

XML

i nowoczesne metody zarządzania treścią

Wykład 5: Dobre praktyki w modelowaniu
oraz inne formalizmy reprezentacji schematów

Maciej Ogrodniczuk

MIMUW, 29 października 2009

- 1 **czytelność**: schematy, które łatwo zrozumieć, są łatwiejsze w utrzymaniu i istnieje większa szansa, że będą wielokrotnie wykorzystywane,
- 2 **precyzja opisu**: poprawnie skonstruowane typy potrafią wyeliminować błędy w danych (bardzo ważne w przypadku wymiany danych z aplikacjami, nad którymi nie mamy kontroli),
- 3 **gotowość do wielokrotnego użytku**: oszczędność, lepsza konstrukcja schematu, „mniej znaczy więcej” ,
- 4 **elastyczność i rozszerzalność**: pomaga spełnić wiele różnorodnych wymagań użytkowników, wspiera przyszłe wykorzystanie schematu, umożliwia obsługę zmian.

Większość za Priscillą Walmsley:

- 1 o nazwach składników schematu,
- 2 własności: nazwy konkretne czy ogólne?
- 3 unikanie konfliktów nazw,
- 4 `xsd:string`, `xsd:normalizedString` czy `xsd:token`?
- 5 elementy grupujące?
- 6 wartości nieokreślone,
- 7 listy wartości,
- 8 przestrzenie nazw,
- 9 globalne czy lokalne deklaracje elementów?
- 10 typy nazwane czy anonimowe?

<http://www.datypic.com/services/xmldesign/XMLDesignColor.pdf>

O nazwach składników schematu

Nazwy powinny być:

- znaczące i proste do zapamiętania,
- jak najmniej magiczne i skrótowe (**PTNM**),
- poręczne (**adresZamawiającegoProjektBudowlany**),
- zapisane w spójny sposób (**KodPocztowy**, **kod-pocztowy**, **kod_pocztowy**, **kod.pocztowy**, **kodPocztowy**), z użyciem ustandaryzowanego słownictwa,
- dla typów i grup opatrzone odpowiednim prefiksem/sufiksem.

O nazwach składników schematu

Nazwy powinny być:

- znaczące i proste do zapamiętania,
- jak najmniej magiczne i skrótowe (PTNM),
- poręczne (adresZamawiającegoProjektBudowlany),
- zapisane w spójny sposób (KodPocztowy, kod-pocztowy, kod_pocztowy, kod.pocztowy, kodPocztowy), z użyciem ustandaryzowanego słownictwa,
- dla typów i grup opatrzone odpowiednim prefiksem/sufiksem.

<produkt>/<numerProduktu> czy jednak <produkt>/<numer>?

- + czytelniejsze znaczenie elementu,
- + prostsze przetwarzanie (niezależność od rodzica, możliwość pobrania elementu wg nazwy),
- zapis mimo wszystko nadmiarowy,
- ukrywa fakt reprezentacji podobnych własności w różnych elementach.

Własności: nazwy konkretne czy ogólne?

Nazwy konkretne:

```
<długość>60</długość>  
<szerokość>50</szerokość>  
<wysokość>52</wysokość>  
<ciężar>25</ciężar>
```

- + możliwość przypisywania typów danych,
- + możliwość określania wymagalności i liczby wystąpień,
- konieczność zmiany schematu w przypadku dodania nowej własności.

Nazwy ogólne:

```
<cecha  
  nazwa="długość">60</cecha>  
<cecha  
  nazwa="szerokość">50</cecha>  
<cecha  
  nazwa="wysokość">52</cecha>  
<cecha  
  nazwa="ciężar">25</cecha>
```

- + niezmiennosc schematu przy dodaniu nowej własności,
- + łatwiejsze przetwarzanie (np. wyświetlenie listy wszystkich własności),
- brak możliwości definiowania typów, określania wymagalności i liczby wystąpień.

Czy taka definicja jest poprawna?

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
  <xsd:complexType name="tekst"/>  
  <xsd:element name="tekst"/>
```

```
</xsd:schema>
```

Czy taka definicja jest poprawna?

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
  <xsd:complexType name="tekst"/>  
  <xsd:element name="tekst"/>
```

```
</xsd:schema>
```

Czy taka definicja jest poprawna?

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
  <xsd:complexType name="tekst"/>  
  <xsd:element name="tekst"/>  
  <xsd:attribute name="tekst"/>
```

```
</xsd:schema>
```

Czy taka definicja jest poprawna?

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">  
  <xsd:complexType name="tekst"/>  
  <xsd:element name="tekst"/>  
  <xsd:attribute name="tekst"/>
```

```
</xsd:schema>
```

Czy taka definicja jest poprawna?

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="tekst"/>
  <xsd:element name="tekst"/>
  <xsd:attribute name="tekst"/>
  <xsd:simpleType name="tekst">
    <xsd:list itemType="xsd:token"/>
  </xsd:simpleType>
</xsd:schema>
```

Czy taka definicja jest poprawna?

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="tekst"/>
  <xsd:element name="tekst"/>
  <xsd:attribute name="tekst"/>
  <xsd:simpleType name="tekst">
    <xsd:list itemType="xsd:token"/>
  </xsd:simpleType>
</xsd:schema>
```

Czy taka definicja jest poprawna?

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:complexType name="tekst"/>
  <xsd:element name="tekst"/>
  <xsd:attribute name="tekst"/>
  <xsd:simpleType name="tekst">
    <xsd:list itemType="xsd:token"/>
  </xsd:simpleType>
</xsd:schema>
```

Zasady ogólne:

- elementy i atrybuty mogą nazywać się tak samo,
- typy mogą nazywać się tak samo jak elementy lub atrybuty,
- typy nie mogą nazywać się tak samo jak inne typy.

xsd:string

- gdy formatowanie białymi znakami ma znaczenie,
- dla długich napisów — warto też wtedy rozważyć użycie zawartości mieszanej.

xsd:string, xsd:normalizedString czy xsd:token?

xsd:string

- gdy formatowanie białymi znakami ma znaczenie,
- dla długich napisów — warto też wtedy rozważyć użycie zawartości mieszanej.

xsd:normalizedString

- gdy formatowanie białymi znakami nie ma znaczenia, ale ważne są pozycje znaków,

xsd:string

- gdy formatowanie białymi znakami ma znaczenie,
- dla długich napisów — warto też wtedy rozważyć użycie zawartości mieszanej.

xsd:normalizedString

- gdy formatowanie białymi znakami nie ma znaczenia, ale ważne są pozycje znaków,

xsd:token

- sprawdza się świetnie w przypadku krótkich napisów, zwłaszcza ograniczonych wyliczeniem lub wzorcem.

Bez elementu grupującego:

```
<klient>  
  <nazwa>MIMUW</nazwa>  
  <ulica>Banacha 2</ulica>  
  <kod>02-097</kod>  
  <miasto>Warszawa</miasto>  
</klient>
```

Bez elementu grupującego:

```
<klient>  
  <nazwa>MIMUW</nazwa>  
  <ulica>Banacha 2</ulica>  
  <kod>02-097</kod>  
  <miasto>Warszawa</miasto>  
</klient>
```

Z elementem grupującym:

```
<klient>  
  <nazwa>MIMUW</nazwa>  
  <adres>  
    <ulica>Banacha 2</ulica>  
    <kod>02-097</kod>  
    <miasto>Warszawa</miasto>  
  </adres>  
</klient>
```

Bez elementu grupującego:

```
<klient>
  <nazwa>MIMUW</nazwa>
  <ulica>Banacha 2</ulica>
  <kod>02-097</kod>
  <miasto>Warszawa</miasto>
</klient>
```

Z elementem grupującym:

```
<klient>
  <nazwa>MIMUW</nazwa>
  <adres>
    <ulica>Banacha 2</ulica>
    <kod>02-097</kod>
    <miasto>Warszawa</miasto>
  </adres>
</klient>
```

- + bardziej intuicyjne, łatwiejsze do wypełnienia przez człowieka,
- + łatwiejsze do przetwarzania, np. przez XSLT,
- trochę nadmiarowe.

O wartościach nieokreślonych

Użycie wartości nieokreślonych (ang. *nil values*):

- nie osłabia definicji typu poprzez dopuszczenie zawartości pustej,
- pozwala na jednoznacznie określenie, że informacja nie istnieje,
- umożliwia przekazanie informacji o nieokreśloności bez usuwania elementu z zawartości (obecność elementu może być wykorzystywana w aplikacji),
- umożliwia wyłączenie dodawania wartości domyślnych.

O wartościach nieokreślonych

Użycie wartości nieokreślonych (ang. *nil values*):

- nie osłabia definicji typu poprzez dopuszczenie zawartości pustej,
- pozwala na jednoznacznie określenie, że informacja nie istnieje,
- umożliwia przekazanie informacji o nieokreśloności bez usuwania elementu z zawartości (obecność elementu może być wykorzystywana w aplikacji),
- umożliwia wyłączenie dodawania wartości domyślnych.

Nie da się ich użyć dla wartości typów nienapisowych, chyba że...

O wartościach nieokreślonych

Użycie wartości nieokreślonych (ang. *nil values*):

- nie osłabia definicji typu poprzez dopuszczenie zawartości pustej,
- pozwala na jednoznacznie określenie, że informacja nie istnieje,
- umożliwia przekazanie informacji o nieokreśloności bez usuwania elementu z zawartości (obecność elementu może być wykorzystywana w aplikacji),
- umożliwia wyłączenie dodawania wartości domyślnych.

Nie da się ich użyć dla wartości typów nienapisowych, chyba że...
wykonamy pewną sztuczkę:

O wartościach nieokreślonych

Użycie wartości nieokreślonych (ang. *nil values*):

- nie osłabia definicji typu poprzez dopuszczenie zawartości pustej,
- pozwala na jednoznacznie określenie, że informacja nie istnieje,
- umożliwia przekazanie informacji o nieokreśloności bez usuwania elementu z zawartości (obecność elementu może być wykorzystywana w aplikacji),
- umożliwia wyłączenie dodawania wartości domyślnych.

Nie da się ich użyć dla wartości typów nienapisowych, chyba że...
wykonamy pewną sztuczkę:

```
<xsd:simpleType>  
  <xsd:union memberTypes="xsd:integer">  
    <xsd:simpleType>  
      <xsd:restriction base="xsd:token">  
        <xsd:enumeration value=""/>  
      </xsd:restriction>  
    </xsd:simpleType>  
  </xsd:union>  
</xsd:simpleType>
```

Przykład:

```
<xsd:simpleType name="województwo">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="mazowieckie"/>
    <xsd:enumeration value="wielkopolskie"/>
    ...
  </xsd:restriction>
</xsd:simpleType>
```

Przykład:

```
<xsd:simpleType name="województwo">  
  <xsd:restriction base="xsd:token">  
    <xsd:enumeration value="mazowieckie"/>  
    <xsd:enumeration value="wielkopolskie"/>  
    ...  
  </xsd:restriction>  
</xsd:simpleType>
```

Wyzwania:

- 1 potencjalnie częste zmiany, często poza kontrolą projektanta schematu (kody języków, walut, państw)

Przykład:

```
<xsd:simpleType name="województwo">  
  <xsd:restriction base="xsd:token">  
    <xsd:enumeration value="mazowieckie"/>  
    <xsd:enumeration value="wielkopolskie"/>  
    ...  
  </xsd:restriction>  
</xsd:simpleType>
```

Wyzwania:

- 1 potencjalnie częste zmiany, często poza kontrolą projektanta schematu (kody języków, walut, państw) → warto utrzymywać je w osobnych dokumentach schematu, co pozwala na ich wersjonowanie,

Przykład:

```
<xsd:simpleType name="województwo">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="mazowieckie"/>
    <xsd:enumeration value="wielkopolskie"/>
    ...
  </xsd:restriction>
</xsd:simpleType>
```

Wyzwania:

- 1 potencjalnie częste zmiany, często poza kontrolą projektanta schematu (kody języków, walut, państw) → warto utrzymywać je w osobnych dokumentach schematu, co pozwala na ich wersjonowanie,
- 2 długość list spowalniająca walidację, zaśmiecająca schemat, utrudniająca zarządzanie

Przykład:

```
<xsd:simpleType name="województwo">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="mazowieckie"/>
    <xsd:enumeration value="wielkopolskie"/>
    ...
  </xsd:restriction>
</xsd:simpleType>
```

Wyzwania:

- 1 potencjalnie częste zmiany, często poza kontrolą projektanta schematu (kody języków, walut, państw) → warto utrzymywać je w osobnych dokumentach schematu, co pozwala na ich wersjonowanie,
- 2 długość list spowalniająca walidację, zaśmiecająca schemat, utrudniająca zarządzanie → warto ograniczać listy do maksymalnie 15-20 wartości, dokumentować listy, używać wzorców,

Przykład:

```
<xsd:simpleType name="województwo">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="mazowieckie"/>
    <xsd:enumeration value="wielkopolskie"/>
    ...
  </xsd:restriction>
</xsd:simpleType>
```

Wyzwania:

- 1 potencjalnie częste zmiany, często poza kontrolą projektanta schematu (kody języków, walut, państw) → warto utrzymywać je w osobnych dokumentach schematu, co pozwala na ich wersjonowanie,
- 2 długość list spowalniająca walidację, zaśmiecająca schemat, utrudniająca zarządzanie → warto ograniczać listy do maksymalnie 15-20 wartości, dokumentować listy, używać wzorców,
- 3 brak rozszerzalności typów prostych

Przykład:

```
<xsd:simpleType name="województwo">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="mazowieckie"/>
    <xsd:enumeration value="wielkopolskie"/>
    ...
  </xsd:restriction>
</xsd:simpleType>
```

Wyzwania:

- 1 potencjalnie częste zmiany, często poza kontrolą projektanta schematu (kody języków, walut, państw) → warto utrzymywać je w osobnych dokumentach schematu, co pozwala na ich wersjonowanie,
- 2 długość list spowalniająca walidację, zaśmiecająca schemat, utrudniająca zarządzanie → warto ograniczać listy do maksymalnie 15-20 wartości, dokumentować listy, używać wzorców,
- 3 brak rozszerzalności typów prostych → warto użyć typu `xsd:token` lub ew. jego unii z listą wyliczeniową, by inne dokumenty mogły zawęzić tę listę.

Rozszerzalne listy wartości: przykład

```
<xsd:simpleType name="językiKanady">
  <xsd:union memberTypes="xsd:token">
    <xsd:simpleType>
      <xsd:restriction base="xsd:token">
        <xsd:enumeration value="angielski"/>
        <xsd:enumeration value="francuski"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>
```

Rozszerzalne listy wartości: przykład

