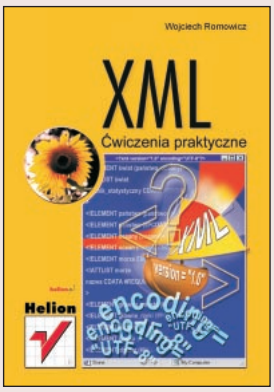
XML. Ćwiczenia praktyczne



Autor: Wojciech Romowicz
ISBN: 83-7197-549-X
Format: B5, stron: 160

XML jest językiem znacznikowym i pochodzi z tej samej rodziny języków co HTML, jednakże nie ma w nim określonych na stałe znaczników. Tworzymy je sami podczas definiowania dokumentu. Wszystkie atrybuty również są definiowane w ciele dokumentu.
Siła XML-a tkwi w jego uniwersalności i możliwości szerokiego zastosowania. Tworzenie gramatyki nowych aplikacji jest sprawą trudną i wymaga od ich twórców dobrej znajomości tematu, natomiast wykorzystywanie już gotowych aplikacji jest sprawą o wiele prostszą. Przykładem takich rozwiązań są języki WML (Wireless Markup Langu-age – język do obsługi dobrze znanego protokołu WAP), VML (Vector Markup Langu-age – język, w którym zapisana jest grafika w pakiecie Office firmy Microsoft) czy też MathML (specjalizowany język przeznaczony dla matematyków). W dodatkach znajdą państwo opis MathML-a, jako ukłon dla moich kolegów po fachu. Na razie przedstawię jedynie rysunek, aby zaostrzyć ich apetyt. Język ten umożliwia wprowadzanie symboli matematycznych i wzorów jak zwykłego tekstu.

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

Spis treści

Rozdział 4.

XML Schema

Schemat, podobnie jak DTD ma za zadanie opisać postać dokumentu XML-a, czyli elementy oraz ich zawartość, atrybuty. Dlaczego więc stworzono ten drugi mechanizm, skoro DTD robi to samo? Faktycznie DTD jest schematem dokumentu, ale go dokładnie nie opisuje. Czytając poprzedni rozdział zauważyłeś pewnie, że dane mogą być tylko rodzaju tekstowego lub binarnego. Nie ma nigdzie mowy o tym, aby dane mogły być liczbami czy też datami. Taką możliwość otrzymasz po zastosowaniu XML-a Schema. Innym faktem przemawiającym za schematami (od tej pory schemat kojarz z opisem w XML-u Schema), jest ich składnia. Składnia schematów jest identyczna ze składnią występującą w dokumentach XML-a. Kolejną zaletą schematów jest możliwość tworzenia elementów zgodnie z ich kontekstem (porównaj z uwagami końcowymi na temat DTD). Jedynymi konstrukcjami, których nie możesz zdefiniować za pomocą schematu są encje. Twórcy schematów w grupie dyskusyjnej poświęconej XML-owi Schema twierdzą, że te konstrukcje zaśmiecają składnię. Jedynym sposobem utworzenia encji jest przemieszanie DTD, w których definiujemy tylko encje ze schematami, gdzie zawarta jest reszta definicji dokumentu. Przykład znajdziesz w tym rozdziale.

Schematy umieszcza się w plikach z rozszerzeniem *xsd*, którego struktura powinna wyglądać następująco:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
  ...
</xsd:schema>
```

Jeżeli korzystasz z nowej rekomendacji (uwaga — nie wszystkie wersje oprogramowania mogą korzystać z tej przestrzeni nazw, na przykład tak zachowuje się XML Spy w wersji 4.0), rozszerzenie może mieć inną postać:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    ...
</xsd:schema>
```

Przestrzeń nazw wskazuje na to, że wszystkie elementy zaczynające się od *xsd:* będą traktowane jako definicje pochodzące z danego adresu, tutaj z rekomendacji zatwierdzonej przez organizację World Wide Web Consortium. Jak widzisz, całość definicji schematu spinamy klamrą <xsd:schema ...> ... </xsd:schema>. Teraz wystarczy przypisać plik ze schematem do elementu głównego w dokumencie XML:

```
<igrzyskaOlimpijskie rok="2000" miasto="Sydney"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\MojeXml\Examples\Olimpiady.xsd">
```

lub

```
<igrzyskaOlimpijskie rok="2000" miasto="Sydney"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\MojeXml\Examples\Olimpiady.xsd">
```

Powyższy przykład przedstawia schemat prosty, to znaczy taki, który nie korzysta lub nie definiuje własnej przestrzeni nazw; wskazuje na to deklaracja xsi:noNameSpaceSchema Location. *O przestrzeniach nazw dowiesz się w dalszej części* tej książki. Schematy zapisywane są w plikach o rozszerzeniu *xsd*, tutaj widzimy przykład deklaracji pliku *Olimpiady.xsd.*

Rozpoczniemy naukę od elementów dokumentujących schemat, czyli po prostu komentarzy do samego opisu. Aby zdefiniować taką notatkę należy skorzystać z konstrukcji:

Listing 4.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xsd:annotation>
    <xsd:appinfo>
    Ten schemat jest częścią aplikacji Olimpiady
    </xsd:appinfo>
    <xsd:documentation>
    Definicja dokumentów przechowujących dane o medalistach
    olimpijskich.
    </xsd:documentation>
  </xsd:annotation>
```

Rozpoczynamy znacznikiem <xsd:annotation>, wewnątrz którego mogą się znajdować dwa znaczniki. Pierwszy z nich, <xsd:appinfo>, powinien zawierać informację na temat aplikacji języka XML, do której należy dany schemat, a drugi — informacje o schemacie. Instruując w ten sposób innych, ułatwiamy im życie, a i nam po pewnym czasie te informacje mogą się przydać.

Ćwiczenie 4.1.

Zadeklaruj początek szablonu dla pliku sąsiedzi.xml (patrz dodatek D).

Typy elementów używanych w schematach

Przed przystąpieniem do deklaracji elementów musisz poznać ich typy. W schematach rozróżniamy dwa typy elementów *proste* i *złożone*. Typ prosty określa element, którego zawartością jest tylko tekst. Natomiast typ złożony, jak sama nazwa wskazuje, skomponowany może być z innych elementów i tekstu.

Element prosty w DTD można było opisać tylko tak, by zawierał dane w postaci `#PCDATA`, natomiast schematy pozwalają dokładnie określić zawartość takiego elementu. Mamy tu możliwość nadania zawartości jednego z typów predefiniowanych (zob. dodatek A) lub stworzenie własnego typu prostego.

Typy złożone, jak i proste, można nazywać i stosować w dowolnym miejscu schematu, mogą one również pozostać *anonimowe*, czyli bez nazwy. Typy anonimowe mogą być używane tylko w elemencie, w którym jest jego definicja.

Przykład definicji typu prostego, nazwanego, który bazuje na predefiniowanym stylu `xsd:string`:

Listing 4.2.

