

## Definiowanie typów dokumentów

### Część 4. XML Schema, RELAX NG, Schematron

## Symbole wieloznaczne w XML Schema

- Symbole wieloznaczne dla elementów (*element wildcards*).
- Symbole wieloznaczne dla atrybutów (*attribute wildcards*).

```
<xsd:complexType name="OsobaTyp">
  <xsd:sequence>
    <xsd:element name="imie" type="xsd:string"/>
    <xsd:element name="nazwisko" type="xsd:string"/>
    <xsd:any namespace="##other" minOccurs="0"
      maxOccurs="unbounded"
      processContents="skip"/>
  </xsd:sequence>
  <xsd:anyAttribute namespace="##other"
    processContents="lax"/>
</xsd:complexType>
```

## Symbole wieloznaczne – typowe zastosowanie

```
<xsd:element name="description">
  <xsd:complexType mixed="true">
    <xsd:sequence>
      <xsd:any
        namespace="http://www.w3.org/1999/xhtml"
        minOccurs="0" maxOccurs="unbounded"
        processContents="skip"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

## Czego nie można zamodelować w XML Schema? (1/2)

- Brak kontekstowego sprawdzania poprawności, np.:
  - zawartość elementu `cena-netto` jest mniejsza lub równa od zawartości elementu `cena-brutto`,
  - jeżeli wartość atrybutu `sposob-transportu` jest `powietrze`, to element `šrodek-transportu` ma zawartość `samołot` lub `balon`.
- Metody kontekstowego sprawdzania poprawności :
  - zaprogramowane w kodzie aplikacji,
  - transformacja XSLT,
  - wykorzystanie innego języka schematów, np. Schematron.

## Czego nie można zamodelować w XML Schema? (2/2)

- Niejednoznaczność (*ambiguity*): egzemplarz jest poprawny względem kilku wzorców, np.:  
(`nazwisko` | `nazwisko`)
- Niedeterminizm (*non-determinism*): procesor ma do wyboru wiele pasujących wzorców (produkcji gramatyki), np.:  
(`nazwisko` | (`nazwisko`, `imie`))
- Równoważny model deterministyczny (nie zawsze istnieje):  
(`nazwisko`, `imie`?)

## Schematron

- Język oparty na własnościach (asercjach), a nie na gramatyce:
  - łatwe wyrażanie reguł walidacji kontekstowej,
  - trudne, nieintuicyjne modelowanie struktury dokumentu,
  - użyteczny w połączeniu ze zwykłą DTD lub schematem XML Schema.
- Status – dwie wersje:
  - The Schematron Assertion Language 1.5,
  - Final Committee Draft of ISO Schematron (ISO/IEC 19757-3).
- Implementacja referencyjna:
  - przekształcenie (generator) XSLT,
  - dla danego schematu Schematronowego, generuje XSLT sprawdzający poprawność dokumentów.
- Dostępnych kilkanaście implementacji.

## Język Schematron

- Własności ewaluowane w kontekście konkretnego węzła dokumentu:
  - `assert` – własność, która musi być spełniona,
  - `report` – własność, której spełnienie oznacza błąd.
- Określanie kontekstu i własności: wyrażenia XPath.
- Przykład:

```
<rule context="towar">
  <assert test="@wartosc = @cena * @liczba">
    wartosc = cena * liczba</assert>
</rule>
<rule context="faktura">
  <report test="@platnosc != 'przelew' and ./przelew">
    Przelew określony, a nie płacimy przelewem
  </report>
</rule>
```

Źródło: Czarnik, P., *DTD, XML Schema – i co dalej?*, Software 2.0, nr 6/2003

## Schematron + XML Schema (1/2)

```
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.demo.org"
  xmlns:sch="http://www.demo.org"
  xmlns:sch="http://www.ascc.net/xml/schematron"
  elementFormDefault="qualified">
  <xsd:annotation>
    <xsd:appinfo>
      <sch:title>Schematron validation</sch:title>
      <sch:ns prefix="d" uri="http://www.demo.org"/>
    </xsd:appinfo>
  </xsd:annotation>
  ...
```

Źródło:  
*Extending XML Schemas. A Collectively Developed  
Set of Schema Design Guidelines*  
<http://www.xfront.com/ExtendingSchemas.html>

## Schematron + XML Schema (2/2)

