Język XSL*

Tomasz Traczyk
Instytut Automatyki i Informatyki Stosowanej, Politechnika Warszawska
e-mail: ttraczyk@ia.pw.edu.pl

Abstrakt. Język XSL (*eXtensible Stylesheet Language*) stanowi uzupełnienie XML. W założeniu ma on służyć do formatowania dokumentów w XML (podobnie jak CSS dla HTML). Może być także używany do przekształcania dokumentów XML-owych; do tego szczególnie dobrze nadaje się podzbiór XSL nazwany XSLT (*XSL Transformations*). Referat stanowi kontynuację prezentacji poświęconych językowi XML z dwóch poprzednich konferencji PLOUG [1, 2]. Przedstawiono w nim podstawy XSL i XSLT oraz zaprezentowano możliwości użycia tych języków w powiązaniu z popularnymi przeglądarkami WWW oraz z DBMS Oracle.

1. Wprowadzenie

Dokument w XML powinien być zbudowany na zasadzie znakowania znaczeniowego, a nie typograficznego. Cała informacja o sposobie formatowania dokumentu przez przeglądarkę musi być zatem sformułowana osobno.

Do określenia sposobu prezentacji dokumentów służą tzw. arkusze stylistyczne (*stylesheets*). Właśnie do definiowania arkuszy stylistycznych dla dokumentów w XML stworzono język XSL (*eXtensible Stylesheet Language*).

Pomysł sposobu przetwarzania, wykorzystany w XSL, pochodzi z DSSSL (*Document Style Semantics and Specification Language*) — związanego z SGML języka podobnego do LISP-u. XSL przejął od DSSSL ogólną ideę, ale jego składnia została w pełni zdefiniowana w XML, z użyciem przestrzeni nazw (*namespaces*).

XSL, pomyślany pierwotnie jako środek definiowania arkuszy stylistycznych, okazał się także znakomitym narzędziem do przekształcania dokumentów XML — takie zastosowanie wydaje się też szczególnie interesujące dla projektantów systemów z bazami danych.

2. Podstawy XSL

Prezentacja dokumentu z użyciem XSL polega na transformacji drzewa hierarchii znaczników dokumentu wejściowego na drzewo tzw. *formatting objects*, którym przypisano określone sposoby prezentacji. Przeglądarka służąca do prezentacji powinna rozumieć język owych obiektów formatujących.

Obecnie przeglądarki, za pomocą których prezentowane są dokumenty XML, nie mają możliwości interpretacji opisu formatowania wyrażonego wprost w XSL, ale są przystosowane do prezentacji dokumentów w HTML. Praktycznym sposobem formatowania dokumentów za pomocą XSL jest więc przetworzenie ich na HTML. Tak też skonstruowany został przykład podany poniżej.

2.1. Rzut oka na XSL

Podano tu prosty przykład, który pozwala zorientować się, jak wyglądać może arkusz stylistyczny w XSL i jaka jest zasada jego działania.

* Praca wspierana z grantu Dziekana Wydziału Elektroniki i Technik Informacyjnych Politechniki Warszawskiej.

Przykład 1

Przedstawiony przykład[1] dokumentu w XML pochodzi z rzeczywistego systemu informacyjnego, wspomagającego zarządzanie dużym wydziałem wyższej uczelni [1, 2].

Oto przykładowy dokument:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<!DOCTYPE eres_konspekty SYSTEM "konspekty.dtd">
<?xml-stylesheet type="text/xsl" href="konspekty.xslt"?>
<!-- Komentarz: to jest przykład nr 1 -->
<eres_konspekty>
   <przedmiot id=”KBD2” wersja=”1”>
      <slowo_kluczowe>bazy danych</slowo_kluczowe>
      <slowo_kluczowe>Oracle</slowo_kluczowe>
      <konspekt>
         <czesc_konspektu id=”Streszczenie”>
            <P> Monograficzny przedmiot poświęcony bazie danych i narzędziom
                Oracle.
            </P>
         </czesc_konspektu>
         <czesc_konspektu id=”Treść”>
            <P> Omawiane są podstawowe zagadnienia związane z wykorzystaniem
                RDBMS Oracle7 i <I>Oracle8</I> oraz administrowaniem nimi.
            </P>
            <P> Przedstawiane są także narzędzia do budowy aplikacji:
            </P>
            <UL>
               <LI> Oracle Forms, </LI>
               <LI> Oracle Reports. </LI>
            </UL>
         </czesc_konspektu>
      </konspekt>
   </przedmiot>
</eres_konspekty>
```

Przykład 2

Poniższy skrypt w XSL służy do prezentacji konspektu dokumentu z przykładu 1.

```
<?xml version="1.0" encoding="windows-1250"?>
<!-- To jest przykład nr 2 -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:output method="html" indent="yes" encoding="windows-1250" />
   <xsl:template match="eres_konspekty">
      <xsl:comment>Wygenerowano za pomocą skryptu konspekty.xslt</xsl:comment>
      <HTML>
         <HEAD><TITLE>Konspekty</TITLE></HEAD>
         <BODY>
            <H1>Konspekty przedmiotów</H1>
            <xsl:apply-templates select="przedmiot">
              <xsl:sort select="@id" />
            </xsl:apply-templates>
         </BODY>
```

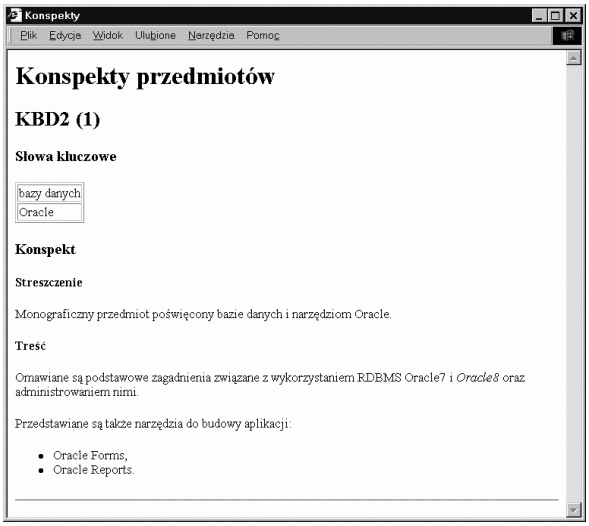
[1] Ten sam przykładowy dokument XML był prezentowany w referacie na poprzedniej konferencji PLOUG [2]. Teraz jednak będzie on formatowany za pomocą innego skryptu XSL, z wykorzystaniem nowszej wersji języka.