```
<xsd:simpleType name="płeć">
  <xsd:restriction base="xsd:string">
          <xsd:pattern value="Kobiety"/>
      <xsd:pattern value="Mężczyźni"/>
  </xsd:restriction>
</xsd:simpleType>
```

Typ złożony, anonimowy:

Listing 4.3.

```
<xsd:element name="dyscyplina" maxOccurs="unbounded">
  <xsd:complexType mixed="false">
  <xsd:sequence>
        <xsd:element name="płeć" maxOccurs="2">
           . . .
        </xsd:element>
        ...
        </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Definiowanie elementów

W XML-u Schema nie istnieje w zasadzie pojęcie elementu głównego. W zasadzie, to znaczy przy definiowaniu ten element sam się pojawi. Mamy tutaj za to podział na deklaracje *lokalne* i *globalne* elementów. Deklaracja globalna występuje wtedy, gdy opis elementu występuje na zewnątrz innych deklaracji, musi ona być jedynie zamknięta w klamrze `<xsd:schema> ... </xsd:schema>`. Pamiętamy, że wszystkie deklaracje w DTD były globalne. Deklaracja lokalna elementu polega na umieszczeniu jej wewnątrz deklaracji innego elementu, tzw. rodzica. Lokalne elementy z tą samą nazwą mogą zawierać różną zawartość w zależności od kontekstu, w którym były użyte. Globalne deklaracje natomiast umożliwiają jednokrotne utworzenie elementu powtarzającego się i korzystanie z niego w dowolnym miejscu schematu.

Przykład poniższy przedstawia definicje dwu elementów globalnych: *igrzyskaOlimpijskie* oraz *wynik*, a także jednego elementu lokalnego: *dyscyplina*.

Listing 4.4.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
            elementFormDefault="qualified">
  <xsd:element name="igrzyskaOlimpijskie">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="dyscyplina">
          ...
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="wynik" type="xsd:string"/>
xsd:schema>
```

Po deklaracji elementu musimy natychmiast podać deklarację zawartości elementu, tutaj `<xsd:complexType>`, wskazującą na typ złożony. Natomiast dla elementów końcowych, takich jak `wynik`, wystarczy zadeklarować jedynie typ zawartości jako wartość parametru `type`. Kolejnym zadaniem, jakie czeka na nas po utworzeniu zawartości złożonej, jest wskazanie struktury. Mamy do wyboru trzy możliwości:

1. `<xsd:sequence>` — sekwencja elementów znana z DTD,
2. `<xsd:choice>` — lista wyboru, również poznana przy okazji omawiania DTD,
3. `<xsd:all>` — elementy, które zadeklarowane takim znacznikiem, mogą występować w dowolnej kolejności w elemencie rodzica.

Ćwiczenie 4.2.

Na podstawie podanego pliku ze schematem, utwórz poprawnie walidowany dokument XML-a. Wytłuszczonym drukiem zapisane są atrybuty, które mówią nam o ilości wystąpień elementu w dokumencie. We wszystkich przypadkach wartość ta jest nieograniczona.

Listing 4.5.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
<xsd:element name="ZOO">
  <xsd:complexType>
  <xsd:sequence>
    <xsd:element name="gatunek" maxOccurs="unbounded">
    <xsd:complexType>
      <xsd:choice>
      <xsd:element name="osobnik" maxOccurs="unbounded" type="xsd:string"/>
        <xsd:element name="stado" maxOccurs="unbounded">
      <xsd:complexType>
      <xsd:sequence>
      <xsd:element name="osobnik" maxOccurs="unbounded" type="xsd:integer"/>
      </xsd:sequence>
      </xsd:complexType>
  </xsd:element>
  </xsd:choice>
    </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
   </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Ćwiczenie 4.3.

Odpowiedz, dlaczego ten dokument XML-a będzie niewłaściwie walidowany, popraw błędy.

Listing 4.6.

```
<?xml version="1.0" encoding="UTF-8"?>
<ZOO xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Program Files\XML Spy 3.5\Examples\trzoda.xsd">
  <gatunek>
    <osobnik>Cheetah</osobnik>
    <stado>
      <osobnik>Gucio</osobnik>
      <osobnik>2343</osobnik>
    </stado>
  </gatunek>
</ZOO>
```

Kolejną metodą, jakiej się przyjrzymy z bliska, jest definiowanie elementu globalnego i przypisanie go do wewnętrznej deklaracji. Najpierw musimy taki element zadeklarować, na przykład tak:

```
...
<xsd:element name="wynik" type="xsd:string"/>
    </xsd:schema>
```

Tak zdefiniowany element możemy teraz używać w dowolnym miejscu schematu, dodając odpowiedni odnośnik (referencję).

Listing 4.7.

```
<xsd:element name="złoto">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="wynik"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="srebro">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="wynik"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="brąz">
  <xsd:complexType>
    <xsd:sequence>
       <xsd:element ref="wynik"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Natomiast, jeżeli utworzysz typ danych (złożony lub prosty) jako globalny, to musisz podać przy deklaracji elementu atrybut *type* z nazwą typu. Deklaracja przykładowego własnego typu prostego i sposób jego zastosowania wyglądają następująco:

Listing 4.8.

```
<xsd:element name="PESEL" type="pesel"/>
  <!-- Deklaracja typu -->
  <xsd:simpleType name="pesel" base="xsd:string">
    <xsd:restriction>
      <xsd:pattern value="\d{10}"/>
    </xsd:restriction>
  </xsd:simpleType>
```

Zauważ, że typ podstawowy możemy tworzyć na podstawie typu predefiniowanego — atrybutu `base`. Istnieje również możliwość tworzenia restrykcji, czyli dostosowania typu do własnych potrzeb za pomocą wyrażeń regularnych `xsd:pattern`. Nie ma sensu opisywać tutaj wszystkich wyrażeń regularnych, pełną ich listę znajdziesz pod adresem: *http://www.w3.org/TR/2000/WD-xmlschema-2-20000922/#regexs*. Możesz również pobrać narzędzie do tworzenia własnych wyrażeń regularnych pod adresem: *http://www.xfront.com/regexGenLicense.html*. Powiem tylko tyle, że nasz typ składa się z 10 cyfr i jest utworzony na bazie typu tekstowego.

