

Schematy XML

Tomasz Traczyk
Politechnika Warszawska
e-mail: ttraczyk@ia.pw.edu.pl

Abstrakt. Schematy XML służą do definiowania struktury dokumentów XML i w założeniu mają zastąpić DTD (*Document Type Definition*). Przedstawiono powody dla których DTD nie jest wystarczającym środkiem do definiowania struktur dokumentów, omówiono koncepcję schematów oraz podstawy propozycji standardu W3C *XML Schema*. Zaprezentowano także możliwości przetwarzania schematów za pomocą narzędzi Oracle.

1. Dlaczego schematy?

Dokument XML przeznaczony do przetwarzania powinien być prawidłowy (*valid*): mieć składnię poprawną w sensie ogólnych zasad XML (poprawność typu *well formed*) oraz zgodną z modelem struktury zdefiniowanym dla konkretnego typu dokumentu (poprawność typu *valid*).

Możliwość zweryfikowania prawidłowości dokumentu jest zatem uwarunkowana istnieniem modelu określającego strukturę dokumentu. W SGML do definiowania takich modeli używa się tzw. DTD (*Document Type Definition*). XML przejął ten sposób i także umożliwia użycie DTD (choć w uproszczonej w stosunku do SGML wersji). DTD zawiera definicje wszystkich elementów używanych w dokumencie: nazwy elementów, następstwo i sposób zagnieżdżenia, definicje atrybutów itp.

W XML, inaczej niż w SGML, istnienie modelu struktury dokumentu nie jest obowiązkowe, zakłada się bowiem, że przeglądarki powinny umieć odczytać i wyświetlić poprawnie zbudowany dokument nawet jeśli nie mają dostępu do takiego modelu. Jeśli jednak ów model nie istnieje, to parsery XML nie mogą automatycznie sprawdzać prawidłowości dokumentu (ponad wymagania poprawności typu *well formed*). Zwykle nie jest to problemem w przypadku dokumentów przeznaczonych tylko do przeglądania. Dla dokumentów przeznaczonych do dalszego przetwarzania (np. dokumentów zawierających złożone dane) takie rozwiązanie nie wydaje się jednak racjonalne. Sprawdzanie prawidłowości dokumentu spoczywa bowiem wówczas na każdym programie analizującym konkretny dokument, nie może zaś być wykonane za pomocą standardowych narzędzi.

Modele dokumentu wyrażony za pomocą DTD pozwala sprawdzić prawidłowość samego znakowania, ale okazuje się niewystarczający dla wielu zastosowań. W DTD nie przewidziano bowiem wielu możliwości, które nie są szczególnie przydatne przy przetwarzaniu tekstów (a do tego przede wszystkim miał służyć SGML), lecz wydają się niezbędne przy przetwarzaniu struktur danych. Tymczasem XML jest często (a może nawet przede wszystkim) stosowany właśnie do wymiany danych. Typowe tego typu zastosowania XML to np. elektroniczna wymiana danych (EDI), handel elektroniczny, elektroniczne transakcje typu B2B (*business to business*).

Dlatego stosunkowo szybko zorientowano się, że potrzebne jest nowe narzędzie do definiowania modeli, uwzględniające nowe potrzeby i mające w przyszłości zastąpić DTD. Powstało kilka propozycji (np. *XML-Data*, wspierana przez Microsoft), sprawą zajęło się także konsorcjum W3C (*World Wide Web Consortium*). Przygotowało ono specyfikację języka *XML Schema* [8, 9, 10], mającą od niedawna rangę rekomendacji (co praktycznie oznacza uznanie standardu za obowiązujący). Należy się spodziewać, że w najbliższym czasie standard ten zostanie powszechnie zaakceptowany i powoli będzie wypierał DTD.

2. Schematy a DTD

Dlaczego model DTD został uznany za niewystarczający dla wielu zastosowań XML?

