście na pojedynczą definicję DTD, całkowicie zgodną zarówno z SGML, jak i z XML. Przykład 6.2 ilustruje prosty dokument DocBook XML, bazujący na czytanej właśnie książce. Nie musimy przypominać, że pełna wersja tego dokumentu byłaby dużo dłuższa.

Przykład 6.2: Dokument DocBook

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE book PUBLIC "-//Norman Walsh//DTD DocBk XML V3.1.7//EN"
                      "docbook/docbookx.dtd">
<book>
  <title>XML. Almanach</title>

  <bookinfo>
    <author>
      <firstname>Elliotte Rusty</firstname>
      <surname>Harold</surname>
    </author>
    <author>
      <firstname>W. Scott</firstname>
      <surname>Means</surname>
    </author>
  </bookinfo>
```

```
<toc>
  <tocchap><tocentry>Prezentacja XML</tocentry></tocchap>
  <tocchap><tocentry>XML jako format dokumentów</tocentry></tocchap>
  <tocchap><tocentry>XML jako "lepszy" HTML</tocentry></tocchap>
</toc>

<chapter>
  <title>Prezentacja XML</title>
  <para></para>
</chapter>

<chapter>
  <title>XML jako format dokumentów</title>

 <para>
   Kiedy powstałXML, był on pierwszym i najważniejszym formatem dokumentów.
   Służył on do zapisywania stron WWW, książek,
   artykułów naukowych, poematów, krótkich opowiadań,
   informatorów, samouczków, podręczników, pism urzędowych,
   kontraktów, instrukcji i innych dokumentów, czytanych
   przez ludzi. Wykorzystanie go jako składni dla danych
   komputerowych w takich aplikacjach, jak przetwarzanie
   rozkazów, szeregowanie obiektów, wymiana i tworzenie kopii
   bezpieczeństwa baz danych i elektroniczna wymiana danych,
   było przeważnie rezultatem szczęśliwego przypadku.
 </para>

 <sect1>
   <title>Spuścizna SGML</title>
   <para></para>
 </sect1>

 <sect1>
   <title>TEI</title>
   <para></para>
 </sect1>

 <sect1>
   <title>DocBook</title>
   <para>
     <ulink url="http://www.oasis-open.org/docbook/">DocBook</ulink>
     jest aplikacją SGML przeznaczoną wyłącznie dla
     nowych dokumentów. Typowe jej użycie ma miejsce
     w przypadku dokumentacji komputerowej. Kilka książek
     wydawnictwa O'Reilly zostało napisanych przy użyciu
     DocBook (m.in. książka<citation>Normana Walsha
     i Leonarda Muellnersa<title>„DocBook: Definitive Guide")</title>
     </citation>. Zostało w nim napisanych także wiele tekstów <ulink
     url="http://www.linuxdoc.org/">
     projektu dokumentacji Linuxa</ulink>.
   </para>
 </sect1>

</chapter>

<chapter>
  <title>XML jako "lepszy" HTML</title>
  <para></para>
</chapter>

<index>
  <indexentry>
    <primaryie>SGML, 46, 145-148</primaryie>
```

```
      </indexentry>
      <indexentry>
        <primaryie>DocBook, 147</primaryie>
      </indexentry>
      <indexentry>
        <primaryie>TEI, 149</primaryie>
      </indexentry>
      <indexentry>
        <primaryie>Text Encoding Initiative</primaryie>
        <seeie>TEI</seeie>
      </indexentry>
    </index>