Ćwiczenie 4.4.

Utwórz kilka typów własnych za pomocą wyrażeń regularnych, które mogą się przydać w innych dokumentach. Na przykład numer telefonu, kod pocztowy czy też numer NIP.

Parametry elementów

Dla elementu możemy wyspecyfikować cały szereg parametrów, które definiują jego wygląd jeszcze dokładniej. Atrybuty te podawane są za obowiązkowym parametrem `name`, definiującym nazwę elementu. Zacznijmy od parametrów określających możliwą ilość wystąpień elementu:

- `minOccurs` — określa minimalną liczbę wystąpień elementu w sekcji; wartością może być tutaj liczba całkowita nieujemna. Jeżeli podamy wartość zero, to ten element staje się elementem opcjonalnym, natomiast podanie jakiejkolwiek liczby dodatniej powoduje, że ten element staje się wymagany w danej sekcji. Pominięcie tego argumentu ustawia wartość atrybutu na liczbę 1.
- `maxOccurs` — określa maksymalną liczbę wystąpień elementu w sekcji; wartością może być tutaj liczba całkowita nieujemna lub napis *unbounded*. Napis ten informuje o tym, że element może wystąpić nieograniczoną ilość razy. Wartość tego elementu nie może być mniejsza niż wartość parametru `minOccurs`, natomiast podanie wartości 0 oznacza zakaz stosowania tego elementu. Opuszczenie tego argumentu przy jednoczesnym zadeklarowaniu poprzedniego atrybutu, ustawia go na wartość podaną dla `minOccurs`.

Jeżeli nie wyspecyfikujemy żadnego z tych argumentów, przyjmowana jest wartość domyślna, czyli `minOccurs="1"` oraz `maxOccurs="1"`; jednokrotne wystąpienie elementu jest wymagane. Poniższy przykład definiuje element *płeć*, który może wystąpić jednorazowo lub dwukrotnie.

```
<xsd:element name="płeć" minOccurs="1" maxOccurs="2">
```

Parametry, które definiują sposób występowania oraz wartości domyślne, wiążą się z poprzednimi parametrami w następujący sposób: jeżeli podamy wartość `minOccurs>0`, to wtedy ten parametr jest wymagany (patrz predykat `#REQUIRED` w DTD). Wartość zero wskazuje na element opcjonalny (predykat `#IMPLIED`). Pozostały jeszcze dwa typy omówione przy okazji DTD — `#FIXED "`*wartość*`"` w DTD jest równoważny deklaracji parametru `fixed="`*wartość*`"`, natomiast samą wartość domyślną deklarujemy w XML-u Shema za pomocą parametru `default="`*wartość*`"`.

```
<xsd:element name="PI" fixed="3.141519">
<xsd:element name="płeć" minOccurs="2" maxOccurs="2" default="K">
```

Dla elementów z zawartością złożoną można również dodać parametr o nazwie *mixed*, wskazujący na element mieszany (tekst i inne elementy). Wartości *1* oraz *true* oznaczają zawartość mieszaną, natomiast *0* lub *false* zawartość zwykłą, bez dodatkowego tekstu.

Ćwiczenie 4.5.

Przyjrzyj się elementom zdefiniowanym w pliku sąsiedzi.xml i maksymalną ilość głównych miast w danym państwie określ jako 6. To samo zrób dla elementów glowne_rzeki.

Restrykcje

Oprócz przedstawionych poprzednio prostych elementów, mamy możliwość wprowadzania *restrykcji*, *list* i *unii*. Restrykcje, jak sama nazwa wskazuje, umożliwiają dalsze dopasowanie wartości elementu tak, by pasowała ona do wprowadzonych danych. Konstruując restrykcje należy pamiętać, że zależą one od predefiniowanych typów wartości. Inne restrykcje dotyczą typu *string*, a inne wartości numerycznych. Struktura takiej sekcji wygląda następująco:

```
<xsd:restriction base="xsd:int">
  <!-- Restrykcje -->
</xsd:restriction>
```

Konstrukcja ta ma swoje miejsce po deklaracji typu zawartości. Oto podstawowe restrykcje, jakie możemy określić (tabela 4.1).

Tabela 4.1. *Podstawowe restrykcje*

Restrykcja	Opis
`<xsd:minExclusive value="10"/>`	Wartość powinna być większa niż wartość wskazana. Dotyczy typów numerycznych, dat.
`<xsd:maxExclusive value="20"/>`	Wartość powinna być mniejsza niż wartość wskazana. Dotyczy typów numerycznych, dat.
`<xsd:minInclusive value="10"/>`	Wartość powinna być większa lub równa wartości wskazanej Dotyczy typów numerycznych, dat.
`<xsd:maxInclusive value="20"/>`	Wartość powinna być mniejsza lub równa wartości wskazanej. Dotyczy typów numerycznych, dat.
`<xsd:totalDigits value="4"/>`	Całkowita liczba cyfr występujących w elemencie; liczone są również cyfry w wartości ułamkowej. Dotyczy typów numerycznych.
`<xsd:fractionDigits value="2"/>`	Liczba miejsc dziesiętnych. Dotyczy typów numerycznych, ułamkowych.
`<xsd:whiteSpace value="collapse"/>`	Białe znaki. Wartość *collapse* oznacza zwinięcie wszystkich sekwencji spacji do jednej. Wartość *preserve* zachowuje wszystkie białe znaki, natomiast wartość *replace* oznacza zastąpienie wszystkich tabulatorów, znaków *line feed* i powrotu karetki pojedynczą spacją. Dotyczy wszystkich typów.
`<xsd:length value="10"/>`	Długość wpisywanego tekstu. Uwaga: każdy element powinien zawierać tekst dokładnie takiej długości, Dotyczy typów tekstowych.
`<xsd:minLength value="5"/>`	Minimalny rozmiar tekstu. Dotyczy typów tekstowych.
`<xsd:maxLength value="20"/>`	Maksymalna długość tekstu. Dotyczy typów tekstowych.

Ćwiczenie 4.6.

Dla elementów obszar *oraz* ludność *w pliku sąsiedzi.xml określ minimalną wartość jako 1.*

Ćwiczenie 4.7.