```
<xsd:simpleType name="językiKanady">
  <xsd:union memberTypes="xsd:token">
    <xsd:simpleType>
      <xsd:restriction base="xsd:token">
        <xsd:enumeration value="angielski"/>
        <xsd:enumeration value="francuski"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>

<xsd:simpleType name="językiKanadyPoImigracji">
  <xsd:restriction base="językiKanady">
    <xsd:enumeration value="angielski"/>
    <xsd:enumeration value="polski"/>
  </xsd:restriction>
</xsd:simpleType>
```

Zalecenia ogólne:

- warto umieszczać deklaracje przestrzeni nazw w elemencie głównym,
- domyślne przestrzenie nazw sprawdzają się tylko wtedy, gdy jedna z nich dominuje w dokumencie,
- prefiksy nie mają znaczenia, ale warto trzymać się konwencji (xsd lub xs dla schematów...)

Zalecenia ogólne:

- warto umieszczać deklaracje przestrzeni nazw w elemencie głównym,
- domyślne przestrzenie nazw sprawdzają się tylko wtedy, gdy jedna z nich dominuje w dokumencie,
- prefiksy nie mają znaczenia, ale warto trzymać się konwencji (xsd lub xs dla schematów...)

Trzy sposoby użycia przestrzeni nazw w schemacie:

- 1 przestrzeń docelowa jako domyślna (rozwiązanie najczęstsze),
- 2 przestrzeń XML Schema jako domyślna,
- 3 bez domyślnej przestrzeni nazw.

Zapis:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://mimuw.edu.pl/usos"
            xmlns="http://mimuw.edu.pl/usos">
  <xsd:complexType name="typOsoba">
    ...
  </xsd:complexType>
  <xsd:element name="student" type="typOsoba"/>
  <xsd:element name="data-rozp-studiów" type="xsd:date"/>
</xsd:schema>
```

Uwagi:

- komponenty XML Schema zawsze prefiksowane,
- bezprefiksowy zapis odwołań do komponentów zdefiniowanych w schemacie (oprócz ścieżek XPath w składnikach elementów `<xsd:key>` i `<xsd:keyref>` – tam domyślna przestrzeń nazw nie obowiązuje).

Zapis:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://mimuw.edu.pl/usos"
            xmlns="http://mimuw.edu.pl/usos">
  <xsd:complexType name="typOsoba">
    ...
  </xsd:complexType>
  <xsd:element name="student" type="typOsoba"/>
  <xsd:element name="data-rozp-studiów" type="xsd:date"/>
</xsd:schema>
```

Uwagi:

- komponenty XML Schema zawsze prefiksowane,
- bezprefiksowy zapis odwołań do komponentów zdefiniowanych w schemacie (oprócz ścieżek XPath w składnikach elementów `<xsd:key>` i `<xsd:keyref>` – tam domyślna przestrzeń nazw nie obowiązuje).

Zapis:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://mimuw.edu.pl/usos"
            xmlns="http://mimuw.edu.pl/usos">
  <xsd:complexType name="typOsoba">
    ...
  </xsd:complexType>
  <xsd:element name="student" type="typOsoba"/>
  <xsd:element name="data-rozp-studiów" type="xsd:date"/>
</xsd:schema>
```

Uwagi:

- komponenty XML Schema zawsze prefiksowane,
- **bezprefiksowy zapis odwołań do komponentów zdefiniowanych w schemacie** (oprócz ścieżek XPath w składnikach elementów `<xsd:key>` i `<xsd:keyref>` – tam domyślna przestrzeń nazw nie obowiązuje).

Zapis:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://mimuw.edu.pl/usos"
        xmlns:usos="http://mimuw.edu.pl/usos">
  <complexType name="typOsoba">
    ...
  </complexType>
  <element name="student" type="usos:typOsoba"/>
  <element name="data-rozp-studiów" type="date"/>
</schema>
```

Uwagi:

- komponenty XML Schema nie są prefiksowane,
- prefiksowy zapis odwołań do komponentów zdefiniowanych w schemacie.

Zapis:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://mimuw.edu.pl/usos"
        xmlns:usos="http://mimuw.edu.pl/usos">
  <complexType name="typOsoba">
    ...
  </complexType>
  <element name="student" type="usos:typOsoba"/>
  <element name="data-rozp-studiów" type="date"/>
</schema>
```

Uwagi:

- komponenty XML Schema nie są prefiksowane,
- prefiksowy zapis odwołań do komponentów zdefiniowanych w schemacie.

Zapis:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://mimuw.edu.pl/usos"
        xmlns:usos="http://mimuw.edu.pl/usos">
  <complexType name="typOsoba">
    ...
  </complexType>
  <element name="student" type="usos:typOsoba"/>
  <element name="data-rozp-studiów" type="date"/>
</schema>
```

Uwagi:

- komponenty XML Schema nie są prefiksowane,
- prefiksowy zapis odwołań do komponentów zdefiniowanych w schemacie.

Zapis:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://mimuw.edu.pl/usos"
        xmlns:usos="http://mimuw.edu.pl/usos">
  <complexType name="typOsoba">
    ...
  </complexType>
  <element name="student" type="usos:typOsoba"/>
  <element name="data-rozp-studiów" type="date"/>
</schema>
```

Uwagi:

- komponenty XML Schema nie są prefiksowane,
- prefiksowy zapis odwołań do komponentów zdefiniowanych w schemacie.

UWAGA: nie da się używać bez docelowej przestrzeni nazw – nie ma jak odwołać się do zdefiniowanych komponentów!

Zapis:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://mimuw.edu.pl/usos"
            xmlns:usos="http://mimuw.edu.pl/usos">
  <xsd:complexType name="typOsoba">
    ...
  </xsd:complexType>
  <xsd:element name="student" type="usos:typOsoba"/>
  <xsd:element name="data-rozp-studiów" type="xsd:date"/>
</xsd:schema>
```

Zapis:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://mimuw.edu.pl/usos"
            xmlns:usos="http://mimuw.edu.pl/usos">
  <xsd:complexType name="typOsoba">
    ...
  </xsd:complexType>
  <xsd:element name="student" type="usos:typOsoba"/>
  <xsd:element name="data-rozp-studiów" type="xsd:date"/>
</xsd:schema>
```

Zapis:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://mimuw.edu.pl/usos"
            xmlns:usos="http://mimuw.edu.pl/usos">
  <xsd:complexType name="typOsoba">
    ...
  </xsd:complexType>
  <xsd:element name="student" type="usos:typOsoba"/>
  <xsd:element name="data-rozp-studiów" type="xsd:date"/>
</xsd:schema>
```

Zapis:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://mimuw.edu.pl/usos"
            xmlns:usos="http://mimuw.edu.pl/usos">
  <xsd:complexType name="typOsoba">
    ...
  </xsd:complexType>
  <xsd:element name="student" type="usos:typOsoba"/>
  <xsd:element name="data-rozp-studiów" type="xsd:date"/>
</xsd:schema>
```

Uwagi:

- wszystko z prefiksem (oczywiście oprócz wprowadzanych nazw),
- najlepsze rozwiązanie dla przypadku współistnienia wielu przestrzeni nazw.

Globalne czy lokalne deklaracje elementów?