```
<xsd:element name="demo">
  <xsd:annotation>
    <xsd:appinfo>
      <sch:pattern name="Check A greater than B">
        <sch:rule context="d:Demo">
          <sch:assert test="d:A > d:B"> A should be
            greater than B. </sch:assert>
        </sch:rule>
      </sch:pattern>
    </xsd:appinfo>
  </xsd:annotation>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="A" type="xsd:integer"/>
      <xsd:element name="B" type="xsd:integer"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Źródło:  
*Extending XML Schemas. A Collectively Developed  
Set of Schema Design Guidelines*  
<http://www.xfront.com/ExtendingSchemas.html>

## RELAX NG

- REgular LAnguage description for XML – New Generation:
  - składnia XML-owa, „bliska opisowi struktury w języku naturalnym”,
  - wspiera typy danych (np. XML Schema Datatypes),
  - atrybuty opisywane (prawie) tak samo, jak elementy,
  - prosta technika modularyzacji: `define / ref`,
  - model przetwarzania oparty na wyrażeniach regularnych.
- RELAX NG a inne języki:
  - dostępne konstrukcje nie występujące w XML DTD:
    - elementy wymagane, ale bez określonego porządku,
    - model mieszany – więcej możliwości;
  - pozwala opisać niedostępne w XML Schema:
    - niejednoznaczne i niedeterministyczne modele zawartości,
    - modele zawartości wzajemnie zależne.

## Przykład

- DTD:

```
<!DOCTYPE addressBook [
  <!ELEMENT addressBook (card*)>
  <!ELEMENT card (name, email)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT email (#PCDATA)>
]>
```
- RELAX NG:

```
<element name="addressBook"
  xmlns="http://relaxng.org/ns/structure/1.0">
  <zeroOrMore>
    <element name="card">
      <element name="name"> <text/> </element>
      <element name="email"> <text/> </element>
    </element>
  </zeroOrMore>
</element>
```

Źródło: *RELAX NG Tutorial*, <http://www.relaxng.org/tutorial-20011203.html>

## Przykład

- Konstrukcja zabroniona w XML Schema:

```
<element name="name">
  <choice>
    <text/>
    <group>
      <element name="firstName"><data type="token"/>
    </element>
    <optional>
      <element name="middleName"><data type="token"/>
    </optional>
    <optional>
      <element name="lastName"><data type="token"/>
    </optional>
    </group>
  </choice>
</name>
```

Źródło: *RELAX NG Tutorial*, <http://www.relaxng.org/tutorial-20011203.html>

## Przykład – niedeterminizm

`(odd, even)*, odd?`

- Model jednoznaczny, ale niedeterministyczny.
- Nie istnieje równoważny model deterministyczny.
- Nie da się zapisać w XML Schema.

```
<element name="even-odd">
  <zeroOrMore>
    <ref name="odd"/>
    <ref name="even"/>
  </zeroOrMore>
  <optional>
    <ref name="odd"/>
  </optional>
</element>
```

Źródło: Vlist, E. van der, *RELAX NG*, <http://books.xmlschemata.org/relaxng>

## Examplotron

- Definiowanie schematu „przez przykład”:
  - egzemplarz dokumentu definiuje schemat,
  - konwencje, np.:
    - powtórzenie elementu oznacza dowolną krotność,
    - przykładowa zawartość elementu definiuje typ.
- Ograniczenia:
  - „przez przykład” nie można wyrazić konstrukcji abstrakcyjnych,
  - dodatkowa, specyficzna składnia pozwala na dokładniejszą kontrolę struktury.
- Status:
  - projekt na wczesnym etapie rozwoju (wersja 0.7),
  - dostępne narzędzie: `compile.xsl` – przetłumacz schematu Examplotronowe do RELAX NG.