Utwórz element kwota i *określ, że powinien on zawierać dwa miejsca po przecinku oraz element Pesel i określ, że ma on zawierać dokładnie 11 cyfr.*

Wyliczenia

Oprócz wymienionych w tabeli deklaracji, można stosować *wyliczenia* (ang. enumeration) w celu pokazania wartości dopuszczalnych. Oto krótki przykład, który pozwoli Ci od razu zrozumieć jego zasadę:

Listing 4.9.

```
<xsd:element name="TrzejBracia" maxOccurs="unbounded">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="Polak"/>
      <xsd:enumeration value="Czech"/>
      <xsd:enumeration value="Rus"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

W restrykcjach można również zawrzeć deklarację wzorca (ang. *pattern*) poznanego trochę wcześniej.

Ćwiczenie 4.8.

Utwórz element będący wyliczeniem województw.

Listy

Kolejną konstrukcją dostępną dla schematów są *listy* wartości, które tworzą typ złożony z kilku typów prostych. Na przykład w elemencie *urodzinyDzieci* mają się znaleźć daty urodzin wszystkich dzieci danej osoby. Popatrzmy jak wygląda deklaracja takiej listy:

Listing 4.10.

```
<xsd:element name="urodzinyDzieci" type="xsd:string">
  <xsd:simpleType>
    <xsd:list itemType="xsd:date">
      <xsd:simpleType>
        <xsd:restriction base="xsd:positiveInteger">
          <xsd:maxLength value="20"/>
```

```
          <xsd:minLength  value="0"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:list>
  </xsd:simpleType>
</xsd:element>
```

Przeciętnym ludziom powinno wystarczyć 20 miejsc na wpisanie daty urodzin swoich dzieci. Kolejne wpisy oddzielamy spacjami i muszą one być datami. W dokumencie XML-a prawidłowo będą walidowane takie oto elementy:

```
<urodzinyDzieci>1997-12-31 1995-11-16</urodzinyDzieci>
<urodzinyDzieci>2001-02-12</urodzinyDzieci>
<urodzinyDzieci></urodzinyDzieci>
```

Ćwiczenie 4.9.

Zdefiniuj typ listowy, w którym wpisuje się wszystkie państwa graniczące z danym krajem.

Jeżeli trzeba będzie użyć nazw wieloczłonowych, takich jak Wielka Brytania czy Republika Południowej Afryki, należy wówczas zamiast spacji wpisać znak podkreślenia.

Unie pozwalają definiować typy proste, które mogą zawierać różnego rodzaju zawartość, na przykład, jeżeli chcemy zapisać rozmiar czcionki za pomocą liczb od 6 – 72 lub zastosować napisy *small*, *normal* lub *large*, to musimy skorzystać z następującej unii.

Listing 4.11.

```
<xsd:simpleType>
    <xsd:union>
      <xsd:simpleType>
        <xsd:restriction base="xsd:positiveInteger">
          <xsd:minInclusive value="8"/>
          <xsd:maxInclusive value="72"/>
        </xsd:restriction>
      </xsd:simpleType>
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="small"/>
          <xsd:enumeration value="normal"/>
          <xsd:enumeration value="large"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:union>
  </xsd:simpleType>
```

Teraz następujący fragment dokumentu XML-a będzie poprawnie interpretowany:

```
<fontSize>small</fontSize>
…
<fontSize>14</fontSize>
…
<fontSize>large</fontSize>
```

Ćwiczenie 4.10.

Zadeklaruj unię dla elementu, który będzie zawierał datę lub tekst „nieznana”. Taki typ można wykorzystać we wszystkich elementach, które do tej pory były zwykłymi datami.

Rozszerzenia typów

Konstrukcje dotyczące typów złożonych pozwalają na rozbudowywanie typów według własnych potrzeb, możemy na przykład rozszerzyć definicję istniejącego typu za pomocą konstrukcji `extension`. Zobaczmy w jaki sposób zadeklarować takie rozszerzenie:

Listing 4.12.

```
<xsd:complexType name="państwo">
  <xsd:sequence>
    <xsd:element name="stolica" type="xsd:string"/>
    <xsd:element name="obszar" type="xsd:positiveInteger"/>
    <xsd:element name="ludność" type="xsd:positiveInteger"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="państwa_info">
 <xsd:complexContent>
  <xsd:extension base="państwo">
    <xsd:sequence>
        <xsd:element name="glowne_miasta" type="xsd:string" maxOccurs="10"/>
        <xsd:element name="glowne_rzeki" type="xsd:string" maxOccurs="10"/>
     </xsd:sequence>
  </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

Poszerzamy definicję typu *państwo* dodając do niego nowe elementy: *glowne_miasta* oraz *glowne_rzeki*. Zauważ również, że poszerzany typ musi mieć złożoną zawartość `<xsd:complexContent>`, składającą się z bazowego typu oraz wyspecyfikowanych struktur. Popatrz teraz, w jaki sposób odwołać się do tego typu:

Listing 4.13.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xsd:element name="świat">
    <xsd:complexType>
      <xsd:sequence>
    <xsd:element name="państwa">
       <xsd:complexType>
        <xsd:sequence>
      <xsd:element name="państwo" type="państwa_info"/>
        </xsd:sequence>
      </xsd:complexType>
```

```
    </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

Grupy

Kolejną konstrukcją, którą możemy zadeklarować jest *grupa* powtarzających się wielokrotnie w schemacie elementów. Umożliwia nam to jednorazowe utworzenie takiej grupy i wielokrotne jej wykorzystanie. Definicja grupy *cechy_fizyczne* będzie się odnosić zarówno do elementu głównego *świat*, jak i do elementów potomnych *kontynent* i *państwo*.

