

# XML i nowoczesne metody zarządzania treścią

Wykład 4: Jeszcze o XML Schema

Maciej Ogrodniczuk

MIMUW, 22 października 2009

Powiązanie schematu z dokumentem składa się z trzech elementów:

Powiązanie schematu z dokumentem składa się z trzech elementów:

- deklaracji przestrzeni nazw dla egzemplarza dokumentu zgodnego z XML Schema:  
`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance",`

Powiązanie schematu z dokumentem składa się z trzech elementów:

- deklaracji przestrzeni nazw dla egzemplarza dokumentu zgodnego z XML Schema:  
`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`,
- powiązania schematu dla elementów nie należących do żadnej przestrzeni nazw — poprzez podanie URL-a schematu w atrybucie `xsi:noNamespaceSchemaLocation`,

Powiązanie schematu z dokumentem składa się z trzech elementów:

- deklaracji przestrzeni nazw dla egzemplarza dokumentu zgodnego z XML Schema:  
`xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"`,
- powiązania schematu dla elementów nie należących do żadnej przestrzeni nazw — poprzez podanie URL-a schematu w atrybucie `xsi:noNamespaceSchemaLocation`,
- ew. powiązań listy używanych przestrzeni nazw z URL-ami schematów mających posłużyć do walidacji elementów, których nazwy należą do używanych w dokumencie przestrzeni nazw — w atrybucie `xsi:schemaLocation`.

```
<?xml version="1.0"?>  
<tekst xmlns:xsi="http://www.w3.org/2001/  
        XMLSchema-instance"  
        xsi:noNamespaceSchemaLocation="tekst.xsd"  
        xsi:schemaLocation="http://www.example.org/wzory  
                            wzory.xsd  
                            http://www.example.org/wykresy  
                            wykresy.xsd">  
    ...  
</tekst>
```

```
<?xml version="1.0"?>  
<tekst xmlns:xsi="http://www.w3.org/2001/  
        XMLSchema-instance"  
        xsi:noNamespaceSchemaLocation="tekst.xsd"  
        xsi:schemaLocation="http://www.example.org/wzory  
                            wzory.xsd  
                            http://www.example.org/wykresy  
                            wykresy.xsd">  
    ...  
</tekst>
```

```
<?xml version="1.0"?>  
<tekst xmlns:xsi="http://www.w3.org/2001/  
        XMLSchema-instance"  
        xsi:noNamespaceSchemaLocation="tekst.xsd"  
        xsi:schemaLocation="http://www.example.org/wzory  
                            wzory.xsd  
                            http://www.example.org/wykresy  
                            wykresy.xsd">  
    ...  
</tekst>
```

```
<?xml version="1.0"?>  
<tekst xmlns:xsi="http://www.w3.org/2001/  
          XMLSchema-instance"  
      xsi:noNamespaceSchemaLocation="tekst.xsd"  
      xsi:schemaLocation="http://www.example.org/wzory  
                          wzory.xsd  
                          http://www.example.org/wykresy  
                          wykresy.xsd">  
  ...  
</tekst>
```

Wielopoziomowa walidacja:

Wielopoziomowa walidacja:

- 1 sprawdź (kaskadowo), że dokumenty schematu są poprawne strukturalnie (zgodne ze schematem dla XML Schema),

Wielopoziomowa walidacja:

- 1 sprawdź (kaskadowo), że dokumenty schematu są poprawne strukturalnie (zgodne ze schematem dla XML Schema),
- 2 sprawdź, że dokument jest zgodny z regułami opisanymi w schemacie.

Jeśli chcemy, by nazwy elementów, atrybutów i typów zdefiniowanych w dokumencie schematu należały do określonej przestrzeni nazw, musimy ją określić w atrybucie `targetNamespace` elementu głównego `<xsd:schema>`.

Brak tego atrybutu oznacza, że nazwy komponentów wynikowych nie będą należeć do żadnej przestrzeni nazw.

## Dobrze:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
>
  <xsd:complexType name="typBazowy">
    ...
  </xsd:complexType>
  <xsd:complexType name="typPochodny">
    <xsd:complexContent>
      <xsd:restriction base="typBazowy">
        ...
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="element" type="typPochodny">
</xsd:schema>
```

## Za mało!

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.example.org/test"
            >
  <xsd:complexType name="typBazowy">
    ...
  </xsd:complexType>
  <xsd:complexType name="typPochodny">
    <xsd:complexContent>
      <xsd:restriction base="typBazowy">
        ...
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="element" type="typPochodny">
</xsd:schema>
```