```
      </HTML>
    </xsl:template>
    <xsl:template match="przedmiot">
      <H2>
        <xsl:value-of select="@id" />
        (<xsl:value-of select="@wersja" />)
      </H2>
      <xsl:if test="slowo_kluczowe">
        <H3>Słowa kluczowe</H3>
        <TABLE BORDER="1">
        <xsl:for-each select="slowo_kluczowe">
          <xsl:sort select="text()"/>
          <TR><TD><xsl:value-of select="."/></TD></TR>
        </xsl:for-each>
        </TABLE>
      </xsl:if>
      <xsl:apply-templates select="*[name() != 'slowo_kluczowe']"/>
      <HR/>
    </xsl:template>
    <xsl:template match="konspekt">
      <H3>Konspekt</H3>
      <xsl:apply-templates />
    </xsl:template>
    <xsl:template match="czesc_konspektu">
      <H4>
        <xsl:value-of select="@id"/>
      </H4>
      <xsl:apply-templates />
    </xsl:template>
    <xsl:template match="P | I | UL | OL | LI">
      <xsl:element name="{name()}">
        <xsl:apply-templates/>
      </xsl:element>
    </xsl:template>
  </xsl:stylesheet>
```

Istotę działania powyższego skryptu stanowi rekurencyjne przetwarzanie znaczników. Instrukcja `xsl:apply-templates` jest wywołaniem owej rekurencji. Atrybuty `select` zawężają zakres działania do wybranych elementów dokumentu. Atrybuty `match` podają wzorzec znacznika, który jest przetwarzany za pomocą danego szablonu `template`. Wzorce mają postać ścieżek (*paths*), określających miejsce elementu lub atrybutu w hierarchicznej strukturze dokumentu.

Instrukcje `value-of` powodują włączenie odpowiedniej części przetwarzanego dokumentu do dokumentu wynikowego. Nazwy poprzedzone znakiem `@` oznaczają odwołanie do wartości atrybutu.

Efekty formatowania przykładowego dokumentu za pomocą powyższego skryptu przedstawia Rys. 1.

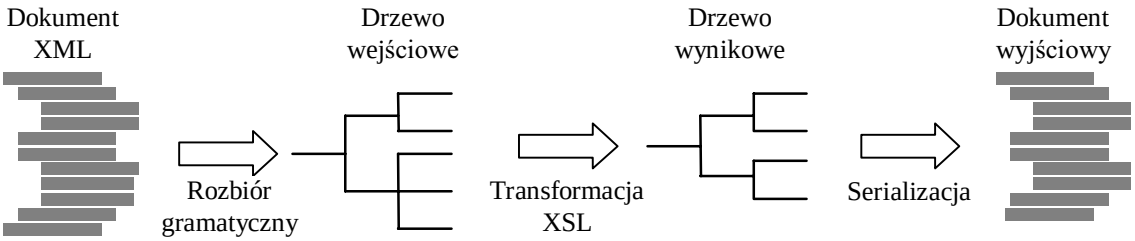



```
<!--Wygenerowano za pomocą skryptu
konspekty.xslt-->
<HTML>
   <HEAD>
      <META http-equiv="Content-Type"
      content="text/html;
      charset=windows-1250">
      <TITLE>Konspekty</TITLE>
   </HEAD>
   <BODY>
      <H1>Konspekty przedmiotów</H1>
      .........
   </BODY>
</HTML>
```

Rys. 1. Wynik formatowania dokumentu XML za pomocą XSL

2.2. Zasada działania XSL

Przetwarzanie za pomocą XSL dotyczy drzew reprezentujących hierarchię elementów i atrybutów. Schemat przetwarzania przedstawiono na Rys. 2.



Rys. 2. Fazy przetwarzania XSL

Wejściowy dokument jest poddawany rozbiorowi gramatycznemu przez parser. W wyniku rozbioru powstaje drzewo reprezentujące hierarchię elementów i atrybutów.

Skrypt w języku XSL steruje przekształceniem tego drzewa w drzewo wynikowe, które może mieć zupełnie inną strukturę. Następnie drzewo wynikowe może być — w wyniku procesu tzw. serializacji — przekształcone w dokument wynikowy, np. w XML lub HTML. Zamiast serializacji może być wykonane inne działanie, np. bezpośrednia prezentacja drzewa wynikowego przez przeglądarkę, poddanie drzewa wynikowego kolejnym przekształceniom itp.

Istotą działania XSL jest więc przekształcanie drzew, reprezentujących strukturę dokumentów. Przekształcenie takie nazywa się transformacją XSL (*XSL transformation*).

Aby przekształcenie było możliwe, dokument wejściowy musi być poprawny przynajmniej w sensie *well-formed*, by parser XML mógł przekształcić go do postaci drzewa.

Poprawność dokumentu XML w sensie *well-formed* (patrz [3, 4]) oznacza przede wszystkim, iż dokument ma dokładnie jeden element główny, znaczniki początkowe i końcowe elementów nie są „skrzyżowane” (każda konstrukcja zawarta w innej, musi być w niej zawarta w całości) oraz wszystkie znaczniki są jawnie zakończone.

Przykład 3

Poniżej podano dwa małe dokumenty w HTML:

- lewy jest poprawnym dokumentem HTML, ale nie jest poprawnym w sensie *well-formed* dokumentem XML;
- prawy jest poprawny także w sensie *well-formed* i może być przetwarzany za pomocą narzędzi dla XML.

```
Paragraf pierwszy.<P>
Paragraf drugi.
<UL>
   <LI> Pozycja 1.
   <LI> Pozycja 2.
</UL>
```