Listing 4.14.

```
<xsd:group name="cechy_fizyczne">
<xsd:sequence>
  <xsd:element name="powierzchnia" type="xsd:positiveInteger"/>
  <xsd:element name="ludność" type="xsd:positiveInteger"/>
</xsd:sequence>
</xsd:group>
```

Musimy odpowiednio odwołać się do grupy przy definiowaniu struktur złożonych:

Listing 4.15.

```
<xsd:element name="świat">
<xsd:complexType>
      <xsd:sequence>
      <xsd:group ref="cechy_fizyczne"/>
...
<xsd:element name="kontynent" maxOccurs="7">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:group ref="cechy_fizyczne"/>
...
<xsd:complexType name="państwo">
  <xsd:sequence>
    <xsd:element name="stolica" type="xsd:string"/>
    <xsd:group ref="cechy_fizyczne"/>
  </xsd:sequence>
</xsd:complexType>
```

Poprawnie walidowany dokument teraz może wyglądać następująco:

Listing 4.16.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.0 NT beta 2 build Jul 24 2001 (http://www.xmlspy.com) by
WR (private) -->
```

```
<świat xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\Program Files\Altova\XML Spy
Suite\Examples\świat.xsd">
  <powierzchnia>44000000</powierzchnia>
  <ludność>6000000000</ludność>
  <kontynent>
    <powierzchnia></powierzchnia>
    <ludność></ludność>
    <państwa>
      <państwo>
        <stolica>Warszawa</stolica>
        <powierzchnia>316000</powierzchnia>
        <ludność>40000000</ludność>
        <glowne_miasta>Katowice</glowne_miasta>
        <glowne_miasta>Łódź</glowne_miasta>
        <glowne_miasta>Poznań</glowne_miasta>
        <glowne_miasta>Wrocław</glowne_miasta>
        <glowne_miasta>Kraków</glowne_miasta>
        <glowne_miasta>Szczecin</glowne_miasta>
        <glowne_rzeki>Wisła</glowne_rzeki>
        <glowne_rzeki>Odra</glowne_rzeki>
        <glowne_rzeki>Warta</glowne_rzeki>
      </państwo>
    </państwa>
  </kontynent>
</świat>
```

Ćwiczenie 4.11.

Utwórz grupę elementów dla opisu człowieka. Powinny tam się znaleźć pozycje takie jak: wzrost, waga, kolor włosów itd.

Podstawianie grup

Ciekawym rozwiązaniem jest możliwość podstawiania grup. W ten sposób możemy na przykład wyspecyfikować kilka wersji językowych. Możemy również zdefiniować grupę ze zlokalizowanymi ustawieniami i wykorzystać odpowiedni schemat w zależności na przykład od jakiegoś atrybutu. Przypuśćmy, że mamy dokument, w którym chcemy wpisywać środki lokomocji w Polsce i w krajach anglojęzycznych. Dla ułatwienia: polskie środki lokomocji będzie się wpisywało za pomocą polsko brzmiących znaczników. Natomiast użytkownicy zagraniczni będą mogli skorzystać z wersji angielskiej. Do tej pory musiałbyś tworzyć dwa typy dokumentów, teraz możesz skorzystać z podstawiania grup.

Listing 4.17.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xsd:element name="pociąg" type="xsd:string"/>
    <xsd:element name="train" type="xsd:string" substitutionGroup="pociąg"/>
  <xsd:complexType name="lokomocja">
    <xsd:sequence maxOccurs="unbounded">
```

```
      <xsd:element ref="pociąg"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="komunikacja" type="lokomocja"/>
    <xsd:element name="transportation" substitutionGroup="komunikacja"/>
</xsd:schema>
```

Dokument XML w polskiej wersji językowej będzie poprawnie walidowany według tego schematu. Również poniższy przykład w wersji angielskiej nie będzie sprawiał trudności.

Listing 4.18.

```
<-- WERSJA POLSKA -->
  <?xml version="1.0" encoding="UTF-8"?>
<komunikacja xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\mojeXML\Examples\lokomocja.xsd">
  <pociąg>Katowice-Szczecin</pociąg>
</komunikacja>

  <-- WERSJA ANGIELSKA -->
  <?xml version="1.0" encoding="UTF-8"?>
<transportation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation=" C:\mojeXML\Examples\lokomocja.xsd">
  <train>London-Manchester</train>
</transportation>
```

Przedstawione tutaj konstrukcje świadczą o wielu zaletach języka XML Schema, za pomocą którego można dokładnie określić kształt elementów, można je grupować, tworzyć własne typy zawartości albo nawet tworzyć różne wersje językowe dokumentów. Do tej pory jednak nie poznaliśmy konstrukcji zajmujących się tworzeniem atrybutów dla elementów. To zagadnienie zostanie omówione po kolejnym ćwiczeniu.

Ćwiczenie 4.12.

Napisz schemat do dokumentu XML-a opisującego zbiór twoich kaset Video, płyt CD lub książek. Pola, których użyjesz, są dowolne, pamiętaj jednak o dokładnym określeniu zawartości elementów. Wszystkie typy znajdziesz w dodatku A. Tam, gdzie to będzie potrzebne, określ odpowiednie restrykcje dla wartości elementów.

Tworzenie atrybutów

Po zapoznaniu się z konstrukcjami dotyczącymi elementów zajmiemy się deklarowaniem atrybutów. Przy omawianiu DTD zauważyłeś zapewne, jak ważną rolę pełnią one w językach wywodzących się z XML-a, dlatego w specyfikacji XML-a Schema nie mogło ich zabraknąć. Atrybuty można deklarować jako pojedyncze, wstawiając je do klamry elementu, dla którego chcemy, aby należały, lub też jako samodzielne grupy przypisywane, w zależności od potrzeb, do odpowiednich elementów. Każdy element musi być typu złożonego, nawet jeżeli element ten zawiera tylko tekst lub jest elementem pustym.

Definiowanie atrybutu lokalnego odbywa się za pomocą znacznika <xsd:attribute>, wewnątrz którego podajemy dwie informacje: nazwę atrybutu (*name*) oraz jego typ (*type*).

Listing 4.19.

```
<xsd:element name="igrzyskaOlimpijskie" id="nazwa">
  <xsd:complexType>
. . .
    <xsd:attribute name="miasto" type="xsd:string"/>
    <xsd:attribute name="rok" type="xsd: xsd:gYear"/>
. . .
  </xsd:complexType>
</xsd:element>
```

Dodatkowo dla atrybutu możemy określić informacje o jego sposobie występowania (parametr `use`), wartościach domyślnych (parametr `default`) oraz możemy zadeklarować atrybut jako stały (parametr `default`).

Listing 4.20.

```
<xsd:attribute name="funkcja" type="xsd:string" use="required" default="Gamma
Eulera"/>
<xsd:attribute name="zdefiniowana_przez" type="xsd:string" use="optional"
default="Euler"/>
         <xsd:attribute name="PI" fixed="3.14159"/>
         <xsd:attribute name="pi" use="prohibited"/>
         <xsd:attribute name="Pi" use="prohibited"/>
         <xsd:attribute name="pI" use="prohibited"/>
```

Parametr *use* może przyjmować trzy wartości: `required` — wtedy ten atrybut jest wymagany i nie można go pominąć podczas pisania elementu; `optional` — atrybut jest opcjonalny oraz `prohibited` — nie można użyć atrybutu o tej nazwie. W przykładzie zadeklarowano, że tylko jedna nazwa argumentu PI jest poprawna i nadano temu atrybutowi wartość stałą.