Deklaracje globalne:

```
<xsd:element name="osoba"/>
<xsd:element name="zwycięzcy">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="osoba"
        minOccurs="1" maxOccurs="10"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Globalne czy lokalne deklaracje elementów?

Deklaracje globalne:

```
<xsd:element name="osoba"/>
<xsd:element name="zwycięzcy">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="osoba"
        minOccurs="1" maxOccurs="10"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Deklaracje lokalne:

```
<xsd:element name="zwycięzcy">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="osoba"
        minOccurs="1" maxOccurs="10"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Globalne czy lokalne deklaracje elementów?

Deklaracje globalne:

```
<xsd:element name="osoba"/>
<xsd:element name="zwycięzcy">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="osoba"
        maxOccurs="10"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

- + mogą być wielokrotnie wykorzystywane,
- muszą mieć unikalne nazwy w całym schemacie (uwaga na części dołączane!)
- jeśli określono docelową przestrzeń nazw, używamy nazw kwalifikowanych.

Deklaracje lokalne:

```
<xsd:element name="zwycięzcy">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="osoba"
        maxOccurs="10"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Globalne czy lokalne deklaracje elementów?

Deklaracje globalne:

```
<xsd:element name="osoba"/>
<xsd:element name="zwyciezcy">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="osoba"
        maxOccurs="10"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

- + mogą być wielokrotnie wykorzystywane,
- muszą mieć unikalne nazwy w całym schemacie (uwaga na części dołączane!)
- jeśli określono docelową przestrzeń nazw, używamy nazw kwalifikowanych.

Deklaracje lokalne:

```
<xsd:element name="zwyciezcy">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="osoba"
        maxOccurs="10"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

- + nie muszą być unikalne w całym schemacie,
- + mogą być kwalifikowane lub nie (co upraszcza dokument),
- nie mogą być walidowane niezależnie od elementu nadrzędnego.

Globalne czy lokalne deklaracje elementów?

W schemacie nie wyróżniamy żadnego elementu – każdy może stać się elementem głównym dokumentu.

Globalne czy lokalne deklaracje elementów?

W schemacie nie wyróżniamy żadnego elementu – każdy może stać się elementem głównym dokumentu.

Jeśli to niepożądane, możemy pozostawić jako globalne tylko te elementy, którym chcemy pozwolić na pełnienie funkcji elementu głównego (wszystkie pozostałe lokalne).

Globalne czy lokalne deklaracje elementów?

W schemacie nie wyróżniamy żadnego elementu – każdy może stać się elementem głównym dokumentu.

Jeśli to niepożądane, możemy pozostawić jako globalne tylko te elementy, którym chcemy pozwolić na pełnienie funkcji elementu głównego (wszystkie pozostałe lokalne).

Aby móc użyć elementów lokalnych wielokrotnie, możemy otoczyć je grupą:

```
<xsd:group name="grupaAdres">  
  <xsd:sequence>  
    <xsd:element name="adres"/>  
  </xsd:sequence>  
</xsd:group>
```

Globalne czy lokalne deklaracje elementów?

W schemacie nie wyróżniamy żadnego elementu – każdy może stać się elementem głównym dokumentu.

Jeśli to niepożądane, możemy pozostawić jako globalne tylko te elementy, którym chcemy pozwolić na pełnienie funkcji elementu głównego (wszystkie pozostałe lokalne).

Aby móc użyć elementów lokalnych wielokrotnie, możemy otoczyć je grupą:

```
<xsd:group name="grupaAdres">  
  <xsd:sequence>  
    <xsd:element name="adres"/>  
  </xsd:sequence>  
</xsd:group>
```

i potem używać odwołania do grupy:

```
<xsd:element name="firma">  
  <xsd:complexType>  
    <xsd:group ref="grupaAdres"/>  
  </xsd:complexType>  
</xsd:element>
```

Globalne czy lokalne deklaracje elementów?

W schemacie nie wyróżniamy żadnego elementu – każdy może stać się elementem głównym dokumentu.

Jeśli to niepożądane, możemy pozostawić jako globalne tylko te elementy, którym chcemy pozwolić na pełnienie funkcji elementu głównego (wszystkie pozostałe lokalne).

Aby móc użyć elementów lokalnych wielokrotnie, możemy otoczyć je grupą:

```
<xsd:group name="grupaAdres">
  <xsd:sequence>
    <xsd:element name="adres"/>
  </xsd:sequence>
</xsd:group>
```

i potem używać odwołania do grupy:

```
<xsd:element name="firma">
  <xsd:complexType>
    <xsd:group ref="grupaAdres"/>
  </xsd:complexType>
</xsd:element>
```

zamiast odwołania do elementu: `<xs:element ref="adres"/>`.

Typy nazwane czy anonimowe?

Typy nazwane:

```
<xsd:complexType
  name="od1do10osób">
  <xsd:sequence>
    <xsd:element name="osoba"
      maxOccurs="10"/>
  </xsd:sequence>
</xsd:complexType>
```

Typy nazwane czy anonimowe?

Typy nazwane:

```
<xsd:complexType
  name="od1do10osób">
  <xsd:sequence>
    <xsd:element name="osoba"
      maxOccurs="10"/>
  </xsd:sequence>
</xsd:complexType>
```

Typy anonimowe:

```
<xsd:element name="zwycięzcy">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="osoba"
        maxOccurs="10"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Typy nazwane czy anonimowe?

Typy nazwane:

```
<xsd:complexType
  name="od1do10osób">
  <xsd:sequence>
    <xsd:element name="osoba"
      maxOccurs="10"/>
  </xsd:sequence>
</xsd:complexType>
```

- + mogą być wielokrotnie wykorzystywane,
- + mogą być podstawą innych typów; można ich użyć w definicjach list i unii,
- + uczytelniają schemat zawierający złożone definicje.

Typy anonimowe:

```
<xsd:element name="zwycięzcy">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="osoba"
        maxOccurs="10"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Typy nazwane czy anonimowe?

Typy nazwane:

```
<xsd:complexType
  name="od1do10osób">
  <xsd:sequence>
    <xsd:element name="osoba"
      maxOccurs="10"/>
  </xsd:sequence>
</xsd:complexType>
```

- + mogą być wielokrotnie wykorzystywane,
- + mogą być podstawą innych typów; można ich użyć w definicjach list i unii,
- + uczylelniają schemat zawierający złożone definicje.

Typy anonimowe:

```
<xsd:element name="zwycięzcy">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="osoba"
        maxOccurs="10"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

- + jeśli definicja nie będzie powtórnie wykorzystywana, mogą uprościć utrzymanie schematu (bo nie trzeba martwić się o ich wpływ na pozostałe komponenty),
- + w prostych przypadkach mogą być czytelniejsze.

Za pomocą:

- 1 specjalnego elementu `<xsd:annotation>`,
- 2 komentarzy XML-owych,
- 3 atrybutów dołączanych (ang. *foreign attributes*).

Za pomocą:

- 1 specjalnego elementu `<xsd:annotation>`,
- 2 komentarzy XML-owych,
- 3 atrybutów dołączanych (ang. *foreign attributes*).

Oraz oczywiście wykorzystując sposoby nieschematocentryczne:

- opisując schemat poza nim samym,
- przechowując łącznie opisy danego komponentu schematu, dostępne przekształcenia i style,
- ...

<xsd:annotation> — dokumentacja schematu

Element <xsd:annotation> może wystąpić w dowolnym miejscu na poziomie globalnym oraz na początku wszystkich konstrukcji XML Schema.

<xsd:annotation> — dokumentacja schematu

Element `<xsd:annotation>` może wystąpić w dowolnym miejscu na poziomie globalnym oraz na początku wszystkich konstrukcji XML Schema.