```
<HTML>
 <P>Paragraf pierwszy.</P>
 <P>Paragraf drugi.</P>
 <UL>
  <LI>Pozycja 1.</LI>
  <LI>Pozycja 2.</LI>
 </UL>
</HTML>
```

Proces serializacji nie musi prowadzić do powstania dokumentu poprawnego z punktu widzenia specyfikacji XML — np. wyjściowy dokument w HTML może nie spełniać warunków zamykania znaczników.

Przekształcenia są sterowane za pomocą tzw. szablonów (*template rules*). Szablon zawiera zwykle atrybut `match`, podający wzorzec (*pattern*), który określa elementy drzewa wejściowego przetwarzane za pomocą szablonu. We wnętrzu szablonu określa się, co ma znaleźć się w drzewie wynikowym. Przetwarzanie ma charakter rekurencyjny.

Jako uzupełnienie przetwarzania rekurencyjnego stosowane jest także przetwarzanie proceduralne: pętle przebiegające przez wybrane węzły drzewa wejściowego, warunki, rozgałęzienia. Możliwe jest też wywoływanie skryptów w języku ECMAScript[2].

XSL ma służyć do definiowania arkuszy stylistycznych, zatem oprócz definiowania przekształceń winien także umożliwiać określanie sposobu prezentacji. Służą do tego elementy zwane obiektami formatującymi (*formatting objects*). Jeśli w drzewie wynikowym znajdą się owe obiekty, to powinny one być interpretowane przez przeglądarkę i sterować sposobem prezentacji.

2.3. Części i wersje specyfikacji XSL

Ponieważ część języka XSL sterująca przekształceniami jest kluczowa, w toku prac nad językiem wydzielono ją w osobną specyfikację, nazwaną XSLT (*XSL Transformations*) [9]; prace nad tą częścią specyfikacji w zasadzie ukończono.

W transformacjach XSL do znalezienia elementu w drzewie wejściowym używane są wzorce (*patterns*) zapisywane w postaci tzw. ścieżek (*paths*). Specyfikacja owych ścieżek — wspólna dla XSL, języka XPointer (patrz [3, 4]) oraz XML Query — została wydzielona pod nazwą XPath [10] i także jest ukończona.

Pełna definicja języka XSL składa się zatem z definicji przekształceń XSLT, wzorców XPath oraz słownika obiektów formatujących.

Niestety, prac nad specyfikacją obiektów formatujących jeszcze nie ukończono, nie ma też ich implementacji w popularnych przeglądarkach. Dlatego obecnie najbardziej typowym sposobem wykorzystania XSL do tworzenia arkuszy stylistycznych jest pisanie skryptów przekształcających XML na HTML. Wydawałoby się, że do tego celu należy użyć wersji języka zawartej w specyfikacji XSLT, lecz niestety i ta część XSL nie jest jeszcze zaimplementowana w popularnych przeglą-

[2] ECMAScript jest to ustandaryzowana odmiana języka JavaScript.

darkach. Dlatego do pisania arkuszy stylistycznych używa się niekiedy starszej wersji XSL, której procesor jest częścią przeglądarki Microsoft Internet Explorer 5 (MSIE 5) — i tak skonstruowano przykład podany w zeszłorocznym referacie [2].

Znacznie wygodniejszej nowszej wersji języka transformacji (wg specyfikacji XSLT) można natomiast z powodzeniem używać do samego przekształcania dokumentów XML w inne dokumenty XML lub HTML, istnieją już bowiem procesory XSL zgodne z najnowszą specyfikacją.

Ponieważ prace nad pełną definicją języka trwają, obecnie dostępna jest jedynie wersja *working draft* specyfikacji całości XSL [7].

2.4. Składnia języka

Składnia XSL została w pełni zdefiniowana w XML, z użyciem przestrzeni nazw (*namespaces*).

2.4.1. Prolog i element główny

Skrypt (czy arkusz stylistyczny) w XSL jest dokumentem XML, może więc rozpoczynać się od instrukcji przetwarzania `<?xml... ?>`. Jeśli skrypt XSL ma zawierać znaki narodowe w kodowaniu innym od UTF-8 (standardu dla XML), to ta instrukcja przetwarzania jest obowiązkowa i musi zawierać atrybut `encoding`, podający właściwe kodowanie (patrz przykład 2).

Jak każdy dokument XML-owy, skrypt w XSL zawiera jeden element główny `xsl:stylesheet`. W znaczniku elementu głównego definiuje się przestrzeń nazw (patrz [3, 4]) języka XSL. Dla wersji XSL zaimplementowanej w MSIE 5.0 znacznik początkowy wygląda tak:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
```

zaś dla XSLT tak:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
```

przy czym atrybut `version` jest obowiązkowy. Procesory XSL powinny analizować adres URI podany w atrybucie `xmlns` i w zależności od niego odpowiednio interpretować zawartość skryptu.

Oczywiście w skryptach XSL można także umieszczać komentarze, w sposób właściwy dla XML (patrz przykład 2).

2.4.2. Przypisanie arkusza stylistycznego do dokumentu

Jeśli skrypt XSL ma pełnić dla dokumentu XML rolę arkusza stylistycznego, to potrzebne jest powiązanie dokumentu XML z owym skryptem. Sposób wskazania domyślnego arkusza stylistycznego określa osobna specyfikacja [11]. Definiuje ona instrukcję przetwarzania o następującej postaci (patrz przykład 1):

```
<?xml-stylesheet type="text/xsl" href="plik_arkusza_stylistycznego"?>
```

Instrukcję tę umieszcza się w prologu formatowanego dokumentu XML.

Ponieważ dokument może wymagać różnego formatowania w zależności od sposobu przeprowadzania prezentacji (np. od programu do tego służącego), w jednym dokumencie można umieścić kilka instrukcji `<?xml-stylesheet?>`, różniących się wartością pseudoatrybutu `media`. W pliku z przykładu 1 można by np. umieścić dodatkowo instrukcję postaci:

```
<?xml-stylesheet type="text/xsl" media="lynx" href="konspekty.lynx.xslt"?>
```

— instrukcja ta byłaby uwzględniona, jeśli do prezentacji użyty byłby program *lynx*. Ta z instrukcji `<?xms-stylesheet?>`, która nie ma pseudoatrybutu `media`, wskazuje domyślny arkusz stylistyczny.

Zwykle jest możliwe także takie wywołanie procesora XSL, by użyty został arkusz stylistyczny różny od arkuszy podanych w samym dokumencie.

2.4.3. Sterowanie serializacją

W skryptach zgodnych ze specyfikacją XSLT można umieścić instrukcję `xsl:output`, sterującą procesem serializacji. Określa się w niej takie cechy wyjściowego pliku tekstowego, jak sposób kodowania znaków narodowych, tworzenie wcięć w tekście, umieszczanie w tekście wyjściowym instrukcji przetwarzania `<?xml ...>` itp.

Najważniejszym parametrem serializacji jest tzw. metoda. Przewidziano trzy metody standardowe:

- `xml` — wyprowadzanie pliku w XML, poprawnego przynajmniej w sensie *well-formed*;
- `html` — wyprowadzanie pliku w HTML (nie jest on poprawnym plikiem XML, gdyż nie przestrzega się XML-owych reguł zamykania znaczników);
- `text` — wyprowadza się wyłącznie tekstową zawartość elementów, ale bez samych znaczników.

Przykładowa instrukcja sterująca serializacją może mieć postać:

```
<xsl:output method="xml" indent="yes" omit-xml-declaration="yes" encoding="iso-8859-2" />
```

Inny przypadek pokazano w przykładzie 2.

2.4.4. „Ciało" skryptu

„Ciało" skryptu zawiera zwykle wiele szablonów (*template rules*) służących do przetwarzania kolejnych „poziomów" drzewa wejściowego. Mogą też tu znajdować się definicje zmiennych i parametrów globalnych.

2.4.5. Wzorce

W atrybutach `match` szablonów podaje się wzorce służące do wyszukiwania odpowiednich elementów w drzewie wejściowym. Wzorce te są zapisane w języku XPath, o którym więcej napisano dalej.

3. XSL bardziej szczegółowo

W kolejnym rozdziale opisano bardziej szczegółowo poszczególne składniki XSL.

3.1. Transformacje XSL

Istotą działania XSL jest transformacja drzewa wejściowego w inne drzewo reprezentujące elementy i atrybuty. Transformacja polega więc na wyszukiwaniu węzłów w drzewie wejściowym i tworzeniu węzłów drzewa wyjściowego.

3.1.1. Szablony i przetwarzanie rekurencyjne

Transformacją drzewa wejściowego sterują szablony. Drzewo jest przeglądane od korzenia i wykonywana jest próba dopasowania wzorców `match` istniejących szablonów. Jeśli dopasowanie uda się, a „ciało" szablonu zawiera instrukcję `xsl:apply-templates`, to proces jest powtarzany rekurencyjnie.

Oczywiście, aby dany szablon mógł być w ogóle wykorzystany w przetwarzaniu, niezbędne jest, by proces rekurencyjnego przetwarzania drzewa doszedł do poziomu elementów przetwarzanych przez ów szablon. Muszą więc istnieć szablony przetwarzające nadrzędne poziomy w drzewie i przekazujące przetwarzanie (za pomocą instrukcji `xsl:apply-templates`) do kolejnych niższych poziomów hierarchii.

W standardzie określono, iż szablon inicjujący przetwarzanie drzewa (tj. powodujący przetwarzanie elementów bezpośrednio podrzędnych korzenia drzewa), czyli