Otóż najważniejszą wadą DTD wydaje się brak możliwości precyzyjnego definiowania typów danych dla zawartości elementów oraz atrybutów. Poza możliwością jawnego wyliczenia dopuszczalnych wartości atrybutów oraz zastrzeżeniem, że wartość ma spełniać wymagania narzucane przez standard XML dla nazw nie ma bowiem w DTD żadnych środków do kontrolowania typów danych: zawartość elementów i atrybutów jest po prostu napisem. O ile jest to na ogół wystarczające przy określaniu struktury dokumentów tekstowych, to zupełnie nie spełnia wymagań, które stawia się zwykle strukturom danych.

Oczywiście programy interpretujące dokumenty XML-owe mogą sprawdzać poprawność danych, analizując owe stanowiące zawartość elementów napisy. Rozwiązanie takie nie jest jednak satysfakcjonujące, gdyż do badania poprawności danych trzeba każdorazowo budować specjalizowany dla danego modelu dokumentu program, a nie można użyć standardowych narzędzi (np. typowego parsera XML). To zaś stawia pod znakiem zapytania przewagę użycia XML nad innymi sposobami zapisu danych.

DTD ma też bardzo ograniczone możliwości co do rozbudowy już zdefiniowanych modeli. Określony przez DTD model jest w zasadzie zamknięty, chyba że użyje się w nim „sztuczki” polegającej na wykorzystaniu tzw. encji parametrycznych. Owe encje parametryczne stanowią rodzaj makrodefinicji, które są rozwijane w czasie interpretacji DTD. Zmiana definicji encji parametrycznej pozwala na modyfikację DTD, np. na jego rozszerzenie. Jest to jednak mechanizm nienaturalny, niewygodny i ograniczony.

DTD nie daje także programom interpretującym dokumenty XML możliwości spożytkowania faktu, że pewne części modelu są identyczne. Dla wyrażenia takich powtarzających się części modelu (np. takich samych zestawów atrybutów dla wielu elementów) w DTD można wprawdzie użyć encji parametrycznych. Jednak programy analizujące dokument XML otrzymują już rozwiniętą postać DTD, nie mogą więc w żaden sposób wykorzystywać informacji wynikających ze sposobu użycia encji parametrycznych.

DTD nie zapewnia dobrego sposobu uwzględnienia przestrzeni nazw (*XML Namespaces* [7]). Jedyną możliwością stworzenia DTD dla dokumentu używającego przestrzeni nazw polega na wpisaniu do DTD „na sztywno” przedrostków związanych z przestrzenią. Jest to bardzo odległe od intencji twórców pomysłu przestrzeni nazw.

Popularności DTD nie przysparza również fakt, iż do zapisu DTD używa się języka zupełnie różnego od XML. Projektant struktur XML-owych musi więc opanować dodatkowy dziwaczny (choć dość prosty) język. Takie rozwiązanie budzi też wątpliwości natury estetycznej oraz – co ważniejsze – uniemożliwia zastosowanie narzędzi XML-owych (np. XSLT) do przetwarzania samych modeli dokumentów.

Schematy wydają się rozwiązywać większość problemów związanych z DTD:

- umożliwiają precyzyjne deklarowanie typów danych, z wykorzystaniem bardzo rozbudowanego słownika typów elementarnych i z możliwością definiowania własnych typów;
- pozwalają na jednokrotne definiowanie powtarzających się fragmentów modelu (np. typów, grup elementów i atrybutów) i wielokrotne odwoływanie się do takich definicji;
- umożliwiają zdefiniowanie zbiorów elementów, w których liczba wystąpień każdego elementu jest kontrolowana, ale kolejność jest dowolna;
- umożliwiają precyzyjne deklarowanie niepowtarzalności wybranych wartości w określonej części dokumentu; pozwala to m.in. definiować klucze i odwołania do nich;

- pozwalają zadeklarować wiele elementów o takiej samej nazwie, ale innym położeniu w dokumencie i innej budowie;
- mają rozbudowane mechanizmy pozwalające na kontrolowane rozszerzanie i uszczegóławianie modeli dokumentów; możliwe jest także korzystanie z wielu schematów w jednym dokumencie;
- w pełni uwzględniają przestrzenie nazw i pozwalają kontrolować je w sposób zgodny z ideą tych przestrzeni (w tym także komponować nowe modele z kilku przestrzeni nazw);
- same zapisane są w XML, z użyciem przestrzeni nazw.