## Examplotron – przykład

```
<?xml version="1.0" encoding="UTF-8"?>
<order
  date="2003-02-01"
  eg:content="eg:interleave"
  xmlns:eg="http://examplotron.org/0/">
  <eg:attribute name="no">1234
</eg:attribute>
  <quantity>1</quantity>
  <ref>AZERTY</ref>
  <ref>ZXCVCBN</ref>
  <item>Tee shirt</item>
  <price unit="USD">10.</price>
</order>
```

atribut opcjonalny  
typu data

element powtarzalny

atribut wymagany  
typu liczba

dowolna kolejność  
podelementów

## Zarządzanie zmianami struktury

- Zmiany niekompatybilne wstecz – przykład:
  - dodanie elementu wymaganego.
- Sposób postępowania w „żywym” systemie:
  - wprowadzamy zmianę modelu kompatybilną wstecz (np. dodajemy element, ale opcjonalny),
  - migrujemy dokumenty:
    - przekształcamy automatycznie i/lub
    - instruujemy użytkowników o konieczności migracji do nowej struktury,
  - po dodaniu brakujących elementów (lub po upływie wyznaczonego czasu)
  - wprowadzenie zmiany docelowej.
- Większe zmiany modelu:
  - deklarujemy osobny element z nowym modelem i przez pewien czas dopuszczamy stary lub nowy model,
  - stosujemy przez pewien czas równoległe dwie wersje schematu.

## Zmiany struktury a aplikacje

- Typowa zależność między treścią programu a strukturą danych:
  - w treści programu zakładamy konkretną postać struktur danych,
  - jeśli są dane wejściowe lub wyjściowe, ich postać może się zmieniać w czasie,
  - zmiana struktury danych powoduje konieczność zmian w kodzie.
- Tworzenie aplikacji niezależnych od zmian struktury danych:
  - ignorowanie elementów i atrybutów, które nie są znaczące dla aplikacji,
  - unikanie nadmiernej zależności od struktury dokumentu,
  - parametryzowanie aplikacji dodatkowymi informacjami umieszczonymi w schemacie, np.:

```
<xsd:element name="NIP">
  <xsd:complexType>
    ...
    <xsd:attribute name="opis" type="xsd:string"
      fixed="Numer Identyfikacji Podatkowej"/>
  </xsd:complexType>
</xsd:element>
```

## Przestrzeń nazwy a aplikacje niezależne od struktury

- Przykład: XLink:
    - linki w elementach o dowolnych nazwach,
    - typ linku i jego parametry przekazywane przez specjalne atrybuty.
- ```
<osoba xmlns:xlink="http://www.w3.org/1999/xlink">
  <nazwisko>Kopernik, Mikołaj</nazwisko>
  <biogram>Wybitny polski astronom, urodzony w <geogr
    xlink:type="simple" xlink:href="Torun.xml">
    Toruniu</geogr>.</biogram>
</osoba>
```

## Przestrzenie nazw a aplikacje niezależne od struktury

- Przykład:
  - aplikacja weryfikująca numery PESEL i daty urodzenia w dokumencie XML,
  - nie powinna zależeć od struktury dokumentu wejściowego,
  - jak „przekazać” aplikacji, co ma zweryfikować?

### Rozwiązanie:

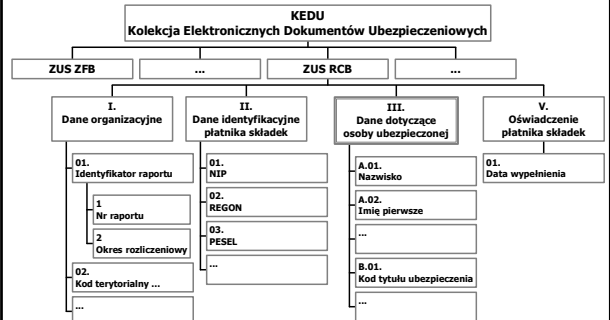
```
<podanie xmlns:pv="http://PeselValidator.pl">
  <nadawca nr-ewid="60101012345" pv:PESEL="nr-ewid">
    <nazwisko>Zenon Niemrawy</nazwisko>
    <urodzony pv:data-ur="#CONTENT">1960-10-10
  </urodzony>
  </nadawca>
  ...
</podanie>
```

## Case study XML jako format dokumentów ubezpieczeniowych ZUS

## Tło projektu

- Formularze ubezpieczeniowe:
  - 22 typy formularzy,
  - przysyłane okresowo przez płatników do ZUS,
  - dotychczas kodowane w pseudo-SGML-u.
- Przyczyny zmiany formatu:
  - błędny projekt formatu SGML-owego,
  - rosnąca popularność XML-a,
  - nadchodząca zmiana rozporządzenia określającego strukturę formularzy.
- Projekt badawczo-rozwojowy prowadzony przez empolis Polska w 2000 roku.