## Dobrze:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.example.org/test"
            xmlns="http://www.example.org/test">
  <xsd:complexType name="typBazowy">
    ...
  </xsd:complexType>
  <xsd:complexType name="typPochodny">
    <xsd:complexContent>
      <xsd:restriction base="typBazowy">
        ...
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="element" type="typPochodny">
</xsd:schema>
```

## Dobrze:

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.example.org/test"
            xmlns:typ="http://www.example.org/test">
  <xsd:complexType name="typBazowy">
    ...
  </xsd:complexType>
  <xsd:complexType name="typPochodny">
    <xsd:complexContent>
      <xsd:restriction base="typ:typBazowy">
        ...
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
  <xsd:element name="element" type="typ:typPochodny">
</xsd:schema>
```

Nazwy **kwalifikowane** (ang. *qualified*) należą do pewnej przestrzeni nazw. Mogą być poprzedzone prefiksem lub należeć do domyślnej przestrzeni nazw.

Nazwy **niekwalifikowane** (ang. *unqualified*) nie należą do żadnej.

Autor schematu może zdecydować, czy w dokumentach elementy i atrybuty zdefiniowane lokalnie muszą być kwalifikowane czy nie:

- na poziomie schematu — z użyciem atrybutów `elementFormDefault` i `attributeFormDefault` o wartościach `qualified` lub `unqualified` (domyślnie),
- na poziomie lokalnej definicji — atrybutem `form` o takiej samej zawartości.

# Lokalne nazwy niekwalifikowane: przykład

## Schemat:

```
<xsd:complexType name="typOsoba">
  <xsd:sequence>
    <xsd:element name="imię" type="xsd:token"
      minOccurs="0" maxOccurs="2"/>
    <xsd:element name="nazwisko" type="xsd:token"/>
  </xsd:sequence>
  <xsd:attribute name="pesel"/>
</xsd:complexType>
<xsd:element name="osoba" type="typOsoba"/>
```

## Dokument:

```
<os:osoba xmlns:os="http://www.example.org/osoby"
  pesel="04250101234">
  <imię>Jan</imię>
  <nazwisko>Kowalski</nazwisko>
</os:osoba>
```

# Lokalne nazwy kwalifikowane: przykład

## Schemat:

```
<xsd:schema ... elementFormDefault="qualified"  
              attributeFormDefault="qualified">
```

## Dokument 1:

```
<os:osoba xmlns:os="http://www.example.org/osoby"  
          os:pesel="26511756789">  
  <os:nazwisko>Nowopolski</os:nazwisko>  
</os:osoba>
```

## Dokument 2:

```
<osoba xmlns="http://www.example.org/osoby"  
      >  
  <nazwisko>Traugutt</nazwisko>  
</osoba>
```

## Schemat:

```
<xsd:schema ... elementFormDefault="qualified"  
              attributeFormDefault="qualified">
```

## Dokument 1:

```
<os:osoba xmlns:os="http://www.example.org/osoby"  
          os:pesel="26511756789">  
  <os:nazwisko>Nowopolski</os:nazwisko>  
</os:osoba>
```

## Dokument 2:

```
<osoba xmlns="http://www.example.org/osoby"  
       xmlns:osoby="http://www.example.org/osoby"  
       osoby:pesel="63812224680">  
  <nazwisko>Traugutt</nazwisko>  
</osoba>
```

# Schematy (dokumentów) i dokumenty schematów

Schemat (struktura logiczna) może być zapisany w wielu dokumentach schematów (plikach .xsd).

Specyfikacja XML Schema określa trzy metody łączenia dokumentów schematów:

- include,
- import,
- redefine,

Lokalizacje dokumentów opisujących schemat są określone w egzemplarzu, a ponadto:

- procesor może używać dokumentów schematów z predefiniowanych lokalizacji,
- lokalizacje dokumentów schematów mogą być przekazywane jako parametry wiersza poleceń.

# Modularyzacja schematów metodą `<xsd:include>`

Metoda `include` dołącza dokument schematu do docelowej przestrzeni nazw głównego dokumentu schematu.

# Modularyzacja schematów metodą `<xsd:include>`

Metoda `include` dołącza dokument schematu do docelowej przestrzeni nazw głównego dokumentu schematu.

Dołączany dokument musi mieć taką samą docelową przestrzeń nazw jak dokument główny lub nie mieć w ogóle docelowej przestrzeni nazw.

# Modularyzacja schematów metodą `<xsd:include>`

Metoda `include` dołącza dokument schematu do docelowej przestrzeni nazw głównego dokumentu schematu.