Jego zawartość to mieszanka elementów `<xsd:documentation>` i `<xsd:appinfo>` zawierających dodatkowe informacje (tekst i znaczniki) odpowiednio dla ludzi i maszyn.

<xsd:annotation> — dokumentacja schematu

Element <xsd:annotation> może wystąpić w dowolnym miejscu na poziomie globalnym oraz na początku wszystkich konstrukcji XML Schema.

Jego zawartość to mieszanka elementów <xsd:documentation> i <xsd:appinfo> zawierających dodatkowe informacje (tekst i znaczniki) odpowiednio dla ludzi i maszyn.

```
<xsd:element name="nazwisko" type="xsd:string">
  <xsd:annotation>
    <xsd:documentation xml:lang="pl">
      Nazwisko adresata.
    </xsd:documentation>
    <xsd:appinfo>
      <form:label>Podaj nazwisko adresata:</form:label>
    </xsd:appinfo>
  </xsd:annotation>
</xsd:element>
```

Definicja <xsd:appinfo>

```
<xsd:element name="appinfo">
  <xsd:annotation>
    <xsd:documentation source="http://www.w3.org/TR/
      xmlschema-1/#element-appinfo"/>
  </xsd:annotation>
  <xsd:complexType mixed="true">
    <xsd:sequence minOccurs="0" maxOccurs="unbounded">
      <xsd:any processContents="lax"/>
    </xsd:sequence>
    <xsd:attribute name="source" type="xsd:anyURI"/>
  </xsd:complexType>
</xsd:element>
```

Oba elementy mogą zostać opatrzone opcjonalnym atrybutem source wskazującym źródło informacji.

Co umieszczać w <xsd:appinfo>?

Na przykład:

- dodatkowe reguły walidacji (np. schematronowe),
- odwzorowanie na inne technologie (np. schemat relacyjny),
- odwzorowanie na formularze (np. XHTML-owe znaczniki <form>).

Każdy komponent schematu może zawierać atrybuty pochodzące z dowolnej przestrzeni nazw. Atrybuty te nie są walidowane i nie trzeba ich deklarować.

Każdy komponent schematu może zawierać atrybuty pochodzące z dowolnej przestrzeni nazw. Atrybuty te nie są walidowane i nie trzeba ich deklarować.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:doc="http://www.mimuw.edu.pl/xml">
  <xsd:element name="wykład" type="typPrezentacja"
              doc:opis="Element główny prezentacji
                       na potrzeby wykładu."/>
</xsd:schema>
```

Warto standaryzować:

Warto standaryzować:

- nazewnictwo,

Warto standaryzować:

- nazewnictwo,
- sposób budowy schematu,

Warto standaryzować:

- nazewnictwo,
- sposób budowy schematu,
- sposoby rozszerzania schematu,

Warto standaryzować:

- nazewnictwo,
- sposób budowy schematu,
- sposoby rozszerzania schematu,
- zakres wykorzystania konstrukcji XML Schema.

Warto standaryzować:

- nazewnictwo,
- sposób budowy schematu,
- sposoby rozszerzania schematu,
- zakres wykorzystania konstrukcji XML Schema.

Często tego rodzaju zalecenia zapisuje się formalnie w dokumencie NDR (ang. *naming and design rules*).

Warto standaryzować:

- nazewnictwo,
- sposób budowy schematu,
- sposoby rozszerzania schematu,
- zakres wykorzystania konstrukcji XML Schema.

Często tego rodzaju zalecenia zapisuje się formalnie w dokumencie NDR (ang. *naming and design rules*).

Warto też używać oprogramowania do kontroli nad:

- edycją: kto co zmienia,

Warto standaryzować:

- nazewnictwo,
- sposób budowy schematu,
- sposoby rozszerzania schematu,
- zakres wykorzystania konstrukcji XML Schema.

Często tego rodzaju zalecenia zapisuje się formalnie w dokumencie NDR (ang. *naming and design rules*).

Warto też używać oprogramowania do kontroli nad:

- edycją: kto co zmienia,
- wersjami składników schematu,

Warto standaryzować:

- nazewnictwo,
- sposób budowy schematu,
- sposoby rozszerzania schematu,
- zakres wykorzystania konstrukcji XML Schema.

Często tego rodzaju zalecenia zapisuje się formalnie w dokumencie NDR (ang. *naming and design rules*).

Warto też używać oprogramowania do kontroli nad:

- edycją: kto co zmienia,
- wersjami składników schematu,
- fragmentami schematu stosowanymi w różnych zastosowaniach.

Jak modelować?

- analizując zależności między modelowanymi obiektami i ich częściami,
- wyodrębniając podstruktury obiektów,
- analizując dostępne dokumenty przykładowe,
- analizując potencjalne zastosowania dokumentów oraz przypadki użycia,
- budując abstrakcyjny projekt struktury,
- zapisując model,
- testując model w „rzeczywistym świecie”,
- utrzymując (pielęgnując) strukturę podczas jej wykorzystania,
- pamiętając o zarządzaniu zmianami.

Problem:

- Konieczność zmiany struktury.

Problem:

- Konieczność zmiany struktury.

Warianty rozwiązania:

- 1 wprowadzamy zmianę modelu kompatybilną wstecz (np. dodajemy elementy, ale opcjonalne),
- 2 używamy dwóch wersji schematu,
- 3 migrujemy dokumenty: przekształcamy automatycznie i/lub instruujemy użytkowników o konieczności migracji do nowej struktury.

Problem:

- Konieczność zmiany struktury.

Warianty rozwiązania:

- 1 wprowadzamy zmianę modelu kompatybilną wstecz (np. dodajemy elementy, ale opcjonalne),
- 2 używamy dwóch wersji schematu,
- 3 migrujemy dokumenty: przekształcamy automatycznie i/lub instruujemy użytkowników o konieczności migracji do nowej struktury.

Najlepiej: tworzymy aplikację odporną na zmianę struktury dokumentów.

Kilka metod:

Kilka metod:

- przeniesienie sprawdzania poprawności dokumentu na poziom schematu,

Kilka metod:

- przeniesienie sprawdzania poprawności dokumentu na poziom schematu,
- pomijanie nieistotnych elementów i atrybutów,

Kilka metod:

- przeniesienie sprawdzania poprawności dokumentu na poziom schematu,
- pomijanie nieistotnych elementów i atrybutów,
- unikanie zależności od struktury dokumentu:
//produkt/numer zamiast /katalog/produkt/numer,

Kilka metod:

- przeniesienie sprawdzania poprawności dokumentu na poziom schematu,
- pomijanie nieistotnych elementów i atrybutów,
- unikanie zależności od struktury dokumentu:
//produkt/numer zamiast /katalog/produkt/numer,
- parametryzacja schematem — użycie atrybutów stałych np. do przechowania etykiet pól formularza, odwzorowania elementów na tabele i pola w bazie danych itp.,

Kilka metod:

- przeniesienie sprawdzania poprawności dokumentu na poziom schematu,
- pomijanie nieistotnych elementów i atrybutów,
- unikanie zależności od struktury dokumentu:
//produkt/numer zamiast /katalog/produkt/numer,
- parametryzacja schematem — użycie atrybutów stałych np. do przechowania etykiet pól formularza, odwzorowania elementów na tabele i pola w bazie danych itp.,
- użycie przestrzeni nazw (jak w standardzie XLink).

Kilka metod:

- przeniesienie sprawdzania poprawności dokumentu na poziom schematu,
- pomijanie nieistotnych elementów i atrybutów,
- unikanie zależności od struktury dokumentu:
//produkt/numer zamiast /katalog/produkt/numer,
- parametryzacja schematem — użycie atrybutów stałych np. do przechowania etykiet pól formularza, odwzorowania elementów na tabele i pola w bazie danych itp.,
- użycie przestrzeni nazw (jak w standardzie XLink).

Nadrzędna zasada: zdrowy rozsądek.

Zachęcam do samodzielnych studiów nad:

- zależnościami aspektów od typów danych,
- regułami walidacji i ograniczeniami budowy schematu,
- użyciem encji, notacji, odpowiedników konstrukcji z DTD,
- grupami zamiennymi (ang. *substitution groups*),
- modelami struktury schematów (*Russian Doll*, *Salami Slice*, *Venetian Blind*, *Garden of Eden*),
- specyfikacją XML Schema w oryginale.

Polecam książkę Priscilli Walmsley *Wszystko o XML Schema* (WNT 2008).
Przykłady z książki: <http://www.datypic.com/books/defxmlschema/examples.html>.

Czego nie da się zrobić w XML Schema?

Problem:

- kontekstowe sprawdzanie poprawności,
np. *zawartość elementu <cena-netto> jest mniejsza
lub równa zawartości elementu <cena-brutto>*,
albo: lista liczb wylosowanych w Lotto jest posortowana.

Rozwiązania:

Czego nie da się zrobić w XML Schema?

Problem:

- kontekstowe sprawdzanie poprawności,
np. *zawartość elementu <cena-netto> jest mniejsza lub równa zawartości elementu <cena-brutto>*,
albo: lista liczb wylosowanych w Lotto jest posortowana.

Rozwiązania:

- zaprogramować w kodzie aplikacji,

Czego nie da się zrobić w XML Schema?

Problem:

- kontekstowe sprawdzanie poprawności,
np. *zawartość elementu <cena-netto> jest mniejsza lub równa zawartości elementu <cena-brutto>*,
albo: lista liczb wylosowanych w Lotto jest posortowana.

Rozwiązania:

- zaprogramować w kodzie aplikacji,
- wykorzystać XSLT,

Czego nie da się zrobić w XML Schema?

Problem:

- kontekstowe sprawdzanie poprawności, np. *zawartość elementu <cena-netto> jest mniejsza lub równa zawartości elementu <cena-brutto>*, albo: lista liczb wylosowanych w Lotto jest posortowana.

Rozwiązania:

- zaprogramować w kodzie aplikacji,
- wykorzystać XSLT,
- użyć innego języka schematów, np. Schematrona.

Problemy:

- niejednoznaczność (ang. *ambiguity*), czyli poprawność fragmentu względem kilku wzorców,
- niedeterminizm (ang. *non-determinism*), czyli sytuacja, w której procesor ma do wyboru wiele pasujących wzorców (produkcji gramatyki), a równoważny model deterministyczny nie istnieje.

Problemy:

- niejednoznaczność (ang. *ambiguity*), czyli poprawność fragmentu względem kilku wzorców,
- niedeterminizm (ang. *non-determinism*), czyli sytuacja, w której procesor ma do wyboru wiele pasujących wzorców (produkcji gramatyki), a równoważny model deterministyczny nie istnieje.

Rozwiązanie:

- Relax NG, czyli
REgular LAnguage description for XML – New Generation.

Autorstwa Ricka Jelliffe'a (1999). Standard ISO od 2006 r.

(jako część 3 standardu DSDL — Document Schema Definition Languages).

Idea: wzorce (`<pattern>`) zawierające kontekstowe reguły walidacji (`<rule>`)

złożone z posługujących się wyrażeniami XPath własności `<assert>`

i `<report>` odpowiednio wymaganych do spełnienia i oznaczających błąd.

Autorstwa Ricka Jelliffe'a (1999). Standard ISO od 2006 r.

(jako część 3 standardu DSDL — Document Schema Definition Languages).

Idea: wzorce (<pattern>) zawierające kontekstowe reguły walidacji (<rule>) złożone z posługujących się wyrażeniami XPath własności <assert> i <report> odpowiednio wymaganych do spełnienia i oznaczających błęd.

Schemat schematronowy dla schematrona (www.schematron.com):