Oczywiście także schematy nie są pozbawione wad, m.in.:

- są dłuższe i znacznie bardziej skomplikowane od DTD (choć jest to rekompensowane większymi możliwościami i znaną składnią XML-ową);
- nie zawierają możliwości definiowania encji; jeśli więc w dokumencie mają być użyte encje, to musi on mieć DTD.

Pewne wady schematów mają, jak się wydaje, charakter przejściowy:

- schematy są jeszcze mało znane, trudno więc o ekspertów umiających je wykorzystywać; nie ma też na ten temat popularnych podręczników;
- narzędzia zawierające wsparcie dla schematów są jeszcze nieliczne, widać tu jednak ostatnio znaczący postęp.

3. XML Schema – wprowadzenie

Schemat XML składa się z deklaracji i definicji. Deklaracje określają elementy i atrybuty mogące znaleźć się w dokumentach, których modelem jest schemat. Definicje określają pomocnicze obiekty wchodzące w skład samego schematu: typy, grupy itp.

Elementy schematu bezpośrednio należące do elementu głównego <schema> są tzw. elementami globalnymi. Elementy globalne tworzą nazwane definicje (np. typów, elementów, atrybutów), do których można się odwoływać (po nazwie) w innych definicjach.

Schemat XML deklaruje nazwy elementów i atrybutów oraz ich dopuszczalne następstwo i zawieranie a także typy danych dla wartości elementów i atrybutów oraz dopuszczalne zakresy tych wartości. Określa się to definiując typy danych.

Typy mogą być

- proste, tzn. nie zawierające zagnieżdżonych elementów (atrybuty muszą być prostych typów);
- złożone – zawierające zagnieżdżone elementy.

3.1. Przykład

Podano tutaj przykład schematu dla dokumentu XML zawierającego konspekty przedmiotów wykładanych na wydziale wyższej uczelni (por. [3, 4]).

Oto przykładowy dokument XML:

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<?xml-stylesheet type="text/xsl" href="konspekty.xsl"?>
<eres_konspekty
  xmlns="http://www.elka.pw.edu.pl/eres"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```

    xsi:schemaLocation="http://www.elka.pw.edu.pl/eres konspekty.xsd">
<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
        Oracle8 i Oracle8<I>i</I>, w tym możliwości wykorzystania języka XML.
      </P>
      <P> Przedstawiane są także narzędzia Oracle:</P>
      <UL>
        <LI> Oracle Forms, </LI>
        <LI> Oracle Reports, </LI>
        <LI> Oracle XDK. </LI>
      </UL>
    </czesc_konspektu>
  </konspekt>
</przedmiot>
</eres_konspekty>

```

W elemencie głównym <eres_konspekty> zdefiniowano domyślną (nie wymagającą prefiksów) przestrzeń nazw (jest ona zgodna z przestrzenią docelową schematu XML). Powołano się także na przestrzeń nazw określającą atrybut schemaLocation, służący do wskazania schematu.

Do dokumentu tego zbudowano schemat (zapisany w pliku konspekty.xsd):

```

<?xml version="1.0" encoding="ISO-8859-2"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.elka.pw.edu.pl/eres"
  xmlns="http://www.elka.pw.edu.pl/eres"
  elementFormDefault="qualified"
  version="1.1">