```
<xsl:template match="/">
   <xsl:apply-templates/>
<xsl:template/>
```

jest szablonem wbudowanym (*built-in template*) i nie wymaga się jego podawania.

Przetwarzanie kolejnych elementów kończy się na poziomie liści drzewa. Liście drzewa odpowiadają zawartości elementów najniższego poziomu — a więc tekstowi nie zawierającemu już znaczników. Najczęściej ta zawartość ma być przepisana do drzewa wyjściowego. Standard XSL określa właśnie taki domyślny sposób przetwarzania, czyli

```
<xsl:template match="text()">
   <xsl:value-of/>
</xsl:template>
```

jest także szablonem wbudowanym i nie wymaga się jego podawania.

W implementacjach (np. w MSIE 5) szablony wbudowane bywają jednak nieuwzględnione, niezbędne jest wówczas jawne ich podanie, co też uczyniono w przykładzie podanym w referacie zeszłorocznym [2].

3.1.2. Kontekst

Wzorce (*patterns*), służące do wybierania elementów podlegających przetwarzaniu, mają postać ścieżek (*paths*) podających współrzędne (*axis*) szukanego elementu względem bieżącego kontekstu. Kontekst ten zależy od sytuacji, w której wywołana została instrukcja powodująca dopasowywanie wzorców.

Najważniejszym kontekstem jest ten związany z wnętrzem szablonu. Przyjęto, że kontekstem instrukcji działających w „ciele" szablonu jest ten element, który jest właśnie przetwarzany przez ów szablon. Wzorce używane w „ciele" szablonu są więc podawane względem bieżącego elementu dopasowanego do wzorca `match` szablonu.

Rekurencyjne przetwarzanie powoduje więc „wędrówkę" kontekstu po kolejnych poziomach drzewa wejściowego. Uaktywnienie szablonu na skutek działania instrukcji `xsl:apply-templates` powoduje zmianę kontekstu o jeden poziom w kierunku liści drzewa[3].

Dzięki temu rozwiązaniu wzorce `match` szablonów nie muszą zawierać ścieżek bezwzględnych i ten sam szablon może być wykorzystany do przetwarzania takich samych znaczników umieszczonych na różnych poziomach hierarchii elementów. W przykładzie 2 w taki sposób są przetwarzane np. znaczniki `<I>`.

3.1.3. Konflikty wzorców

Zdarzyć się może, iż do danego elementu w drzewie wejściowym pasują wzorce kilku szablonów. Ów konflikt jest rozstrzygany z pomocą określonych w specyfikacji priorytetów. Stosowany jest tylko ten z szablonów, którego wzorzec ma najwyższy priorytet.

Jeśli priorytety pasujących wzorców są równe, to decyduje kolejność umieszczenia szablonów w skrypcie — stosowany jest ostatni z szablonów.

[3] W XSLT istnieje ważny wyjątek od tej reguły, dotyczący szablonów nazwanych.

3.1.4. Przetwarzanie proceduralne

Uzupełnieniem przetwarzania rekurencyjnego za pomocą szablonów jest przetwarzanie proceduralne. W „ciele" szablonu może znajdować się program zapisany za pomocą instrukcji pętli `xsl:for-each` czy warunków `xsl:if` lub `xsl:choose`.

Pętla `xsl:for-each` umożliwia iteracyjne przeglądanie elementów dopasowanych do wzorca podanego w atrybucie `select`. W przykładzie 2 wykorzystano pętlę do wypełniania tabeli.

Instrukcja warunkowa `xsl:if` wykorzystywana jest często do tworzenia elementów drzewa wyjściowego pochodzących od opcjonalnych elementów wejściowego dokumentu. Jeśli w atrybucie `test` podamy wzorzec, to warunek będzie spełniony gdy wskazywany przez ów wzorzec element lub atrybut istnieje. Wzorce definiowane są w języku XPath, w którym można podawać warunki logiczne w postaci tzw. predykatów.

Instrukcja `xsl:if` nie ma części *else*. Jeśli więc potrzebna jest tego typu konstrukcja, to wykorzystać należy instrukcję zwrotnicy `xsl:choose`. Ma ona postać ciągu alternatywnych bloków warunkowych, z możliwością podania bloku domyślnego:

```
<xsl:choose>
   <xsl:when test="...">
      ...
   </xsl:when>
   ...
   <xsl:otherwise>
      ...
   </xsl:otherwise>
</xsl:choose>
```

3.1.5. Tworzenie drzewa wyjściowego

Instrukcje i teksty zawarte w „ciele" szablonu mają za zadanie utworzenie odpowiedniej części drzewa wyjściowego. W tym celu określono szereg instrukcji służących do tworzenia struktury drzewa wyjściowego, np.:

- `xsl:element` służy do tworzenia elementów w drzewie wyjściowym (patrz przykład 2);
- `xsl:attribute` umożliwia dodanie atrybutu do ostatnio utworzonego elementu;
- `xsl:text` umieszcza w drzewie wyjściowym podany w instrukcji tekst;
- `xsl:comment` pozwala umieszczać w drzewie wyjściowym komentarze (patrz przykład 2);
- `xsl:processing-instruction` tworzy instrukcję przetwarzania.

Do przenoszenia do drzewa wyjściowego zawartości elementów lub atrybutów służy instrukcja `xsl:value-of` (patrz przykład 2). Instrukcje `xsl:copy` i `xsl:copy-of` pozwalają na skopiowanie całych fragmentów drzewa wejściowego do drzewa wyjściowego.

Przyjęto też ważną i bardzo wygodną zasadę, iż wszystkie części „tekstowe" skryptu — tj. te, które nie należą do przestrzeni nazw `xsl` lub innej przestrzeni nazw znanej procesorowi XSL — są po prostu kopiowane „literalnie" (tzw. *literal result elements*) do drzewa wyjściowego. Jeśli w owych częściach są znaczniki (np. HTML), to muszą spełniać warunki poprawności w sensie *well-formed*. Kontekst, do którego skopiowane zostaną odpowiednie części skryptu, jest wyznaczony przez instrukcję XSL wewnątrz której znajduje się kopiowany tekst. Jest to bardzo często wykorzystywana cecha; użyto jej też w przykładzie 2 — w ten sposób tworzone są np. wyjściowe znaczniki HTML.

3.1.6. Sortowanie

W tych instrukcjach XSL, które przetwarzają zbiory elementów (np. `xsl:apply-templates` czy `xsl:for-each`) możliwe jest wykonanie sortowania, tj. przekazanie do drzewa wyjściowego

elementów uporządkowanych względem wartości elementu lub atrybutu. W starszej wersji XSL do określenia sposobu sortowania służył atrybut `order-by` (patrz przykład w referacie [2]). W XSLT wprowadzono dodatkową instrukcję o większych możliwościach `xsl:sort` (patrz przykład 2). Wartość atrybutu `select` decyduje o tym, według zawartości którego elementu czy atrybutu wykonane ma być sortowanie. Określić można także inne cechy sortowania, np. język (od którego zależy kolejność znaków alfabetu), porządek (rosnący lub malejący) itp.

3.2. XPath

W wielu instrukcjach XSL stosuje się wzorce do wyszukiwania odpowiednich elementów lub atrybutów w drzewie wejściowym. Ponieważ takie same wzorce, zapisane w postaci ścieżek (*paths*), używane są także do adresowania fragmentów dokumentów w języku XPointer (patrz [4]) i zapewne będą podstawą specyfikacji języka zapytań XML Query (patrz [4]), specyfikację języka wzorców wydzielono i nazwano XPath [10].

Wyszukiwanie odbywa się w drzewie wejściowym. Węzły (*nodes*) tego drzewa odpowiadają elementom, atrybutom oraz tekstom stanowiącym zawartość elementów. Także komentarze i instrukcje przetwarzania powodują powstanie odpowiednich węzłów w drzewie wejściowym.

Ścieżki, za pomocą których zapisuje się wzorce, mają odzwierciedlać miejsce poszukiwanego elementu w drzewie — wzorowano się tu zapewne na ścieżkach znanych z systemu Unix. Ścieżka określa położenie szukanych węzłów w drzewie wejściowym oraz warunki które muszą spełniać szukane węzły.

Ścieżka XPath składa się z oddzielonych ukośnikami (`/`) kroków (*steps*). Każdy z kroków może być:

- specyfikatorem współrzędnych (*axis specifier*);
- testem węzła (*node test*) ewentualnie uzupełnionym predykatem.

W wyniku wyszukiwania za pomocą wzorca XPath odnajdywane są odpowiednie węzły drzewa wejściowego. Jeśli wzorzec „pasuje” do wielu obiektów, to wynik wyszukania zawiera je wszystkie.

3.2.1. Współrzędne

Zapis oznaczający tzw. współrzędne (*axis*) wyszukiwanego elementu służy do określenia miejsca w drzewie wejściowym, w którym wyszukiwane będą elementy. Współrzędne są zwykle podawane względem bieżącego kontekstu; można też podać współrzędne bezwzględne, rozpoczynając ścieżkę od symbolu korzenia (podobnie jak w Unixie).

Do zapisu współrzędnych służą odpowiednie symbole słowne; najważniejsze z nich mają też skróty. Najczęściej używane specyfikatory współrzędnych podaje Tabela 1.

Tabela 1. Najczęściej używane specyfikatory współrzędnych w XPath

Symbol	Skrót	Opis znaczenia
`/` (na początku ścieżki)		Korzeń drzewa
`self`	`.`	Bieżący węzeł
`parent`	`..`	Węzeł bezpośrednio nadrzędny
`ancestor`		Węzły nadrzędne (niekoniecznie bezpośrednio)
`child`		Węzły bezpośrednio podrzędne (współrzędne domyślne)
`descendant`		Węzły podrzędne (niekoniecznie bezpośrednio)
`descendant-or-self`	`//`	Węzły podrzędne wraz z bieżącym
`attribute`	`@`	Atrybuty bieżącego elementu

Słowne symbole współrzędnych oddziela się od następujących po nich kroków znakiem podwójnego cudzysłowu, np. `descendant::przedmiot` oznacza wszystkie elementy typu `przedmiot`, podrzędne (niekoniecznie bezpośrednio) w stosunku do bieżącego węzła.

3.2.2. Testy węzłów

Test węzła (*node test*) podaje które elementy są wyszukiwane w określonym przez podane współrzędne obszarze drzewa. Test ten może wyszukiwać elementy, atrybuty lub węzły tekstowe. Zapis testu węzła zawiera nazwę wyszukiwanego elementu[4], nazwę atrybutu lub określenie typu węzła. Symbole typów węzłów zestawia Tabela 2.

Tabela 2. Symbole typów węzłów w XPath

Symbol	Opis znaczenia
`node()`	Dowolny węzeł
`*`	Dowolny element
`@*`	Dowolny atrybut
`text()`	Węzeł tekstowy
`comment()`	Węzeł komentarza
`processing-instruction()`	Węzeł instrukcji przetwarzania

Krok w ścieżce może zawierać alternatywę kilku testów, oddzielonych pionową kreską (`|`); wykorzystano to w przykładzie 2.

3.2.3. Predykaty

Test węzła może być ograniczany predykatem, który podaje warunek, jaki muszą spełniać wyszukiwane węzły. Predykat podaje się w nawiasach kwadratowych, bezpośrednio po tekście testu węzła.

Najprostszy predykat zawiera liczbę, która określa kolejny numer elementu, np. `przedmiot[2]` oznacza drugi element typu przedmiot. Zamiast liczby może być użyty symbol funkcji określającej numer kolejny, np. `przedmiot[last()]` oznacza ostatni w kolejności element typu `przedmiot`.

Predykat zawierać może wewnątrz ścieżkę, zapisaną względem kontekstu tego węzła, którego test jest ograniczany danym predykatem. Taki predykat jest prawdziwy, jeśli owa ścieżka prowadzi do istniejącego elementu. Np. zapis `przedmiot[slowo_kluczowe]` oznacza elementy typu `przedmiot`, posiadające podrzędne elementy typu `slowo_kluczowe`. Używając funkcji `not()` możemy znaleźć przedmioty nie mające słów kluczowych: `przedmiot[not(slowo_kluczowe)]`.

Predykat może zawierać warunek logiczny, mający z reguły postać porównania. W porównaniu użyta może być wartość elementu lub atrybutu, np. `przedmiot[@id='KBD2']` wyszuka element typu `przedmiot` o identyfikatorze 'KBD2'[5]. Użyć też można specjalnych funkcji i operatorów, np. `przedmiot[position() mod 2 = 0]` wyszuka przedmioty o parzystych pozycjach.

Zestawienie ważniejszych funkcji i operatorów podaje Tabela 3.

Tabela 3. Ważniejsze operatory i funkcje w predykatach XPath

Symbol	Opis znaczenia
=, !=, <, >, <=, >=	Porównania

[4] Nazwa elementu może być poprzedzona nazwą przestrzeni nazw (*namespace*) i dwukropkiem.

[5] Jeśli w warunku wystąpić ma stała znakowa, to jest ona ujmowana w pojedyncze apostrofy.

`or`, `and`, `not(...)`	Operacje logiczne
`+`, `-`, `div`, `mod`	Operacje arytmetyczne
`position()`	Numer kolejny węzła (wśród węzłów tego samego typu)
`last()`	Numer kolejny ostatniego z węzłów tego samego typu
`count(...)`	Liczba węzłów pasujących do wzorca podanego w argumencie
`name()`	Nazwa elementu/atrybutu
`string(...)`	Konwersja na napis
`format-number(...)`	Konwersja liczby na napis z formatowaniem
`number(...)`	Konwersja na liczbę
`normalize-space(...)`	Usunięcie wiodących i końcowych spacji
`starts-with(...)`	Test początku tekstu
`contains(...)`	Test zawartości tekstu

Predykaty mogą być użyte w wielu miejscach w jednej ścieżce, wszędzie tam, gdzie użyto testu węzła. Np. `przedmiot[contains(@id,'2')]/slowo_kluczowe[. !='Oracle']` poda różne od 'Oracle' słowa kluczowe przedmiotów, których identyfikator zawiera cyfrę 2.

W warunkach porównywać można wartości elementów lub atrybutów różnych od bieżącego kontekstu — podaje się do nich ścieżkę, np. `przedmiot[//czesc_konspektu/@id='Treść']` oznacza elementy typu `przedmiot`, mające elementy podrzędne (niekoniecznie bezpośrednio) typu `czesc_konspektu` o identyfikatorze 'Treść'.

3.3. XSLT

Początkowo język transformacji nie był wydzielony ze specyfikacji XSL. W roku 1999 dokonano takiego wydzielenia i opublikowano osobną specyfikację XSLT (*XSL Transformations*) [9]. W specyfikacji tej dokonano istotnych zmian i uzupełnień języka. Powoduje to, iż wcześniejsze implementacje XSL nie działają z XSLT. W szczególności dotyczy to procesora XSL wbudowanego w popularną przeglądarkę MSIE 5.

Najważniejsze zmiany wprowadzone w specyfikacji XSLT opisano niżej.

3.3.1. Zmienne

W skrypcie XSLT można definiować zmienne. Służy do tego instrukcja `xsl:variable`, mogąca przybrać dwie postacie:

```
<xsl:variable name="nazwa_zmiennej" select="wyrażenie"/>

<xsl:variable name="nazwa_zmiennej">wzorzec</xsl:variable>
```

Pierwsza z tych postaci podaje wartość jako wynik wyrażenia, druga powoduje wyliczenie wartości jako rozwinięcia podanego wzorca. Zmienna może zawierać wartość tekstową, zbiór węzłów lub drzewo węzłów.

Zasięg zmiennej zdefiniowanej w określonym elemencie skryptu XSL obejmuje ten element oraz jego elementy podrzędne. Instrukcja `xsl:variable` może być użyta jako tzw. *top-level element*, tj. element bezpośrednio podporządkowany elementowi głównemu `xsl:stylesheet`. Tak zdefiniowana zmienna jest globalna.

Zdefiniowana zmienna może być użyta w atrybutach instrukcji XSL, wszędzie tam gdzie oczekiwana jest wartość wyrażenia. Nazwę zmiennej poprzedza się znakiem `$`.

Zmiennej (a także parametru lub wyrażenia) można użyć do podstawienia wartości takiego atrybutu, w którym oczekiwana jest wartość tekstowa. Taką cechę mają np. atrybuty `name` instrukcji `xsl:element` i `xsl:attribute`. W takich sytuacjach wyrażenie (lub nazwę zmiennej poprze-

dzoną znakiem $) ujmuje się dodatkowo w nawiasy klamrowe — oznacza to, iż wartość zmiennej lub wyrażenia będzie zamieniona na napis. Konstrukcję tę wykorzystano w przykładzie 2.

3.3.2. Tryby (*modes*)

Zdarza się, iż ten sam element powinien być przetwarzany kilkukrotnie, w różnych kontekstach. Stosując prosty mechanizm szablonów z wzorcami nie można zróżnicować sposobu przetwarzania w zależności od kontekstu. Dlatego umożliwiono dodanie do szablonu atrybutu `mode`, który umożliwia wybranie konkretnego szablonu spośród kilku mających wzorzec pasujący do danego elementu. Wywołując przetwarzanie rekurencyjne instrukcją `xsl:apply-templates` możemy ją także uzupełnić o atrybut `mode`. Wybieramy w ten sposób ten spośród szablonów o pasującym wzorcu, który ma taką samą wartość atrybutu `mode`.

3.3.3. Szablony nazwane

Inny mechanizm umożliwia wywoływanie przetwarzania za pomocą konkretnego szablonu — w ten sposób można zbudować szablon używany do przetwarzania różnych elementów, nie pasujących do wspólnego wzorca. Działa to w sposób zbliżony do wywołania procedury.

Szablon, który ma być w ten sposób użyty, uzupełnia się o atrybut o nazwie `name` — staje się on tzw. szablonem nazwanym (*named template*). Szablon taki można wywołać instrukcją

```
<xsl:call-template name="nazwa_szablonu"/>
```

W przypadku takiego wywołania kontekst, w którym działają instrukcje zawarte w „ciele" szablonu, nie zmienia się i — inaczej niż dla wywołania na skutek `xsl:apply-templates` — pozostaje taki sam, jaki obowiązywał dla miejsca, skąd szablon wywołano.

Szablon nazwany może mieć parametry. Parametry formalne szablonu definiuje się umieszczając w jego wnętrzu instrukcje `xsl:param`. Nazwę parametru podaje atrybut `name`, a wartość domyślną określa się tak, jak dla zmiennej.

Do wartości tak zdefiniowanego parametru odwoływać się można wewnątrz szablonu przez podanie jego nazwy poprzedzonej znakiem `$` — podobnie jak do zmiennych.

W celu przekazania parametrów aktualnych przy wywołaniu szablonu, w zawartości elementu `xsl:call-template` umieszcza się instrukcje `xsl:with-param`. Znowu nazwę parametru podaje się w atrybucie `name`, zaś wartość określa się tak jak dla zmiennej — podając wyrażenie w atrybucie `select` lub wzorzec we wnętrzu elementu.

3.3.4. Włączanie plików XSL

W XSLT istnieją instrukcje `xsl:include` i `xsl:import`, umożliwiające włączanie do bieżącego skryptu zawartości innego pliku XSL. Adres (URI) włączanego pliku podaje się w atrybucie `href`. Instrukcja `xsl:include` powoduje włączenie zawartości pliku w miejscu wystąpienia instrukcji, włączona zawartość jest więc traktowana tak samo jak pozostała treść skryptu. Natomiast użycie instrukcji `xsl:import` powoduje, że zaimportowane szablony są przetwarzane z niższym priorytetem niż szablony lokalne.

3.4. Formatowanie

Choć obecnie do formatowania dokumentów w XML najczęściej stosuje się przekształcanie ich na HTML, w propozycji specyfikacji XSL przewidziano do tego celu specjalny język obiektów formatujących.

Dokument XML ma być za pomocą XSLT przekształcany do postaci drzewa owych obiektów formatujących, które ma być interpretowane przez przeglądarki i wyświetlane. Możliwości formatowania w XSL stanowią mniej więcej sumę możliwości języków HTML i CSS.

W specyfikacji XSL zdefiniowano słownik obiektów formatujących. Powołano dla nich osobną przestrzeń nazw `fo`.

3.4.1. Model prezentacji dokumentu

W XSL przyjęto model prezentacji, w którym dokument może zawierać następujące obiekty:

- *page-sequence* — główne części dokumentu, różniące się między sobą rozkładem strony;
- *flow* — części dokumentu mające charakter sekcji (rozdziały, podrozdziały itp.), zawarte w obiektach *page-sequence*;
- *block* — akapity i podobne części dokumentu;
- *inline* — części akapitów, różniące się np. czcionką;
- *wrapper* — „niewidzialne" obiekty, będące częściami bloku lub *inline*, służące do „zaczepienia" dających się dziedziczyć właściwości;
- *graphic* — referencje do zewnętrznych obiektów graficznych;
- *list* — listy (blok listy składa się z elementów listy, każdy element ma wydzieloną etykietę);
- *table* — tabele (o budowie podobnej jak w HTML).

XSL zawiera także złożony model stronicowania dokumentów.

3.4.2. Właściwości i zawartość obiektów

Obiekty formatujące mają właściwości (podawane jako atrybuty), które decydują o ich sposobie prezentacji. Do typowych właściwości należą: czcionka, margines i właściwości dotyczące odstępów, właściwości dotyczące justowania, wcięcia itd., np.