Natomiast jeżeli zestaw argumentów będzie potrzebny dla wielu elementów, wygodniej go zdefiniować jako grupę, a następnie za pomocą odnośnika wskazać go w konkretnych elementach złożonych. Zdefiniujmy grupę atrybutów dla wstawianych obrazków; elementy zawierające takie informacje mogą występować w różnych miejscach dokumentu i pod różnymi nazwami, ale argumenty będą miały takie same.

Listing 4.21.

```
<xsd:attributeGroup name="atrybutyObrazków">
         <xsd:attribute name="nazwaPliku" type="xsd:anyURI" use="required"/>
         <xsd:attribute name="alternatywnyTekst" type="xsd:string" use="optional"/>
         <xsd:attribute name="wysokość" type="xsd:integer" use="optional"/>
         <xsd:attribute name="szerokość" type="xsd:integer" use="optional"/>
</xsd:attributeGroup>
```

Odwołania w elementach wyglądają następująco:

Listing 4.22.

```
. . .
<xsd:element name="zdjęcieZawodnika">
  <xsd:complexType>
    <xsd:attributeGroup ref="atrybutyObrazków"/>
  </xsd:complexType>
</xsd:element>
. . .
<xsd:element name="zdjęcieMiasta">
  <xsd:complexType>
    <xsd:attributeGroup ref="atrybutyObrazków"/>
  </xsd:complexType>
</xsd:element>
. . .
```

Ćwiczenie 4.13.

Dodaj atrybuty do elementów z poprzedniego ćwiczenia. Zastanów się, gdzie używać atrybutów, a gdzie oprzeć się na kolejnym elemencie.

Ćwiczenie 4.14.

Zadeklaruj grupę atrybutów, w której znajdą się `rocznikStatystyczny`, `nazwaPaństwa`, *oraz* `obszar Igrzysk Olimpijskich`. *Nazwij ją państwaInfo. Pierwszy z atrybutów powinien być rokiem, drugi tekstem, a trzeci liczbą całkowitą nieujemną.*

Listing 4.23.

```
<xsd:attributeGroup name="państwaInfo">
   <xsd:attribute name="rocznikStatystyczny" type="xsd:gYear" use="required"/>
   <xsd:attribute name="nazwaPaństwa" type="xsd:string" use="required"/>
   <xsd:attribute name="obszar" type="xsd:positiveInteger" use="required"/>
   </xsd:attributeGroup>
```

Ćwiczenie 4.15.

Utwórz schemat dla elementu państwo *z zawartością złożoną, następnie dołącz do tego elementu grupę atrybutów zdefiniowanych powyżej oraz dodaj dodatkowe elementy potomne o nazwach główne_miasta* oraz *główne_rzeki. Elementy potomne są typu prostego, występują wielokrotnie i zawierają tekst.*

Listing 4.24.

```
<xsd:element name="państwo">
    <xsd:complexType>
      <xsd:sequence>
<xsd:element name="główne_miasta" type="xsd:string" maxOccurs="unbounded"/>
<xsd:element name="główne_rzeki" type="xsd:string" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attributeGroup ref="państwaInfo"/>
    </xsd:complexType>
  </xsd:element>
```

Przestrzenie nazw

Do tej pory traktowaliśmy nazwy elementów tak, jakby nie miały one prawa się powtórzyć. Co się jednak stanie gdy pobierzemy z Internetu inny schemat, w którym występuje nazwa elementu deklarowana przez nas w innym schemacie? Tu właśnie stajemy przed problemem: jak połączyć ze sobą dwa schematy, w których występuje ta sama nazwa elementu mającego różne zadania i strukturę w obu schematach. W celu jednoznacznej identyfikacji elementu stosuje się dodatkowy prefiks, który będzie kojarzył konkretny element ze schematem. Na przykład element *konkurencja* ze schematu *igrzyskaOlimpijskie.xsd* może wystąpić gdzie indziej. Można by mu było nadać prefiks *olimpiada*, ale nie jest to zbyt oryginalny pomysł, gdyż równie łatwo może wpaść na niego ktoś inny. Dlatego w praktyce stosuje się jako nazwy prefiksów adresy URL naszej domeny. W ten sposób zapewniamy niepowtarzalność i niezmienność nazw. Dodatkowy prefiks elementu nazywa się właśnie *przestrzenią nazw*. Jej zastosowanie pozwala spać spokojnie twórcom języków bazujących na XML-u. Trzeba jeszcze powiedzieć, że rzadko się zdarza, aby elementy wewnętrzne, czyli lokalne, potrzebowały kwalifikowania przez przestrzeń nazw, gdyż wystarczy, aby ich rodzic znajdował się w odpowiedniej przestrzeni. Jednak mimo to przedstawię taką konstrukcję i powiem, kiedy ona jest potrzebna.

Problem zapewnienia jednoznaczności przestrzeni nazw w obrębie jednej domeny rozwiązuje się za pomocą dodania do adresu URL naszej domeny dodatkowych informacji oddzielonych ukośnikami „/”. Można to zrobić, gdyż ten powstały adres URL nie musi, a nawet nie powinien wskazywać konkretnego dokumentu w naszych zasobach sieciowych. Dlatego poniższy przykład nazwy przestrzeni jest jak najbardziej prawidłowy:

```
http://www.mojadomena.pl/xml/schematy/WRomowicz/olimpiady/1.0
```

Deklaracje w dokumentach XML-a można podzielić na domyślne przestrzenie nazw oraz przestrzenie nazw dla poszczególnych elementów. *Domyślna przestrzeń nazw* jest definiowana w elemencie głównym lub tym, dla którego jego potomkowie mają się znaleźć w tej przestrzeni. Przestrzeń definiuje się dodając parametr *xmlns* (*XML NameSpace*), po którym wpisujemy jej nazwę. Zobacz przykłady:

Listing 4.25.

```
<!-- Deklaracja przestrzeni nazw dla elementu głównego -->
<igrzyskaOLimpijskie
  xmlns="http://www.mojadomena.pl/xml/WRomowicz/olimpiady/1.0">
<dyscyplina nazwa="Lekka Atletyka">
...
</igrzyskaOlimpijskie>

<!-- Deklaracja przestrzeni nazw dla potomka elementu głównego -->
<mójŚwiat>
  <państwa xmlns="http://www.mojadomena.pl/xml/WRomowicz/kraje/1.0">
    ...
  </państwa>
</mójŚwiat>
```

Można również zadeklarować przestrzeń nazw dla konkretnego elementu bez wpływania na elementy potomne. W tym celu musimy jeszcze do deklaracji elementu dołożyć specjalny *przedrostek*, który będzie kojarzony z tym elementem. Przedrostek ten musimy również użyć przed nazwą elementu. Przedrostek od nazwy elementu i deklaracji *xmlns* oddzielamy znakiem dwukropka.