  <xsd:include schemaLocation="teksty.xsd"/>

  <xsd:element name="eres_konspekty">
    <xsd:annotation>
      <xsd:documentation>Przykład na konferencję PLOUG'2001</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="przedmiot" minOccurs="1" maxOccurs="unbounded"/>
      </xsd:sequence>
    </xsd:complexType>
    <xsd:key name="id_wersji">
      <xsd:selector xpath="przedmiot"/>
      <xsd:field xpath="@id"/>
      <xsd:field xpath="@wersja"/>
    </xsd:key>
  </xsd:element>

  <xsd:element name="przedmiot">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="slowo_kluczowe" type="xsd:string"
          minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="konspekt">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element ref="czesc_konspektu"
                minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

```

```

        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:sequence>
<xsd:attributeGroup ref="identyfikatory"/>
<xsd:attribute name="wersja">
    <xsd:simpleType>
        <xsd:restriction base="xsd:unsignedByte">
            <xsd:maxExclusive value="10"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:attribute>
</xsd:complexType>
<xsd:key name="id_czesci">
    <xsd:selector xpath="konspekt/czesc_konspektu"/>
    <xsd:field xpath="@id"/>
</xsd:key>
</xsd:element>

<xsd:element name="czesc_konspektu">
    <xsd:complexType>
        <xsd:group ref="tekst" minOccurs="1" maxOccurs="unbounded"/>
        <xsd:attributeGroup ref="identyfikatory"/>
    </xsd:complexType>
</xsd:element>

<xsd:attributeGroup name="identyfikatory">
    <xsd:attribute name="id" type="xsd:Name" use="required"/>
</xsd:attributeGroup>
</xsd:schema>

```

Część modelu, zawierającą typowe definicje, przydatne zapewne w wielu typach dokumentów, wydzielono do osobnego schematu i włączono do schematu głównego za pomocą elementu <include>. Oto owa „uniwersalna” część (zawarta w pliku teksty.xsd):

```

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">

    <xsd:element name="P">
        <xsd:complexType mixed="true">
            <xsd:sequence>
                <xsd:element ref="I" minOccurs="0" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="UL">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="LI" type="xsd:string"
                    minOccurs="1" maxOccurs="unbounded"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:group name="tekst">
        <xsd:choice >
            <xsd:element ref="P"/>
            <xsd:element ref="UL"/>
        </xsd:choice>
    </xsd:group>