```
<fo:page-sequence font-family="serif" font-size="12pt">
```

określa czcionkę używaną do formatowania „dużej" części dokumentu.

Właściwości mają wartości domyślne. Niektóre właściwości podlegają dziedziczeniu — obiekty podrzędne dziedziczą je od nadrzędnych.

Tekst podlegający formatowaniu jest umieszczany jako zawartość elementów formatujących.

4. Zastosowania XSL

Podstawowym zastosowaniem XSL jest oczywiście formatowanie dokumentów w XML. Jednakże okazało się, że XSL może być z powodzeniem wykorzystywany do innych celów, wszędzie tam, gdzie potrzebne jest przekształcanie dokumentów.

4.1. Implementacje

Obecnie istnieje sporo implementacji XSL, ale dotyczą one niemal wyłącznie języka transformacji. Żadna z popularnych przeglądarek nie rozumie języka obiektów formatujących.

Istniejące implementacje XSL zwykle związane są z parserami XML, natomiast większość istniejących przeglądarek nie zawiera obsługi XSL.

Najpopularniejszym narzędziem wyposażonym w procesor XSL jest niewątpliwie Microsoft Internet Explorer 5. Wbudowano weń obsługę większej części instrukcji XSL dotyczących transformacji, ale w wersji sprzed specyfikacji XSLT.

Niewątpliwą zaletą MSIE 5 jest fakt, że procesor XML jest wbudowany w przeglądarkę, można więc przesyłać do niej bezpośrednio dokumenty w XML i arkusze stylistyczne w XSL. Przeglądarka dokonuje transformacji na HTML i prezentacji wynikowego dokumentu. Użytkownik ma dostęp

do źródłowego dokumentu w XML (może go użyć np. do dalszego przetwarzania). Po zainstalowaniu dostępnych od niedawna poprawek użytkownik może także obejrzeć wynik transformacji — jest to bardzo przydatne w czasie uruchamiania skryptów XSL. Zarówno dokument XML jak arkusz stylistyczny mogą być przechowywane w pamięci *cache* przeglądarki.

Niestety, ponieważ w MSIE 5 wbudowano starą wersję XSL, nie można użyć nowych możliwości XSLT. Prace nad procesorem XSLT są jednak zaawansowane, można go nawet doinstalować do przeglądarki. Wywołanie nowego procesora jest jednak możliwe jedynie programowo (np. z programu VBScript) i nie może on być normalnie użyty do formatowania dokumentów *on-line*.

Konkurencja — czyli firma Netscape — także pracuje nad obsługą XML w swojej przeglądarce. Niestety, nawet w najnowszych wersjach próbnych programu nie ma wsparcia dla XSL, a arkusze stylistyczne do dokumentów XML definiować można jedynie w CSS.

4.2. XSL a Oracle

Oracle od dłuższego czasu popiera rozwój i zastosowania XML. Toteż już od ponad roku dostępne są różnorodne narzędzia umożliwiające tworzenie aplikacji XML, w tym także procesor XSLT.

Procesor ten wchodzi w skład pakietu *Oracle XML Developer's Kit* (XDK). Pakiet ów, współpracujący z DBMS Oracle8*i*, zawiera m.in. parsery XML wyposażone w procesor XSLT, przeznaczone dla języków Java, C, C++ oraz PL/SQL. Np. API parsera dla języka Java udostępnia klasy `XSLStylesheet` i `XSLProcessor` wraz z odpowiednimi metodami, które przetwarzają dokumenty XML z użyciem arkuszy stylistycznych w XSL.

Przetwarzanie za pomocą XSLT może być użyte do formatowania wyników działania wchodzącego w skład XDK narzędzia *Oracle SQL Utility for Java*. Narzędzie to, na podstawie podanego zapytania w SQL, tworzy dokument XML zawierający wynik zapytania o standardowym generycznym formacie. Dokument ten można programowo poddać transformacji za pomocą XSLT, uzyskując pożądany format wynikowy. Narzędzie może także wczytywać dane z plików XML do bazy danych, ale wówczas dane muszą być podane w pliku o ustalonej składni. Używając procesora XSLT można przekształcić oryginalny dokument XML do postaci wymaganej przez narzędzie.

Procesor XSLT pełni ważną rolę w działaniu kolejnego składnika XDK, zwanego *XSQL Servlet*. Narzędzie to przetwarza zapytania SQL opakowane w składnię XML, a wyniki zapytania także formatuje w postaci dokumentu XML o standardowej generycznej budowie. Jeśli potrzebna jest, np. do celów prezentacji, inna budowa dokumentu wynikowego, narzędzie automatycznie wywołuje procesor XSLT, który przekształca wynikowy dokument używając arkusza stylistycznego. W ten sposób ze standardowych wyników zapytania uzyskać można niemal dowolną postać prezentacji.

W skład XDK wchodzi także ciekawy pakiet zwany *XML Transviewer Java Beans*, będący zbiorem komponentów Java Beans. Jeden z tych komponentów, o nazwie *XSL Transformer Bean*, służy do przeprowadzania transformacji dokumentów XML z użyciem XSL. Komponent ten może być użyty zarówno po stronie serwera (jako składników aplikacji serwerowych lub servletów), jak po stronie klienta — np. w apletach.

Możliwości transformacji XSL są także wykorzystywane przez produkt o nazwie *Oracle Portal-To-Go*. Produkt ten transformuje dynamicznie zawartość stron internetowych na formaty rozumiane przez różnorodne urządzenia, w tym na język WML dla telefonów komórkowych z protokołem WAP. Informacje z Internetu są przekształcane do generycznego formatu XML-owego, a następnie procesor XSL przekształca w je do formatów specyficznych dla danego urządzenia.

4.3. Przykłady zastosowań XSL

Najbardziej typowym zastosowaniem XSL jest oczywiście prezentacja dokumentów XML. Ale możliwości zastosowania XSL są znacznie szersze, kilka z nich, szczególnie związanych z bazami danych, opisano poniżej.

4.3.1. Przetwarzanie stron na serwerach WWW

Za pomocą XSL można dynamicznie przetwarzać strony WWW na serwerach, by udostępnić użytkownikowi dane w pożądanej przez niego formie. Przetwarzanie takie może dotyczyć stron kreowanych dynamicznie, np. w wyniku realizacji zapytań do bazy danych — przypadek taki opisano wyżej, w odniesieniu do narzędzia *XSQL Servlet*. Przetwarzać w ten sposób można także statyczne strony w HTML lub XML — tego typu zastosowanie XSL w produkcie *Portal-To-Go* także opisano wyżej.