</book>
```

DocBook oferuje autorom tekstów technicznych wiele udogodnień. Po pierwsze, jest otwarty, nie zastrzeżony i można go tworzyć w dowolnym edytorze tekstu. Pisanie dokumentacji z otwartym dostępem do kodu źródłowego dla oprogramowania z takim samym dostępem (za pomocą zamkniętych i zastrzeżonych narzędzi, np. Microsoft Word) można by uznać za niezbyt mądre. Dokumenty napisane w DocBook nie są związane z żadną platformą, producentem ani aplikacją. Można je przenosić pomiędzy wszystkimi środowiskami, jakie tylko można sobie wyobrazić.

Po drugie, prostota DocBook umożliwia jego edycję nie tylko w teorii, ale również w praktyce (można tego dokonać za pomocą najprostszych edytorów tekstowych). Jeżeli potrzebna jest pomoc, pod adresem *http://www.nwalsh.com/emacs/docbookide/index.html* dostępne są bezpłatne narzędzia, m.in. główny tryb emacs. Po trzecie, podobnie jak wiele innych aplikacji XML, również DocBook jest modularny. Można wykorzystać wybrane jego kawałki i zignorować resztę. Jeżeli potrzebne są tabele, dostępny jest moduł tabel o rozległych możliwościach. Jeżeli tabele są nie potrzebne, nie trzeba tego modułu używać. Inne moduły zawierają różne zestawy encji i równań.

DocBook jest formatem autorskim, a nie formatem gotowej prezentacji. Zanim ktoś przeczyta dokument tego typu, musi on być skonwertowany na jeden z kilku następujących formatów:

- HTML
- XSL Formatting Objects
- Rich Text Format (RTF)
- TEX

Na przykład, gdy potrzebna jest drukowana dokumentacja programu wysokiej jakości, należy skonwertować dokument DocBook do TEX, korzystając ze standardowych narzędzi TEX-a skonwertować plik TEX na DVI lub, bądź również, na plik PostScript, a następnie wydrukować. Jeżeli ten dokument ma być czytany tylko na komputerze, najprawdopodobniej skonwertujemy go do formatu HTML i załadujemy do przeglądarki stron WWW. Jeśli chcemy osiągnąć inne cele, należy wybrać inny format. Korzystając z DocBook, otrzymujemy wszystkie te formaty w zasadzie za darmo. Tworzenie wielu dokumentów wyjściowych w różnych formatach z pojedynczego dokumentu źródłowego DocBook jest bardzo proste. Zalety te posiada nie tylko format DocBook. Najbardziej wymyślne formaty wejściowe XML można w razie potrzeby opublikować w innych formatach w równie prosty sposób.

Trwałość dokumentów

Dokumenty XML przeznaczone do czytania przez komputery są często nietrwałe. Jeżeli utworzymy dokument Simple Object Access Protocol (SOAP), który reprezentuje żądanie skierowane do serwera Windows realizującego usługi Next Generation Windows Services (NGWS), to dokument ten jest trwały tak długo, jak klient wysyła ten dokumentu do serwera, który następnie przetwarza go na wewnętrzne struktury danych. Po zakończeniu tego procesu dokument zostanie usunięty. Prawdopodobnie nie będzie on istniał dłużej niż przez dwie minuty. Dokument jest ulotną formą komunikacji pomiędzy dwoma systemami, jego trwałość nie jest większa niż trwałość miliardów innych wiadomości, jakie komputery wymieniają w ciągu dnia (większość tych wiadomości nigdy nie będzie nawet zapisana na dysku, a tym bardziej zarchiwizowana dla potomności).

Niektóre aplikacje przechowują w formacie XML bardziej trwałe dane zorientowane komputerowo. Na przykład, XML jest rodzimym formatem plików arkusza kalkulacyjnego Gnumeric. Ten format jest jednakże rozumiany tylko przez Gnumeric i być może przez inne aplikacje GNOME. Został zaprojektowany dla specyficznych potrzeb tego jednego programu. Wymiana danych z innymi aplikacjami, m.in. z tymi, które nie zostały jeszcze wynalezione, jest sprawą drugorzędną.

Dokumenty przeznaczone dla ludzi wydają się być bardziej trwałe i mniej związane z oprogramowaniem. Jeżeli zakodujemy w XML Deklarację Niepodległości, chcemy, by ludzie mogli ją przeczytać za 2 lata, za 200 lat lub za 2000 lat. Chcemy również, by mogli ją odczytać za pomocą dowolnego, wygodnego narzędzia, również takiego, które jeszcze nie zostało wynalezione. Te wymagania oznaczają istotne zmiany zarówno dla aplikacji XML przeznaczonych do przechowywania danych, jak również dla narzędzi, których używamy do ich czytania i pisania.

Po pierwsze, format powinien być bardzo dobrze udokumentowany. Powinna istnieć dobrze skomentowana definicja DTD. Ponadto powinna istnieć znacząca liczba opracowań dokumentacyjnych. Opracowania te nie zastąpią oficjalnej dokumentacji, ale stanowią nieocenioną pomoc przy zrozumieniu danej DTD.

Należy wybierać formaty standardowe, takie jak DocBook i TEI, a nie nietypowe, jednorazowe aplikacje XML. Powinniśmy unikać zastrzeżonych definicji DTD, należących do indywidualnych osób lub firm, których przyszłość może zależeć od losów tych osób czy firm. Nawet jeżeli DTD pochodzi od niekomercyjnego konsorcjum, takiego jak OASIS czy TEI, jej licencja powinna być na tyle liberalna, aby ograniczenia wynikające z prawa autorskiego nikomu nie blokowały działania. Obecnie o opatentowanie swojej DTD stara się co najmniej jeden z jej dostawców. Takich DTD trzeba unikać. Należy trzymać się takiej definicji DTD, którą można bezpłatnie kopiować i udostępniać oraz którą można uzyskać z wielu różnych lokalizacji.

Kiedy już zdecydujemy się na standardową definicję DTD, starajmy się unikać jej modyfikacji. Jeżeli koniecznie chcemy coś zmodyfikować, zmiany należy bardzo szczegółowo dokumentować. Komentarze należy dodać zarówno do własnych definicji DTD, jak i do dokumentów, objaśniając wprowadzane przez siebie zmiany. Przy dodawaniu nowych typów elementów lub usuwaniu starych należy korzystać z wbudowanych w DTD encji parametrycznych, a nie modyfikować same pliki definicji DTD.

Format nie powinien być zbyt trudny do odtworzenia (na wypadek, gdyby zaginęła dokumentacja). Wewnątrz dokumentu należy koniecznie używać pełnych nazw elementów i atrybutów. Element `para` DocBook jest lepszy od elementu `p` z TEI. `paragraph` byłby nawet jeszcze lepszy.

Całą strukturę dokumentu powinno wyznaczać tylko oznaczenie. Nie należy pozostawiać użytkownikowi pola dla domysłów ani nie należy stosować kodowania wykorzystującego separatory takie, jak znaki niewidoczne. Oto przykład tego, czego nie należy robić w przypadku grafiki SVG:

```
<polygon style="fill: blue; stroke: green; stroke-width: 12"
  points="350,75 379,161 469,161 397,215 423,301 350,250
          277,301 303,215 231,161 321,161"/>
```

Atrybut `style` zawiera trzy odrębne, słabo związane pozycje. Zrozumienie tych elementów wymaga analizy nie XML-owego formatu CSS. Atrybut `points` jest pod tym względem jeszcze gorszy. Jest to długa lista liczb, która nie zwiera informacji i znaczenia każdej z liczb. Nie można, na przykład, określić, które z liczb są współrzędnymi *x*, a które *y*. Bardziej pożądany byłby zapis przedstawiony w następnym przykładzie:

```
<polygon fill=" blue" stroke="green" stroke-width="12">
   <point x="350" y="75"/>
   <point x="379" y="161"/>
   <point x="469" y="161"/>
   <point x="397" y="215"/>
   <point x="423" y="301"/>
   <point x="350" y="250"/>
   <point x="277" y="301"/>
   <point x="303" y="215"/>
   <point x="231" y="161"/>
   <point x="321" y="161"/>
</polygon>
```

Składnia stylu, która bazuje na atrybutach, jest obecnie dopuszczalna w przypadku SVG, jednakże w grupie roboczej W3C SVG odbyła się dość gorąca dyskusja, jakiego formatu należy używać dla współrzędnych. Grupa ostatecznie zadecydowała, naszym zdaniem błędnie, że bardziej obszerna forma nigdy nie zostanie przyjęta (z uwagi na objętość), mimo że większość członków uważała, że lepiej oddaje ona charakter XML. Twierdzimy, że w erze gwałtownie rosnącej pojemności dysków twardych i dostępności pasma sieci grupa robocza przeceniła wagę rozmiaru dokumentu. Zignorowała również łatwość, z jaką drugi format mógłby być skompresowany w celu ułatwienia transportu lub przechowywania.

Arkusze stylów są ważne. Wszyscy chcemy oddzielić styl od treści. Ostrzegano nas, by nie dołączać do swoich dokumentów XML zwykłych informacji o stylach, takich jak pochylenie tekstu i wybór czcionek. W arkuszach stylów nie należy również załączać treści. Nazwisko autora, tytuły, prawa autorskie i inne informacje, które różnią się w zależności od dokumentu, należą do dokumentu, a nie do arkusza stylów, nawet gdy zawierają metainformacje o dokumencie, a nie rzeczywistą jego zawartość.

Pamiętajmy, że nasze dokumenty powinny przetrwać następnych kilka tysięcy lat. Ulitujmy się nad historykami, którzy będą musieli rozszyfrować nasze oznaczenie za pomocą ograniczonych narzędzi i informacji.

Transformacja i prezentacja

Oznaczenie w typowym dokumencie XML opisuje z reguły jego strukturę, ale nie opisuje prezentacji dokumentu; informuje ono, jak dokument jest zorganizowany, a nie jak wygląda. Mimo że dokumenty XML są tekstem i czytelnik mógłby, gdyby chciał, czytać je w pierwotnej formie, zanim zostaną one zaprezentowane publiczności, dokumenty te są zwykle przetwarzane do innego

formatu. Kluczowym zadaniem języków znaczników (szczególnie w przypadku XML) jest sprawienie, by format wejściowy nie był taki sam jak format wyjściowy. Innymi słowy, to, co widzimy, nie jest tym, co otrzymamy lub chcielibyśmy otrzymać. Wejściowy format języków znaczników został zaprojektowany dla wygody autora. Wyjściowy język ma służyć wygodzie czytelnika.

Oczywiście, takie przedsięwzięcie wymaga posiadania środków przekształcania formatu wejściowego na format wyjściowy. Większość dokumentów XML zostaje poddana przekształceniu przed zaprezentowaniem ich czytelnikowi. Może zostać przeprowadzona transformacja na inny słownik XML, taki jak XHTML lub XSL-FO, lub na zupełnie inny format, na przykład PostScript lub RTF.