    <xsd:element name="I" type="xsd:string"/>
</xsd:schema>

```

Jak się łatwo domyślić, elementy należące do przestrzeni nazw oznaczonej prefiksem `xsd` stanowią składniki języka definiowania schematów.

Element `<annotation>` służy do umieszczania dodatkowych informacji o definiowanym schemacie i zawierać może elementy `<documentation>` przeznaczone dla ludzi czytających schemat oraz elementy `<appInfo>` w których umieszcza się informacje sterujące działaniem programów korzystających ze schematu.

W przykładowym schemacie zastosowano mieszaną metodę definiowania elementów: niektóre elementy są deklarowane od razu w miejscu ich wykorzystania (*inline*); tak zadeklarowano np. element `<konspekt>`. Inne są definiowane jako elementy globalne i powoływane przez nazwę (za pomocą atrybutu `ref`) w miejscu, gdzie mają być wykorzystane. Tak zdefiniowano np. elementy `<P>` i ``. Oczywiście wszelkie obiekty, które mają być wykorzystywane wielokrotnie, muszą być definiowane jako elementy globalne.

Podobnie także inne obiekty – atrybuty, typy proste i złożone – można definiować globalnie (i wykorzystywać przez odwołanie do nazwy) albo lokalnie (*inline*).

3.2. Proste typy danych

Typy proste buduje się na podstawie elementarnych typów wbudowanych. Podstawowe typy wbudowane wymieniono w tabeli 1.

Tabela 1. Wbudowane typy XML Schema (wybór)

Nazwa typu	Opis	Przykładowe podtypy
string	Napis	normalizedString, language, Name
boolean	Wartości true i false	
decimal	Liczba stałoprzecinkowa	integer, negativeInteger, nonPositiveInteger, int, short, byte, unsignedInt
float, double	Liczba zmiennoprzecinkowa	
duration	Długość okresu czasu	
dateTime	Data i czas	
date, time	Data, czas	
gYear	Rok (gregoriański)	

Niektóre z typów elementarnych mają jeszcze podtypy (*derived types*), np. typ `string` ma m.in. podtypy `normalizedString` (bez tabulatorów i znaków zmiany linii), `language` (dwuliterowe skróty języków stosowane w XML), `Name` – słowa nadające się na nazwy XML-owe itd.

W powyższym przykładzie na ogół odwoływano się bezpośrednio do wbudowanych typów elementarnych, np. tak zadeklarowano element `<slowo_kluczowe>` czy atrybuty `id`.

Można budować własne typy proste w oparciu o wbudowane typy elementarne lub zdefiniowane w schemacie typy proste. Konstruując te typy można użyć kilku środków:

- Ograniczenie (*restriction*) wykorzystuje tzw. fasety (*facets*) typów bazowych. Dla typu `string` są to np. ograniczenia na długość oraz wzorce (*patterns*) formułowane za pomocą wyrażeń regularnych (podobnych jak w Perlu), dla typów numerycznych – np. dolne lub górne ograniczenia wartości. W powyższym przykładzie wykorzystano takie ograniczenie dla określenia zakresu wartości atrybutu `wersja`.
- Wyliczenie (*enumeration*) jest rodzajem fasety, polegającym na jawnym podaniu wszystkich dopuszczalnych wartości.

- Połączenie (`union`) tworzy typ do którego należą wszystkie wartości należące do któregokolwiek z typów łączonych; stosuje się najczęściej do sumowania typów wyliczeniowych.

Specyficzny rodzaj ograniczeń wartości stanowi wymaganie niepowtarzalności wartości w ramach dokumentu lub jego części. Wymaganie niepowtarzalności może dotyczyć wartości pojedynczego elementu lub atrybutu albo złożenia kilku elementów/attributów. Takie wymaganie deklaruje się za pomocą elementów `<unique>`. Zakres, w którym wartości nie mogą się powtarzać, określa się przez umieszczenie tej deklaracji na końcu odpowiedniego elementu.

Bardzo podobne znaczenie ma deklaracja `<key>`; ona oprócz unikalności narzuca ona także niepustą zawartość wszystkich składników klucza. Takie właśnie klucze zdefiniowano w naszym przykładzie. Do takich kluczy można za pomocą elementu `<keyref>` deklorować odwołania; ich poprawność (integralność referencyjna) winna być sprawdzana w czasie walidacji.

3.3. Złożone typy danych

Złożone typy danych (`complexType`) buduje się z elementów podrzędnych. Dopuszczalne są różne sposoby grupowania tych elementów:

- ustalający ścisłą kolejność (`sequence`);
- dający wybór z kilku możliwości (`choice`);
- określający zbiór elementów i liczbę powtórzeń, ale nie narzucające kolejności (`all`).

Liczbę powtórzeń elementu w grupie określa się za pomocą atrybutów `minOccurs` i `maxOccurs` (domyślne są wartości 1), można też określić liczbę powtórzeń całej grupy. Grupy mogą być w sobie zagnieżdżane, co daje możliwość wyrażenia dowolnego modelu następstwa.