## Kolekcja Elektronicznych Dokumentów Ubezpieczeniowych



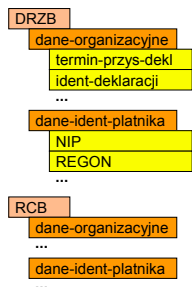
## Przykład: fragment formularza ZUS RCB

## Problemy

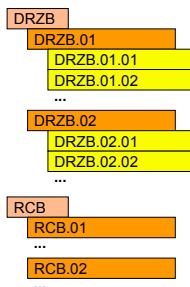
- Wybór logicznego modelu struktury dokumentów:
  - model semantyczny,
  - model składniowy.
- Modelowanie w DTD informacji pozwalających na sprawdzanie poprawności treści dokumentów.
- Modelowanie informacji zwrotnych:
  - informacje o błędach w dokumentach,
  - informacje o korektach automatycznie wprowadzonych przez ZUS.
- Oznaczenie pól wypełnianych przez ZUS.

## Logiczny model struktury dokumentów

Semantyczny:



Składniowy:



## Logiczny model struktury dokumentów

- Model semantyczny:
  - zwęższy i elegancki,
  - pozwala na modelowanie relacji wiele-do-wielu,
  - ale: nazwy szybko przestają być semantyczne.
- Model składniowy:
  - łatwość automatyzacji przetwarzania:
    - oprowanie nazwami elementów,
    - generowanie DTD oraz samych dokumentów,
  - możliwość wzbogacenia o informacje semantyczne.
- Wybór: model składniowy.

## Modelowanie informacji dodatkowych

- Informacje dodatkowe:
  - opisy pól,
  - informacje o sposobie walidacji pól,
  - informacje o polach wypełnianych przez ZUS.
- Sposób kodowania: atrybuty #FIXED:
  - umieszczane w DTD wraz z wartościami,
  - wartości dostępne w instancji dokumentu,
  - nie ma możliwości zmiany wartości atrybutu w instancji dokumentu.

## Informacje dodatkowe – przykład

```
<!ELEMENT DRSB.01.04 (#PCDATA)>
<!ATTLIST DRSB.01.04
  OPIS          CDATA #FIXED "Data nadania"
  TYP           CDATA #FIXED "data"
  DŁUGOSC      CDATA #FIXED "8"
  WYPELNIENIA.ZUS CDATA #FIXED "TAK">
<!ELEMENT DRSB.02.04 (#PCDATA)>
<!ATTLIST DRSB.02.04
  OPIS          CDATA #FIXED "Rodzaj dokumentu"
  TYP           CDATA #FIXED "kod"
  SŁOWNIK       CDATA #FIXED "rodzaj.dok">
```

## Informacje zwrotne

- Nie mogą być kodowane w atrybutach:
  - może być więcej niż jeden błąd lub korekta, dotycząca tego samego pola,
  - zawartości mogą zawierać podelementy,
  - niedozwolony model (#PCDATA, BLAD\*, KOREKTA\*)
- Rozwiązanie:
  - opcjonalne elementy po elemencie, w którym wystąpił błąd.

## Informacje zwrotne – przykład

```
<!ELEMENT BLAD      EMPTY>
<!ATTLIST BLAD      KOD CDATA #REQUIRED
  OPIS CDATA #IMPLIED>
<!ELEMENT KOREKTA  ANY>
<!ATTLIST KOREKTA  NR CDATA #REQUIRED
  TYP (OCR.1|OCR.2|OCR.3|SYSTEM)
  #REQUIRED>

<!ELEMENT DRSB
  ((DRSB.01, (BLAD*, KOREKTA*)),
   (DRSB.02, (BLAD*, KOREKTA*)),
   (DRSB.03, (BLAD*, KOREKTA*)),
   ...
  )>
```