Półoficjalnym językiem transformacji XML-a jest XSLT (Extensible Stylesheet Language Transformations — transformacje rozszerzalnego języka arkuszy stylów). Dokument XSLT zawiera listę szablonów. Każdy szablon zawiera wzór opisujący dopasowanie do elementów i innych węzłów. Procesor XSLT czyta dokument wejściowy. Jeżeli napotka w nim na coś, co pasuje do szablonu arkusza stylów, wyprowadza zawartość szablonu. Część szablonu jest zazwyczaj poleceniem nakazującym procesorowi, by dołączył treść wejścia na wyjściu. Polecenie to umożliwia na przykład, aby tekst dokumentu wyjściowego był taki sam, natomiast zmianie uległo całe oznaczenie. Można napisać arkusz stylów, który dokona transformacji dokumentów DocBook na dokumenty TEI. XSLT zostanie omówiony bardziej szczegółowo w rozdziale 8., *XSLT Transformations*.

Jednak XSLT nie jest jedynym językiem transformacji przydatnym w dokumentach XML. Dostępne są inne języki arkuszy stylów, takie jak Document Style Sheet and Semantics Language (DSSSL) (*http://www.netfolder.com/DSSSL/*), jak również rozmaite narzędzia firmowe, takie jak OmniMark (*http://www.omnimark.com/*). Większość z tych narzędzi, w odniesieniu do określonych typów dokumentów, ma swoje wady i zalety. Niestandardowe programy pisane w różnych językach programowania, takich jak Java, C++, Perl i Python, mogą wykorzystywać do transformacji dokumentów różne interfejsy programowania aplikacji (API), takie jak SAX, DOM i JDOM. Te języki są przydatne w sytuacji, gdy potrzebujemy czegoś więcej niż tylko transformacji; na przykład, interpretowania pewnych elementów jako zapytań do bazy danych i wstawiania wyników tych zapytań do dokumentu wyjściowego lub odpytywania użytkownika w trakcie transformacji. Jednak najważniejszym czynnikiem decydującym o wyborze narzędzia jest to, w jakim języku i składni czujemy się najbardziej swobodnie. De linguis non disputandum est (łac. „O języku nie należy dyskutować").

Istnieje wiele różnych możliwości wyboru formatu wyjściowego transformacji. Plik PostScript może być wydrukowany na papierze, foliach do rzutników, slajdach czy nawet na koszulkach. Dokumenty PDF można oglądać za pomocą wszystkich tych mediów, a także na ekranie. Jednak w trakcie wyświetlania na ekranie PDF jest znacznie gorszy od prostego HTML-a, który jest powszechnie dostępny na różnych platformach i bardzo łatwy do wygenerowania ze źródeł XML za pomocą XSLT. Wygenerowanie pliku PDF lub PostScript wymaga zwykle przejścia przez etap pośredni, w którym specjalne oprogramowanie dokona konwersji niestandardowego formatu wyjściowego XML, na przykład XSL-FO, na taki format, jaki jest rzeczywiście potrzebny.

Alternatywą dla prezentacji opartych o transformacje jest zapewnienie opisowego arkusza stylów, który określa, jak powinien być sformatowany każdy element w oryginalnym dokumencie. Zadanie to można wykonać za pomocą kaskadowych arkuszy stylów — Cascading Stylesheet (CSS). CSS działa szczególnie dobrze w przypadku dokumentów narracyjnych, które wymagają tylko listy czcionek, stylów lub rozmiarów, które należy zastosować wobec zawartości każdego elementu. Kiedy zostanie usunięte całe oznaczenie, pozostanie nam czysto tekstowa wersja tego, co chcielibyśmy zobaczyć. Ten sposób nie sprawdza się zbyt dobrze w przypadku dokumentów zorientowanych

na dane. Nieprzetworzona zawartość tych dokumentów może być tylko niemożliwą do rozróżnienia masą liczb, dat lub innych informacji (trudnych do zrozumienia bez kontekstu i przypisów, oferowanych przez oznaczenie). W takim przypadku najlepiej wybrać wyjście pośrednie: w wyniku transformacji powstaje nowy dokument, zawierający uporządkowane i opisane informacje. Wtedy arkusz stylów CSS może wobec elementów przetransformowanego dokumentu zastosować reguły stylów.