W dokumentach XML często stosowane są elementy mogące zawierać zwykły tekst przemieszany ze znacznikami. Taką mieszaną zawartość deklaruje się używając atrybutu `mixed` dla typu złożonego. Określenie sposobu grupowania (np. `sequence`) wyraża wtedy, jak zwykle, reguły występowania elementów podrzędnych, natomiast atrybut `mixed` decyduje, czy elementy te mogą być „przeplecione” tekstem. W powyższym przykładzie w taki sposób zadeklarowano element `<P>`.

Na końcu definicji typu złożonego można podać deklaracje atrybutów.

3.4. Mechanizmy wielokrotnego użycia i rozszerzanie schematów

Jedną z ważniejszych zalet schematów jest możliwość wielokrotnego użycia zdefiniowanych typów, elementów i atrybutów. Schematy pozwalają na sformalizowane modyfikowanie (np. rozszerzanie lub ograniczanie) przywoływanych definicji, ze ścisłą kontrolą możliwości ich modyfikowania.

Najprostszym z mechanizmów pozwalających na wielokrotne użycie definicji jest grupowanie. Można raz zdefiniować grupę elementów (`group`) lub atrybutów (`attributeGroup`) i wielokrotnie odwoływać się do niej. W powyższym przykładzie takiego sposobu użyto w przypadku atrybutów `id` oraz elementu `<czesc_konspektu>`.

Bardziej złożony model wielokrotnego użycia polega na wyprowadzaniu nowych definicji z już istniejących. Można w ten sposób tworzyć nowe pochodne (*derived*) typy proste i złożone. Dla typów prostych użyć można mechanizmów opisanych w części 3.2. Dla typów złożonych dostępne są zaś mechanizmy:

- ograniczenia (`restriction`), polegającego na zawężeniu zakresu dopuszczalnych wartości albo ograniczeniu liczności występowania elementów;
- rozszerzenia (`extension`), tj. dodania do typu bazowego nowych elementów lub atrybutów.

Mechanizmem związanym z wyprowadzaniem nowych typów jest zastępowanie (*substitution*) elementów oraz typów.

Zastępowanie elementów polega na możliwości zdefiniowania grupy (*substitutionGroup*) elementów, które w definiowanym dokumencie mogą być używane zamiast elementu bazowego (*head*) grupy. Elementy w takiej grupie muszą mieć typ taki sam jak typ elementu bazowego albo pochodny od niego. Zastępowania elementów można użyć np. do utworzenia wielu wersji językowych systemu znakowania.

Istnieje także mechanizm zastępowania typów, polegający na tym, że element typu bazowego może być w dokumencie zastąpiony przez wystąpienie typu pochodnego. Do określenia typu zastępującego używa się w dokumencie specjalnego atrybutu.

Twórca schematu może jawnie zdefiniować możliwości jego rozszerzania, określając za pomocą specjalnych atrybutów, jakie mechanizmy (np. rozszerzania, ograniczania lub zastępowania) nie mogą być stosowane do konkretnych typów lub elementów. Pewne elementy i typy można zdefiniować jako abstrakcyjne; oznacza to, że nie mogą one być bezpośrednio wykorzystane w deklaracjach, ale służą wyłącznie do zastępowania lub wyprowadzania nowych definicji.

Jeszcze inny mechanizm może być wykorzystany do modyfikowania deklaracji zawartych w schemacie włączanym do nowego schematu. Specjalny element `<redefine>` umożliwia przedefiniowanie wybranych elementów, atrybutów i typów włączanego schematu.

W definicjach typów złożonych można wykorzystywać specjalne elementy `<any>` i `<anyAttribute>`. Deklarują one zezwolenie na użycie w dokumencie, w miejscu przez nie określonym, znaczników w ogóle nie zdefiniowanych w schemacie. Jest to zatem zezwolenie na rozszerzanie struktury dokumentu przez twórcę dokumentu opartego na schemacie.

3.5. Schematy a przestrzenie nazw

Schematy w pełni wykorzystują możliwości przestrzeni nazw. Każdy schemat określa model dokumentu w konkretnej przestrzeni nazw, a walidacja z użyciem schematu następuje w ramach tej przestrzeni. Przestrzeń docelową, tzn. tę w której będą funkcjonować zadeklarowane w schemacie znaczniki, podaje się w atrybucie `targetNamespace` elementu `<schema>`. Atrybut `elementFormDefault` określa z kolei reguły kwalifikowania (czyli przypisania do przestrzeni nazw) deklarowanych w schemacie znaczników. Wartość `qualified` nakazuje kwalifikowanie wszystkich znaczników (tę możliwość wykorzystano w przykładzie), zaś wartość `unqualified` określa, że kwalifikowane będą tylko elementy globalne, zaś ich elementy podrzędne (lokalne) nie będą kwalifikowane. Przy tym drugim podejściu w przestrzeni nazw znajdują się tylko elementy globalne, zaś ich podrzędne elementy lokalne są do przestrzeni przypisane za pośrednictwem swych elementów nadrzędnych (globalnych). Umożliwia to tworzenie wielu elementów o takich samych nazwach (ale różnym położeniu, typie i znaczeniu) bez wprowadzania konfliktu w przestrzeni nazw.