Listing 4.26.

```
<!-- Deklaracja przestrzeni nazw dla potomka elementu głównego -->
<mójŚwiat>
  <państwa xmlns="http://www.mojadomena.pl/xml/WRomowicz/kraje/1.0">
    <państwo nazwa=Polska">
    <stolica>Warszawa</stolica>
    <obszar>312680</obszar>
    <ludność>38643000</ludność>
<wody:rzeki xmlns:wody="http://www.mojadomena.pl/xml/WRomowicz/wody/1.0">
  <wody:rzeka>Wisła</wody:rzeka>
</wody:rzeki>
    </państwo>
  </państwa>
</mójŚwiat>
```

Wszystkie inne elementy, które nie są oznakowane przedrostkiem *wody* należą do domyślnej przestrzeni nazw. Widzisz zatem, że w dokumencie elementy mogą należeć do wielu przestrzeni i jest to zachowanie jak najbardziej prawidłowe.

Atrybutów nie trzeba przypisywać do żadnej przestrzeni nazw, gdyż są one nierozerwalnie związane z elementami.

Teraz przyjrzymy się sposobowi deklaracji elementów w przestrzeniach nazw. Aby określić przestrzeń nazw, należy zamiast parametru `xsi:noNamespaceSchemaLocation` podać parametr `targetNamespace` z nazwą przestrzeni.

Listing 4.27.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.mojadomena.pl/xml/WRomowicz/tr/1.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="transportLądowy">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="ciężarówki"/>
        …
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="transportMorski">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="statki"/>
        …
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

W tym przykładzie tylko elementy definiowane globalnie będą przypisane do przestrzeni nazw, natomiast jeżeli chcemy przypisać tę przestrzeń do wszystkich elementów, musimy dopisać jeszcze parametr `elementFormDefault="qualified"`.

Listing 4.28.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.mojadomena.pl/xml/WRomowicz/tr/1.0"
elementFormDefault="qualified"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="transportLądowy">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="ciężarówki"/>
        …
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="transportMorski">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="statki"/>
        …
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Jeżeli chcemy, aby jakiś element przy poprzedniej deklaracji nie należał do domyślnej przestrzeni nazw, należy ustawić dla niego parametr `form` na wartość `unqualified`.

Listing 4.29.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.mojadomena.pl/xml/WRomowicz/tr/1.0"
elementFormDefault="qualified"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="transportLądowy">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="ciężarówki"/>
        …
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="transportMorski">
    <xsd:complexType>
      <xsd:sequence>
         <xsd:element name="statki" form="unqualified"/>
        …
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Warto również wspomnieć, że jedną przestrzeń nazw już poznałeś, nie wiedząc o tym. Jest to przestrzeń z przedrostkiem *xsd*. Definiuje ona sam schemat. Zauważ, że wszystkie deklaracje w schemacie są opisywane właśnie takim przedrostkiem. Zmieniając trochę definicję nagłówka schematu, możemy go zadeklarować jako domyślny i nie używać przedrostka.

Listing 4.30.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.mojadomena.pl/xml/WRomowicz/tr/1.0"
elementFormDefault="qualified"
xmlns="http://www.w3.org/2001/XMLSchema">

<element name="transportLądowy">
    <complexType>
      <sequence>
        <element name="ciężarówki"/>
        …
      </sequence>
    </complexType>
  </element>
</schema>
```

Jeżeli chcemy nadać przestrzeni nazw nagłówek, to musimy to również zadeklarować w schemacie za pomocą konstrukcji:

Listing 4.31.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.mojadomena.pl/xml/WRomowicz/tr/1.0"
xmlns:tr="http://www.mojadomena.pl/xml/WRomowicz/tr/1.0"
xmlns="http://www.w3.org/2001/XMLSchema">
```

Nazw elementów w schemacie nie trzeba wówczas poprzedzać przedrostkiem, gdyż wskazuje na to parametr `targetNamespace`. Natomiast przedrostek jest wymagany przy wartościach atrybutów wywoływanych parametrem `ref` oraz przy deklarowaniu typu — parametr `type`.

Listing 4.32.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://www.mojadomena.pl/xml/WRomowicz/tr/1.0"
xmlns:tr="http://www.mojadomena.pl/xml/WRomowicz/tr/1.0"
xmlns="http://www.w3.org/2001/XMLSchema">

<element name="transportLądowy">
    <complexType>
      <sequence>
        <element name="ciężarówki type="tr:auto"/>
        …
      </sequence>
    </complexType>
  </element>
</schema>
```

Natomiast w dokumencie XML-a konieczne jest deklarowanie przedrostka, tak jak to pokazano wcześniej.

Jeszcze jedna sprawa pozostała nie omówiona, jak zadeklarować w dokumencie XML-a przypisanie przestrzeni nazw do schematu. Do tej pory omawialiśmy schematy, które nie miały przestrzeni nazw, więc używaliśmy parametru `xsi:noNamespaceSchemaLocation`, obecnie musimy zastąpić go parametrem `xsi:schemaLocation`. Popatrz jak zmodyfikować postać nagłówka pliku XML:

Listing 4.33.