```
<schema xmlns="http://www.mimuw.edu.pl/dsdl/schematron">
  <ns prefix="sch" uri="http://www.mimuw.edu.pl/dsdl/schematron">
    <pattern>
      <rule context="sch:schema">
        <assert test="sch:pattern">Schemat składa się z wzorców.</assert>
        <assert test="sch:pattern/sch:rule[@context]">Wzorzec składa
          się z reguł. Każda powinna mieć atrybut 'context'.</assert>
        <assert test="sch:pattern/sch:rule/sch:assert[@test]
          or sch:pattern/sch:rule/sch:report[@test]">Reguła
          składa się z instrukcji 'assert' i 'report',
          które muszą posiadać atrybut 'test'.</assert>
      </rule>
    </pattern>
  </schema>
```

Asercje wbudowane w schemat: przykład

```
<xsd:element name="towar">
  <xsd:annotation>
    <xsd:appinfo>
      <sch:pattern name="Cena brutto większa od netto.">
        <sch:rule context="sklep:towar">
          <sch:assert test="sklep:cenaBrutto > sklep:cenaNetto">
            Zawartość elementu 'cenaBrutto' powinna być większa
            niż zawartość elementu 'cenaNetto'.</sch:assert>
          </sch:rule>
        </sch:pattern>
      </xsd:appinfo>
    </xsd:annotation>
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="cenaNetto" type="xsd:integer"/>
        <xsd:element name="cenaBrutto" type="xsd:integer"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

Autorstwa OASIS, bazuje na wcześniejszych propozycjach RELAX Makoto Muraty i TREX Jamesa Clarka.

Od 2006 r. standard ISO — jako część 2 standardu DSDL.

Regular Language description for XML – New Generation:

- dwa warianty składni: XML-owa i „kompaktowa”,
- wsparcie dla przestrzeni nazw,
- jednolite traktowanie elementów i atrybutów,
- wsparcie dla zawartości nieuporządkowanej i mieszanej,
- może funkcjonować wraz z osobnym językiem typów (np. XML Schema).

Autorstwa OASIS, bazuje na wcześniejszych propozycjach RELAX Makoto Muraty i TREX Jamesa Clarka.

Od 2006 r. standard ISO — jako część 2 standardu DSDL.

Regular Language description for XML – New Generation:

- dwa warianty składni: XML-owa i „kompaktowa”,
- wsparcie dla przestrzeni nazw,
- jednolite traktowanie elementów i atrybutów,
- wsparcie dla zawartości nieuporządkowanej i mieszanej,
- może funkcjonować wraz z osobnym językiem typów (np. XML Schema).

W zasadzie same zalety:

- + prostszy w opisie schematu,
- + bardziej zaawansowany technicznie,
- + oferujący więcej możliwości,
- cieszący się zdecydowanie mniejszą popularnością (czyt.: wsparciem producentów oprogramowania).

RELAX NG rozszerza funkcjonalność DTD — w szczególności:

- wprowadza typy danych,
- integruje atrybuty z modelem zawartości,
- obsługuje przestrzenie nazw,
- zapewnia wsparcie dla dowolnej kolejności wystąpień elementów,
- obsługuje modele kontekstowe.

Jednocześnie RELAX NG:

- nie zapewnia walidacji ID/IDREF (jest dodatek, który to umożliwia),
- nie obsługuje atrybutów domyślnych,
- nie obsługuje encji znakowych, notacji,
- nie pozwala na określenie, czy białe znaki są znaczące,
- nie określa sposobu powiązania schematu Relax NG z dokumentem.