Ponieważ schemat określa docelową przestrzeń nazw, znaczniki w dokumencie wykorzystującym schemat muszą być położone w tej przestrzeni nazw. Przypisane do przestrzeni nazw muszą być oczywiście tylko te znaczniki, które zostały zdefiniowane jako kwalifikowane. Przypisania do przestrzeni nazw zwykle dokonuje się przez użycie prefiksów, ale można także zdefiniować przestrzeń domyślną i dla niej użycie prefiksów jest zbędne; tak właśnie zbudowano dokument XML w powyższym przykładzie.

Przypisanie definiowanych przez schematy znaczników do konkretnych przestrzeni nazw może powodować kłopoty gdy chcemy w jednym schemacie wykorzystać znaczniki zadeklarowane w innym schemacie. Problemu nie ma, gdy oba schematy używają tej samej docelowej przestrzeni nazw. Jeśli schemat włączany do nowego modelu w ogóle nie określa przestrzeni, to włączenie (za pomocą elementu `<include>`) powoduje przypisanie włączanych znaczników do docelowej przestrzeni nazw schematu włączającego; nazywa się to *chameleon effect*. W przypadku schematów dostarczających „uniwersalnych” znaczników do wielokrotnego wykorzystania w

innych schematach rozsądne jest zatem pominięcie deklaracji docelowej przestrzeni nazw; tak właśnie zbudowano przykładowy schemat `teksty.xsd`.

Jeśli jednak włączamy znaczniki ze schematu który określał docelową przestrzeń nazw, to musimy pozostawić je w tej przestrzeni, tzn. zgodzić się na to, by definiowany dokument wykorzystywał więcej niż jedną przestrzeń nazw. Do takiego włączania schematów służy element `<import>`.

Z kilku przestrzeni nazw możemy także korzystać bezpośrednio w dokumencie XML. Jeśli dokument powołuje się na kilka przestrzeni nazw, to może także podać lokalizacje schematów z nimi związanych. Do walidacji konkretnego elementu użyty będzie schemat związany z przestrzenią nazw do której dany element należy.

Zapis schematu można rozszerzyć o atrybuty nie należące do standardu *XML Schema*, np. sterujące specyficznym sposobem przetwarzania dokumentu. Takie atrybuty należy umieścić w przestrzeni nazw różnej od `XMLSchema`.

4. Narzędzia

Ponieważ specyfikacja *XML Schema* jest dość nowa (status rekomendacji otrzymała dopiero w maju bieżącego roku!), nie ma jeszcze zbyt wielu narzędzi, które umożliwiłyby praktyczne zastosowanie schematów. Na szczęście jednak producenci narzędzi XML-owych szybko zaakceptowali ten standard i już pojawiają się pierwsze (na ogół jeszcze próbne) wersje narzędzi uwzględniające go.

4.1. XML Schema w narzędziach firmy Microsoft

Microsoft w poprzednich latach lansował własną specyfikację schematów (*XML-Data*, potem *XDR – XML-Data Reduced*) ale, choć zachowano wsparcie dla tamtych koncepcji, w najnowszych wersjach narzędzi pojawiła się także możliwość wykorzystania schematów *XML Schema* (XSD).

Do wykorzystania schematów XSD niezbędne jest zainstalowanie produktu Microsoft XML SDK 4.0. Zawiera on parser zgodny ze standardem DOM, który może przeprowadzać walidację dokumentów z użyciem schematów XSD. Do załadowania schematu służy specjalny obiekt `XMLSchemaCache`. Parser ten jest dostępny przez API i może być używany w programach np. w języku JScript czy VisualBasic.

4.2. XML Schema w narzędziach Oracle

W najnowsze wersje pakietu XDK (*XML Developer Kit*) firmy Oracle także włączono możliwość wykorzystania schematów *XML Schema*.

W skład XDK wchodzi *Oracle XML Schema Processor for Java*, wykorzystywany przez parser typu DOM. Parser ten może być przeprowadzać walidację z użyciem schematu, załadowanego automatycznie na podstawie zawartości dokumentu XML albo programowo – za pomocą specjalnych klas umożliwiających ładowanie schematu i manipulacje na nim.

Walidację dokumentów XML-owych z użyciem schematów można także przeprowadzać z linii komendy, za pomocą dostarczanego z XDK skryptu, który należy wywołać tak:

```
oraxml -schema <nazwa_pliku_XML>
```

5. Podsumowanie

Choć definiowanie znakowania za pomocą DTD ma długą tradycję, sposób ten nie spełnia wielu ważnych wymagań typowych dla zastosowań XML, zwłaszcza związanych z wymianą danych przez Internet.

Definiowanie modelu dokumentu za pomocą schematów, choć trudniejsze od użycia DTD, daje jednak znacznie większe możliwości, a przede wszystkim zapewnia możliwość precyzyjnego określenia typów danych, wielokrotnego wykorzystania i sformalizowanej rozbudowy modeli oraz pozwala w pełni wykorzystać zalety użycia przestrzeni nazw.

Choć specyfikacja *XML Schema* jest jeszcze całkiem nowa, już pojawiają się narzędzia programowe umożliwiające jej praktyczne zastosowanie, w tym także narzędzia firmy Oracle.

Należy zatem przypuszczać, że DTD zostaną już wkrótce wyparte przez schematy, a projektanci systemów informacyjnych zyskają nowy silny środek do formalnego definiowania złożonych struktur danych.

Bibliografia

1. Traczyk T.: Wprowadzenie do języka XML, Informatyka, 12/1999.
2. Traczyk T.: XML i XSL, Materiały XII Górskiej Szkoły PTI, Szczyrk 2000.
3. Traczyk T.: Język XML w aplikacjach z bazami danych – po roku, Materiały V Konferencji Deweloperów i Użytkowników Oracle „Integracja danych i systemów informatycznych”, Zakopane 1999.
4. Traczyk T.: Język XSL, Materiały VI Konferencji Deweloperów i Użytkowników Oracle „Systemy informatyczne w dobie Internetu”, Zakopane 2000.
5. Traczyk T.: Czy już warto używać XML? Tutorial, Materiały I Seminarium PLOUG, Warszawa 2001.
6. Extensible Markup Language (XML) 1.0, W3C Recommendation.
7. Namespaces in XML, W3C Recommendation, <http://www.w3.org/TR/REC-xml-names/>
8. XML Schema Part 0: Primer, W3C Recommendation, <http://www.w3.org/TR/xmlschema-0/>
9. XML Schema Part 1: Structures, W3C Recommendation, <http://www.w3.org/TR/xmlschema-1/>
10. XML Schema Part 2: Datatypes, W3C Recommendation, <http://www.w3.org/TR/xmlschema-2/>
11. Walsh N.: Understanding XML Schemas, <http://www.xml.com/pub/a/1999/07/schemas/>
12. Costello R.L.: XML Schemas, Tutorial, <http://www.xfront.com>
13. Jelliffe R.: The XML Schema Specification in Context, <http://www.xml.com/pub/a/2001/01/10/schemasincontext.html>
14. Oracle XML-enabled, <http://www.oracle.com/xml/>
15. Oracle XML Developer's Kit, <http://technet.oracle.com/tech/xml/>