Dołączany dokument musi mieć taką samą docelową przestrzeń nazw jak dokument główny lub nie mieć w ogóle docelowej przestrzeni nazw.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="http://www.example.org/ceny"
            targetNamespace="http://www.example.org/ceny">
  <xsd:include schemaLocation="produkty.xsd"/>
  ...
</xsd:schema>
```

# Modularyzacja schematów metodą `<xsd:include>`

Metoda `include` dołącza dokument schematu do docelowej przestrzeni nazw głównego dokumentu schematu.

Dołączany dokument musi mieć taką samą docelową przestrzeń nazw jak dokument główny lub nie mieć w ogóle docelowej przestrzeni nazw.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns="http://www.example.org/ceny"
            targetNamespace="http://www.example.org/ceny">
  <xsd:include schemaLocation="produkty.xsd"/>
  ...
</xsd:schema>
```

Uwaga: dołączane schematy nie muszą być kompletne:

- **źle**, bo musimy pilnować zależności między schematami,
- **dobrze**, bo możemy parametryzować schematy (np. definiować różne wersje typów dla elementów o danych nazwach).

## <xsd:redefine> i <xsd:import>

Metoda `redefine` łączy dokument schematu do docelowej przestrzeni nazw głównego dokumentu schematu z możliwością przedefiniowania komponentów:

```
<xsd:redefine schemaLocation="redefine.xsd"/>
```

## <xsd:redefine> i <xsd:import>

Metoda `redefine` dołącza dokument schematu do docelowej przestrzeni nazw głównego dokumentu schematu z możliwością przedefiniowania komponentów:

```
<xsd:redefine schemaLocation="redefine.xsd"/>
```

Metoda `import` dołącza dokument schematu z innej przestrzeni nazw:

```
<xsd:import schemaLocation="import.xsd"  
            namespace="http://www.example.org/firmy"/>
```

## <xsd:import>: przykład z życia

```
<xsd:import namespace="http://www.w3.org/1999/xhtml"
            schemaLocation="http://www.w3.org/2002/08/
                            xhtml/xhtml1-strict.xsd"/>
```

## <xsd:import>: przykład z życia

```
<xsd:import namespace="http://www.w3.org/1999/xhtml"
            schemaLocation="http://www.w3.org/2002/08/
                            xhtml/xhtml1-strict.xsd"/>
<xsd:complexType name="typKodXHTMLowy">
  <xsd:sequence>
    <xsd:any namespace="http://www.w3.org/1999/xhtml"
             processContents="skip"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="xhtml" type="typKodXHTMLowy">
```

## <xsd:import>: przykład z życia

```
<xsd:import namespace="http://www.w3.org/1999/xhtml"
            schemaLocation="http://www.w3.org/2002/08/
                            xhtml/xhtml1-strict.xsd"/>
<xsd:complexType name="typKodXHTMLowy">
  <xsd:sequence>
    <xsd:any namespace="http://www.w3.org/1999/xhtml"
             processContents="skip"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="xhtml" type="typKodXHTMLowy">


---


<xhtml xsi:noNamespaceSchemaLocation="test2.xsd"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:html="http://www.w3.org/1999/xhtml">
  <html:body> ... </html:body>
</xhtml>
```

Pamiętamy przykład definicji umożliwiającej użycie dowolnych elementów z danej przestrzeni nazw:

```
<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>
```

Atrybut `namespace` może też zawierać listę wartości lub przyjmować wartości specjalne:

- `##any` — elementy mogą należeć do dowolnej przestrzeni nazw lub nie należeć do żadnej przestrzeni nazw,
- `##other` — elementy mogą należeć do dowolnej przestrzeni nazw, z wyjątkiem docelowej przestrzeni nazw dokumentu schematu; jeśli dokument schematu nie ma docelowej przestrzeni nazw, elementy zastępujące mogą należeć do dowolnej przestrzeni nazw, ale nie mogą nie należeć do żadnej przestrzeni nazw.
- `##targetNamespace` — elementy mogą należeć do docelowej przestrzeni nazw dokumentu schematu,
- `##local` — elementy mogą nie należeć do żadnej przestrzeni nazw.

Koncepcja wartości niezdefiniowanych (ang. *nil values*) umożliwia zapis informacji o nieokreśloności konstrukcji wyrażonej danym elementem XML-owym.

Koncepcja wartości niezdefiniowanych (ang. *nil values*) umożliwia zapis informacji o nieokreśloności konstrukcji wyrażonej danym elementem XML-owym.

Użycie:

- 1. Możliwość wystąpienia wartości nieokreślonej zapisuje się w schemacie oznaczając element atrybutem `nillable="true"`.

Koncepcja wartości niezdefiniowanych (ang. *nil values*) umożliwia zapis informacji o nieokreśloności konstrukcji wyrażonej danym elementem XML-owym.

Użycie:

1. Możliwość wystąpienia wartości nieokreślonej zapisuje się w schemacie oznaczając element atrybutem `nillable="true"`.
2. Tak oznaczony element będzie mógł być w dokumencie opatrywany specjalnym atrybutem `xsi:nil` (z przestrzeni nazw dla egzemplarza dokumentu — <http://www.w3.org/2001/XMLSchema-instance>) o wartości `true`, co będzie odpowiadać wartości nieokreślonej.