DTD:

```
<!ELEMENT wizytownik  
    (wizytówka*)>  
<!ELEMENT wizytówka  
    (osoba, email)>  
<!ELEMENT osoba (#PCDATA)>  
<!ELEMENT email (#PCDATA)>
```

DTD:

```
<!ELEMENT wizytownik  
    (wizytówka*)>  
<!ELEMENT wizytówka  
    (osoba, email)>  
<!ELEMENT osoba (#PCDATA)>  
<!ELEMENT email (#PCDATA)>
```

Składnia skrócona:

```
element wizytownik {  
    element wizytówka {  
        element osoba { text },  
        element e-mail { text }  
    }*  
}
```

Dwie składnie RELAX NG

DTD:

```
<!ELEMENT wizytownik  
    (wizytówka*)>  
<!ELEMENT wizytówka  
    (osoba, email)>  
<!ELEMENT osoba (#PCDATA)>  
<!ELEMENT email (#PCDATA)>
```

Składnia skrócona:

```
element wizytownik {  
    element wizytówka {  
        element osoba { text },  
        element e-mail { text }  
    }*  
}
```

Składnia XML-owa:

```
<element name="wizytownik"  
    xmlns="http://relaxng.org/  
        ns/structure/1.0">  
    <zeroOrMore>  
        <element  
            name="wizytówka">  
            <element name="osoba">  
                <text/>  
            </element>  
            <element name="e-mail">  
                <text/>  
            </element>  
        </zeroOrMore>  
    </element>
```

<http://www.relaxng.org>

Dwie składnie RELAX NG

DTD:

```
<!ELEMENT wizytownik  
    (wizytówka*)>  
<!ELEMENT wizytówka  
    (osoba, email)>  
<!ELEMENT osoba (#PCDATA)>  
<!ELEMENT email (#PCDATA)>
```

Składnia skrócona:

```
element wizytownik {  
    element wizytówka {  
        element osoba { text },  
        element e-mail { text }  
    }*  
}
```

Składnia XML-owa:

```
<element name="wizytownik"  
    xmlns="http://relaxng.org/  
        ns/structure/1.0">  
    <zeroOrMore>  
        <element  
            name="wizytówka">  
            <element name="osoba">  
                <text/>  
            </element>  
            <element name="e-mail">  
                <text/>  
            </element>  
        </zeroOrMore>  
    </element>
```

<http://www.relaxng.org>

Dwie składnie RELAX NG

DTD:

```
<!ELEMENT wizytownik  
    (wizytówka*)>  
<!ELEMENT wizytówka  
    (osoba, email)>  
<!ELEMENT osoba (#PCDATA)>  
<!ELEMENT email (#PCDATA)>
```

Składnia skrócona:

```
element wizytownik {  
    element wizytówka {  
        element osoba { text },  
        element e-mail { text }  
    }*  
}
```

Składnia XML-owa:

```
<element name="wizytownik"  
    xmlns="http://relaxng.org/  
        ns/structure/1.0">  
    <zeroOrMore>  
        <element  
            name="wizytówka">  
            <element name="osoba">  
                <text/>  
            </element>  
            <element name="e-mail">  
                <text/>  
            </element>  
        </element>  
    </zeroOrMore>  
</element>
```

<http://www.relaxng.org>

Dwie składnie RELAX NG

DTD:

```
<!ELEMENT wizytownik  
    (wizytówka*)>  
<!ELEMENT wizytówka  
    (osoba, email)>  
<!ELEMENT osoba (#PCDATA)>  
<!ELEMENT email (#PCDATA)>
```

Składnia skrócona:

```
element wizytownik {  
    element wizytówka {  
        element osoba { text },  
        element e-mail { text }  
    }*  
}
```

Składnia XML-owa:

```
<element name="wizytownik"  
    xmlns="http://relaxng.org/  
        ns/structure/1.0">  
    <zeroOrMore>  
        <element  
            name="wizytówka">  
            <element name="osoba">  
                <text/>  
            </element>  
            <element name="e-mail">  
                <text/>  
            </element>  
        </element>  
    </zeroOrMore>  
</element>
```

<http://www.relaxng.org>

Dwie składnie RELAX NG

DTD:

```
<!ELEMENT wizytownik  
    (wizytówka*)>  
<!ELEMENT wizytówka  
    (osoba, email)>  
<!ELEMENT osoba (#PCDATA)>  
<!ELEMENT email (#PCDATA)>
```

Składnia skrócona:

```
element wizytownik {  
    element wizytówka {  
        element osoba { text },  
        element e-mail { text }  
    }*  
}
```

Składnia XML-owa:

```
<element name="wizytownik"  
    xmlns="http://relaxng.org/  
        ns/structure/1.0">  
    <zeroOrMore>  
        <element  
            name="wizytówka">  
            <element name="osoba">  
                <text/>  
            </element>  
            <element name="e-mail">  
                <text/>  
            </element>  
        </element>  
    </zeroOrMore>  
</element>
```

<http://www.relaxng.org>

RELAX NG:

```
<element name="wizytownik"
  xmlns="http://relaxng.org/ns/structure/1.0">
  <oneOrMore>
    <element name="wizytówka">
      ...
    </element>
  </oneOrMore>
</element>
```

RELAX NG:

```
<element name="wizytownik"
  xmlns="http://relaxng.org/ns/structure/1.0">
  <oneOrMore>
    <element name="wizytówka">
      ...
    </element>
  </oneOrMore>
</element>
```

DTD:

```
<!ELEMENT wizytownik (wizytówka+)>
```

RELAX NG:

```
<element name="wizytówka">
  <element name="osoba">
    <text/>
  </element>
  <element name="e-mail">
    <text/>
  </element>
  <optional>
    <element name="telefon">
      <text/>
    </element>
  </optional>
</element>
```

RELAX NG:

```
<element name="wizytówka">
  <element name="osoba">
    <text/>
  </element>
  <element name="e-mail">
    <text/>
  </element>
  <optional>
    <element name="telefon">
      <text/>
    </element>
  </optional>
</element>
```

DTD:

```
<!ELEMENT wizytówka (osoba, e-mail, telefon?)>
```

RELAX NG:

```
<element name="wizytówka">
  <choice>
    <element name="osoba">
      <text/>
    </element>
    <group>
      <element name="imię">
        <text/>
      </element>
      <element name="nazwisko">
        <text/>
      </element>
    </group>
  </choice>
</element>
```

RELAX NG:

```
<element name="wizytówka">  
  <choice>  
    <element name="osoba">  
      <text/>  
    </element>  
    <group>  
      <element name="imię">  
        <text/>  
      </element>  
      <element name="nazwisko">  
        <text/>  
      </element>  
    </group>  
  </choice>  
</element>
```

DTD:

```
<!ELEMENT wizytówka (osoba | (imię, nazwisko))>
```

RELAX NG:

```
<element name="osoba">  
  <attribute name="telefon"/>  
</element>
```

Uwagi:

- `<text/>` jest domyślną zawartością atrybutów,
- atrybuty są domyślnie wymagane, IMPLIED wymaga użycia `<optional>`,
- gdy nie ma atrybutów, element pusty oznaczamy jako `<empty/>`.