Ciekawym pomysłem jest dynamiczne generowanie arkuszy stylistycznych na podstawie zapamiętanych w bazie danych preferencji użytkownika. W ten sposób można tworzyć portale z usługami personalizowanymi nie tylko co do treści, ale także co do formy.

4.3.2. Wymiana danych między bazami

XML wydaje się idealnym medium do wymiany danych między bazami danych. W pliku XML przenoszone są nie tylko dane, ale także metainformacja. Jednak jeśli modele danych w bazie wyjściowej i docelowej różnią się, konieczne jest dokonanie odpowiedniej transformacji. XSL może być użyty do tej transformacji, zwłaszcza jeśli różnice między modelami są znaczne i niezbędne jest złożone przekształcenie.

4.3.3. Zastosowanie XSL do tworzenia dokumentacji

Autor tego referatu ma bardzo pozytywne doświadczenia w nieco nietypowym stosowaniu XSL. Otóż używa go ostatnio dość intensywnie do produkowania dokumentacji projektowej w złożonych projektach informatycznych.

XSL szczególnie przydaje się we wczesnych fazach projektu (planowanie, analiza strategiczna), gdy przetwarzana informacja jest w dużej części tekstowa, a nie dokonano jeszcze wyboru narzędzi dla projektu (nie można więc posłużyć się np. pakietem CASE).

Przykład 3

W dużej korporacji prowadzono projekt pilotowy mający na celu próbne zastosowanie technik *data mining* do usprawnienia procesów biznesowych. Pierwszym etapem projektu było wytypowanie tych procesów, w których zastosowanie eksploracji danych jest w ogóle możliwe (np. istnieją tam dane w odpowiedniej ilości i jakości) i daje jakieś nadzieje na poprawę procesu.

Stworzono sformalizowany sposób opisu procesów biznesowych, zależności pomiędzy nimi, magazynów danych i ich użycia w poszczególnych procesach. Wprowadzono też odpowiednie kryteria oceny.

Wyniki tak prowadzonej analizy zapisano w postaci plików w XML, o specjalnie zaprojektowanej strukturze. Przekształcając owe pliki za pomocą skryptów w XSL uzyskano różnorodne opisy i zestawienia, w tym finalny ranking procesów najbardziej odpowiednich do zastosowania eksploracji danych.

Wytworzone w ten sposób zestawienia i opisy złożyły się na główną część sprawozdania z pierwszej fazy projektu. Co ważne, uzyskane wyniki były od razu sformatowane w sposób odpowiedni do zamieszczenia w sprawozdaniu, autor nie musiał więc tracić czasu na pracochłonne redagowanie opracowania.

Ważną zaletą tego podejścia do tworzenia dokumentacji jest unikanie wielokrotnego zapisywania tej samej informacji, co oszczędza czas i — co ważniejsze — nie dopuszcza niespójności. Informacja jest wprowadzana do pliku XML jednokrotnie, a wykorzystywana jest w wielu zestawieniach, produkowanych przez odpowiednie skrypty XSL.

Kolejną ważną cechą takiego podejścia jest uniknięcie pracochłonnych czynności redakcyjnych. Dokument przekształcony za pomocą XSL do HTML daje się bez trudności przekonwertować do formatu `.rtf`, rozumianego przez większość procesorów tekstu. Miłośnicy bardziej wyrafinowanych metod składu mogą posłużyć się specjalnymi wersjami programu TEX, potrafiącymi bezpośrednio interpretować pliki w XML.

Skrypty XSL mogą oczywiście produkować zarówno sprawozdania przeznaczone do druku, jak i zawierające łączniki hipertekstowe dokumenty do prezentacji w WWW.

Nie dającą się pominąć zaletą tego zastosowania XSL jest użycie darmowych narzędzi — wiele procesorów XSL można pozyskać bez opłat. Nie są zatem wymagane żadne inwestycje, co we wczesnych stadiach projektów jest na ogół bardzo korzystne.

5. Podsumowanie

XSL jest elastycznym i silnym narzędziem, służącym do przekształcania i formatowania dokumentów w XML. Ze względu na możliwości zastosowania do złożonych transformacji dokumentów zawierających dane, XSL wydaje się być interesujący dla projektantów systemów informacyjnych z bazami danych.

Specyfikacja XSL jest na ukończeniu, a części najbardziej interesujące z punktu widzenia zastosowań związanych z bazami danych są gotowe. Pojawiły się też pierwsze zdatne do praktycznego zastosowania implementacje; jedna z nich zawarta jest w narzędziach towarzyszących DBMS Oracle8*i*.

Wydaje się, że w najbliższych latach XML będzie jednym z podstawowych środków wymiany informacji, a znajomość związanych z nim narzędzi, w tym XSL, będzie niezbędna każdemu projektantowi systemów informacyjnych.

Bibliografia

1. Traczyk T., Macewicz W.: Język XML w aplikacjach z bazami danych — możliwości zastosowania, pierwsze doświadczenia, Materiały IV Konferencji Developerów i użytkowników Oracle *Ewolucja systemów informatycznych: dane, sprzęt, oprogramowanie i aplikacje*, Zakopane 1998, ss. 143-146.
2. Traczyk T.: Język XML w aplikacjach z bazami danych — po roku, Materiały V Konferencji Developerów i użytkowników Oracle *Integracja danych i systemów informatycznych*, Zakopane 1999, ss. 45-58.
3. Traczyk T.: Wprowadzenie do języka XML, Informatyka, 12/1999, ISSN 0542 9951, ss. 8-13.
4. Traczyk T.: XML i XSL, Materiały XII Górskiej Szkoły PTI, Szczyrk, czerwiec 2000.
5. Grosso P., Walsh N.: XSL Concepts and Practical Use. XML Europe 2000, Paris, June 2000. http://www.arbortext.com/xsl/tutorial.html
6. Extensible Markup Language (XML) 1.0. W3C Recommendation.
7. Extensible Stylesheet Language (XSL) 1.0. W3C Working Draft.
8. Namespaces in XML. W3C Recommendation.
9. XSL Transformations (XSLT) 1.0. W3C Recommendation.
10. XML Path Language (XPath) 1.0. W3C Recommendation.
11. Associating Style Sheets with XML Documents. Version 1. W3C Recommendation.
12. XML Support in Oracle8i and Beyond. http://technet.oracle.com/tech/xml/
13. Using XML in Oracle Database Applications. http://technet.oracle.com/tech/xml/