```
<?xml version="1.0" encoding="UTF-8"?>
<io:igrzyskaOlimpijskie rok="2000" miasto="Sydney"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="
http://www.mojadomena.pl/xml/WRomowicz/olimpiady/1.0
http://www.mojadomena.pl/xml/Olimpiady.xsd">
```

Parametr `schemaLocation` ma specjalną formę: `xsi:schemaLocation="nazwaPrzestrzeni URL_pliku_xsd"`.

Ćwiczenie 4.16.

Utwórz przestrzeń nazw z prefiksem sąsiad. Będzie ona służyła do opisywania elementów z pliku sąsiedzi.xml (dodatek D). Następnie zbuduj całość schematu i odpowiednio zmodyfikuj plik sąsiedzi.xml, tak by obsługiwał nowo zdefiniowaną przestrzeń nazw.

Ćwiczenie 4.17.

Utwórz przestrzeń nazw z prefiksem Igrzyska. Będzie ona służyła do opisywania elementów z plików dotyczących Olimpiad ery nowożytnej (plik Sydney2000.xml — dodatek D). Następnie zbuduj całość schematu i odpowiednio zmodyfikuj plik Sydney2000.xml, tak by obsługiwał nowo zdefiniowaną przestrzeń nazw.

Zaawansowane techniki używane w schematach

Schemat może być plikiem bardzo dużym, dlatego warto go podzielić na kilka mniejszych fragmentów, tak by łatwiej nam było go później edytować. Korzystanie z możliwości dzielenia plików schematu daje nam deklaracja `xsd:include`. Jako wartość parametru `schemaLocation` podajemy położenie pliku z wybranym fragmentem.

Listing 4.34.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.mojadomena.pl/xml/WRomowicz/tr/1.0"
elementFormDefault="qualified"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
...
<xsd:include schemaLocation="http://www.mojadomena.pl/xml/tr/ladowy.xsd"/>
<xsd:include schemaLocation="http://www.mojadomena.pl/xml/tr/morski.xsd"/>
...
</xsd:schema>
```

Import umożliwia pobranie z innych przestrzeni nazw elementów globalnych i zastosowanie ich w schemacie importującym. Poniżej znajdziesz przykład takiej konstrukcji.

Listing 4.35.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema targetNamespace="http://www.mojadomena.pl/xml/WRomowicz/kraje/1.0"
elementFormDefault="qualified"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wody="http://www.mojadomena.pl/xml/WRomowicz/wody/1.0">

<xsd:import namespace= "http://www.mojadomena.pl/xml/WRomowicz/wody/1.0"
xsi:schemaLocation="http://www.mojadomena.pl/xml/WRomowicz/wody/1.0
http://www.mojadomena.pl/xml/wody.xsd"/>
...
  <xsd:element name="rzeki" …
</xsd:schema>
```

Importowanie i załączanie pomaga w pracy z dużymi plikami schematów, a także pomaga w rozjaśnieniu składni takiego schematu.

Klucze w schematach

Wprawdzie omówienie kluczy i wartości unikatowych powinno się znaleźć wcześniej, ale ze względu na to, że ogólnodostępne walidatory nie do końca poprawnie interpretują te konstrukcje, umieszczam ich omówienie w części dotyczącej zaawansowanych zagadnień. Założenia są takie: deklarujemy wartość unikatową, aby zapewnić, że żadna wartość elementu nie może się powtórzyć. Posługujemy się w tym celu konstrukcją:

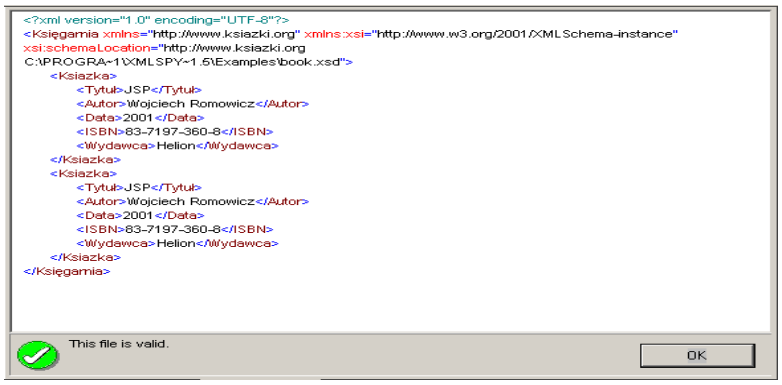
Listing 4.36.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                    targetNamespace="http://www.ksiazki.org"
                    xmlns="http://www.ksiazki.org"
                    xmlns:ks="http://www.ksiazki.org"
                    elementFormDefault="qualified">
```

```
<xsd:element name="Księgarnia">
  <xsd:complexType>
     <xsd:sequence>
    <xsd:element name="Ksiazka" maxOccurs="unbounded">
    <xsd:complexType>
        <xsd:sequence>
      <xsd:element name="Tytuł" type="xsd:string"/>
      <xsd:element name="Autor" type="xsd:string"/>
      <xsd:element name="Data" type="xsd:string"/>
      <xsd:element name="ISBN" type="xsd:string"/>
      <xsd:element name="Wydawca" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    </xsd:element>
      </xsd:sequence>
  </xsd:complexType>
  <xsd:unique name="unikatowyISBN">
    <xsd:selector xpath="ks:Ksiazka"/>
    <xsd:field xpath="ks:ISBN"/>
  </xsd:unique>
</xsd:element>
</xsd:schema>
```

Tak jak powiedziałem, deklaracja samej konstrukcji jest poprawna i w specyfikacji opisano ją jako mającą gwarantować niepowtarzalność argumentów. Na przykład XML Spy traktuje taki dokument jako poprawny, a nie powinien, ponieważ numery ISBN się powtarzają. Jest to wersja programu XML Spy 4.0 Beta 2 i może w wersji końcowej zostaną wprowadzone te elementy do standardu.

Rysunek 4.1. *Poprawnie interpretowany dokument*



Podobnie ma się sprawa z kluczami i ich odnośnikami. Klucz jest rozszerzeniem wartości unikatowej. Oprócz tego, że element taki musi być niepowtarzalny, wymaga się również, aby jego zawartość nie była łańcuchem pustym (argument `nillable` musi być ustawiony na *false*) oraz musi on wystąpić (`parametr maxOccurs` musi być większy od zera). Znaczenie klucza jest bardzo podobne do określenia klucza dla tabeli w bazie danych.

Oto zmiany, jakie należy wprowadzić w powyższym przykładzie, by zadeklarować klucz:

Listing 4.37.

```
...
<xsd:element name="ISBN" nillable="0" minOccurs="1"/>
...
<!-- Zamiast sekcji unique -->
<xsd:key name="kluczISBN">
  <xsd:selector xpath="ks:Ksiazka"/>
  <xsd:field xpath="ks:ISBN"/>
</xsd:unique>
```

Więcej o wybieraniu elementów za pomocą selektorów i pól dowiemy się w następnym rozdziale poświęconym specyfikacji XPath. Należy tutaj jedynie nadmienić, że wybieramy element *Ksiazka*, a z niego pole ISBN, które ma być kluczem w dokumencie.