RELAX NG:

```
<element name="osoba">  
  <attribute name="telefon"/>  
</element>
```

Uwagi:

- `<text/>` jest domyślną zawartością atrybutów,
- atrybuty są domyślnie wymagane, `IMPLIED` wymaga użycia `<optional>`,
- gdy nie ma atrybutów, element pusty oznaczamy jako `<empty/>`.

DTD:

```
<!ELEMENT osoba EMPTY>  
<!ATTLIST osoba telefon CDATA REQUIRED>
```

RELAX NG:

```
<grammar>
  <start>
    <element name="wizytownik">
      <zeroOrMore>
        <element name="wizytówka">
          <ref name="treśćWizytówki"/>
        </element>
      </zeroOrMore>
    </element>
  </start>

  <define name="treśćWizytówki">
    <element name="osoba">
      <text/>
    </element>
    <element name="e-mail">
      <text/>
    </element>
  </define>
</grammar>
```

Kurs RELAX NG w przykładach: fragmenty i *doctype*

RELAX NG:

```
<grammar>
  <start>
    <element name="wizytownik">
      <zeroOrMore>
        <element name="wizytówka">
          <ref name="treśćWizytówki"/>
        </element>
      </zeroOrMore>
    </element>
  </start>

  <define name="treśćWizytówki">
    <element name="osoba">
      <text/>
    </element>
    <element name="e-mail">
      <text/>
    </element>
  </define>
</grammar>
```

DTD:

```
<!DOCTYPE
  wizytownik

  [<!ELEMENT
    wizytownik
    (wizytówka*)>

  <!ENTITY %
    treśćWizytówki
    "osoba, e-mail">

  <!ELEMENT
    wizytówka
    (%treśćWizytówki;)>

  <!ELEMENT
    osoba
    (#PCDATA)>

  <!ELEMENT
    e-mail
```

RELAX NG:

```
<element name="e-mail" datatypeLibrary=  
    "http://www.w3.org/2001/XMLSchema-datatypes">  
  <data type="string">  
    <param name="maxLength">127</param>  
  </data>  
</element>
```

RELAX NG:

```
<element name="e-mail" datatypeLibrary=
    "http://www.w3.org/2001/XMLSchema-datatypes">
  <data type="string">
    <param name="maxLength">127</param>
  </data>
</element>
```

XML Schema:

```
<xsd:element name="e-mail">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="127"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

RELAX NG:

```
<element name="plik">  
  <attribute name="format">  
    <choice>  
      <value>HTML</value>  
      <value>PDF</value>  
    </choice>  
  </attribute>  
</element>
```

W podobny sposób można wyliczać zawartość elementów (jak w XML Schema).

RELAX NG:

```
<element name="plik">  
  <attribute name="format">  
    <choice>  
      <value>HTML</value>  
      <value>PDF</value>  
    </choice>  
  </attribute>  
</element>
```

W podobny sposób można wyliczać zawartość elementów (jak w XML Schema).

DTD:

```
<!ATTLIST plik format (HTML | PDF) #REQUIRED>
```

RELAX NG:

```
<element name="listaParzysta">  
  <list>  
    <oneOrMore>  
      <data type="integer"/>  
      <data type="integer"/>  
    </oneOrMore>  
  </list>  
</element>
```

RELAX NG:

```
<element name="head">
  <interleave>
    <ref name="title"/>
    <zeroOrMore>
      <ref name="meta"/>
    </zeroOrMore>
  </interleave>
</element>

<element name="p">
  <mixed>
    <ref name="b"/>
    <ref name="i"/>
    <ref name="u"/>
  </mixed>
</element>
```

RELAX NG:

```
<element name="head">
  <interleave>
    <ref name="title"/>
    <zeroOrMore>
      <ref name="meta"/>
    </zeroOrMore>
  </interleave>
</element>

<element name="p">
  <mixed>
    <ref name="b"/>
    <ref name="i"/>
    <ref name="u"/>
  </mixed>
</element>
```

DTD dla modelu mieszanego:

```
<!ELEMENT p
  (#PCDATA | b | i | u)*>
```

- `<element name="nazwa">`, `<attribute name="nazwa">`,
- `<optional>` — atrybut lub element opcjonalny (domyślnie atrybuty wymagane),
- `<text/>` — to samo, co `#PCDATA`
- `<empty/>` — element pusty,
- `<zeroOrMore>`, `<oneOrMore>`,
- `<choice>`, `<group>`, `<interleave>` (dowolny porządek),
- `<mixed>`,
- `<grammar>` — element główny gramatyki,
- `<start>` — element główny dokumentu,
- ...

```
<element name="osoba">
  <choice>
    <text/>
    <group>
      <element name="imię">
        <data type="token"/>
      </element>
      <optional>
        <element name="drugieImię">
          <data type="token"/>
        </element>
      </optional>
      <element name="nazwisko">
        <data type="token"/>
      </element>
    </group>
  </choice>
</name>
```

Autorstwa Erica van der Vlista, prace rozpoczęte w 2001 r.,
zamary w 2003 r.

Idea: „definicja przez przykład” – egzemplarze dokumentów
definiują schemat.

Autorstwa Erica van der Vlista, prace rozpoczęte w 2001 r., zamaryły w 2003 r.

Idea: „definicja przez przykład” – egzemplarze dokumentów definiują schemat.

Jak to działa? Przekształcenie XSLT zamienia egzemplarze na „prawdziwy” schemat RELAX NG.

Autorstwa Erica van der Vlista, prace rozpoczęte w 2001 r., zamaryły w 2003 r.

Idea: „definicja przez przykład” – egzemplarze dokumentów definiują schemat.

Jak to działa? Przekształcenie XSLT zamienia egzemplarze na „prawdziwy” schemat RELAX NG.

Obsługuje w prosty sposób 80% przypadków:

- definicja elementów i atrybutów,
- kontrola nad liczbą wystąpień elementów,
- zawartość mieszana,
- predefiniowane typy proste XML Schema.

<http://examplotron.org>

Examplotron: przykład

Examplotron:

```
<osoba>
  <imię>Petronela</imię>
  <imię>Zenobia</imię>
</osoba>
```

Relax NG:

```
<grammar xmlns:ega="http://examplotron.org/annotations/"
  xmlns="http://relaxng.org/ns/structure/1.0"
  xmlns:sch="http://www.ascc.net/xml/schematron"
  datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">
  <start>
    <element name="osoba">
      <oneOrMore>
        <element name="imię">
          <text><ega:example>Petronela</ega:example>/text>
        </element>
      </oneOrMore>
    </element>
  </start>
</grammar>
```

Examplotron: przykład

Examplotron:

```
<zamówienie nr="123" eg:content="eg:interleave"  
    xmlns:eg="http://examplotron.org/0/">  
  <produkt>nakręcana mysz</produkt>  
  <cena>10,99</price>  
</zamówienie>
```

Relax NG:

```
<element name='zamówienie'>  
  <interleave>  
    <optional>  
      <attribute name='nr'>  
        <data type='integer'><ega:example nr='123' /></data>  
      </attribute>  
    </optional>  
    <element name='produkt'>  
      <text><ega:example>nakręcana mysz</ega:example></text>  
    </element>  
    ...  
  </interleave>  
</element>
```

Na następnych wykładach:

- 5 XI: Patryk Czarnik o tym, jak prezentować XML-a,
- 12 XI: Patryk Czarnik o standardach XPath/XQuery,
- 19 XI: Patryk Czarnik o XSLT,
- 26 XI: ja o zastosowaniach XML-a (XML DocBook, SVG, MathML i przede wszystkim OOXML vs. OpenDocument).