## Schemat:

```
<xsd:element name="książka">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="autor" nillable="true">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="imię-i-nazwisko"/>
            <xsd:element name="data-urodzenia"/>
            <xsd:element name="data-śmierci"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="tytuł"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

## Użycie w dokumencie:

```
<książka xsi:noNamespaceSchemaLocation="book.xsd"
          xmlns:xsi="http://www.w3.org/2001/
                    XMLSchema-instance">
  <autor xsi:nil="true"/>
  <tytuł>Biblia</tytuł>
</książka>
```

### Użycie w dokumencie:

```
<książka xsi:noNamespaceSchemaLocation="book.xsd"
          xmlns:xsi="http://www.w3.org/2001/
                    XMLSchema-instance">
  <autor xsi:nil="true"/>
  <tytuł>Biblia</tytuł>
</książka>
```

### Uwagi:

- Element o wartości nieokreślonej musi mieć zawartość pustą.

## Użycie w dokumencie:

```
<książka xsi:noNamespaceSchemaLocation="book.xsd"
          xmlns:xsi="http://www.w3.org/2001/
                    XMLSchema-instance">
  <autor xsi:nil="true"/>
  <tytuł>Biblia</tytuł>
</książka>
```

## Uwagi:

- Element o wartości nieokreślonej musi mieć zawartość pustą.
- Nieokreśloność jest **ważniejsza niż zdefiniowany model zawartości**.

## Użycie w dokumencie:

```
<książka xsi:noNamespaceSchemaLocation="book.xsd"
          xmlns:xsi="http://www.w3.org/2001/
                    XMLSchema-instance">
  <autor xsi:nil="true"/>
  <tytuł>Biblia</tytuł>
</książka>
```

## Uwagi:

- Element o wartości nieokreślonej musi mieć zawartość pustą.
- Nieokreśloność jest **ważniejsza niż zdefiniowany model zawartości**.
- Atrybuty elementu o wartości nieokreślonej muszą być w każdym wypadku zgodne z modelem.

## Dobre praktyki: sposoby reprezentacji wartości pustej

Dla atrybutów wartość pustą można reprezentować na jeden sposób:

## Dobre praktyki: sposoby reprezentacji wartości pustej

Dla atrybutów wartość pustą można reprezentować na jeden sposób: `use="optional"`.

Dla atrybutów wartość pustą można reprezentować na jeden sposób: `use="optional"`.

Dla elementów mamy więcej sposobów reprezentacji wartości pustej, np.:

- 1 brak elementu,
- 2 element pusty,
- 3 element nieokreślony.

Dla atrybutów wartość pustą można reprezentować na jeden sposób: `use="optional"`.

Dla elementów mamy więcej sposobów reprezentacji wartości pustej, np.:

- 1 brak elementu,
- 2 element pusty,
- 3 element nieokreślony.

Dany element może więc mieć następujący model:

```
<xsd:element name="pojawiam-się-i-znikam"
```

```
>
```

```
</xsd:element>
```

Dla atrybutów wartość pustą można reprezentować na jeden sposób: `use="optional"`.

Dla elementów mamy więcej sposobów reprezentacji wartości pustej, np.:

- 1 brak elementu,
- 2 element pusty,
- 3 element nieokreślony.

Dany element może więc mieć następujący model:

```
<xsd:element name="pojawiam-się-i-znikam"  
             minOccurs="0"  
             >
```

```
</xsd:element>
```

Dla atrybutów wartość pustą można reprezentować na jeden sposób: `use="optional"`.

Dla elementów mamy więcej sposobów reprezentacji wartości pustej, np.:

- 1 brak elementu,
- 2 `element pusty`,
- 3 `element nieokreślony`.

Dany element może więc mieć następujący model:

```
<xsd:element name="pojawiam-się-i-znikam"
              minOccurs="0"
              >
  <xsd:complexType/>
</xsd:element>
```

Dla atrybutów wartość pustą można reprezentować na jeden sposób: `use="optional"`.

Dla elementów mamy więcej sposobów reprezentacji wartości pustej, np.:

- 1 brak elementu,
- 2 element pusty,
- 3 element nieokreślony.

Dany element może więc mieć następujący model:

```
<xsd:element name="pojawiam-się-i-znikam"
             minOccurs="0"
             nillable="true">
  <xsd:complexType/>
</xsd:element>
```

## Przykład:

```
<tablica>  
  <napis>Ala ma żółwia</napis>  
  <napis xsi:nil="true"/>  
  <napis/>  
  <napis xsi:nil="true"/>  
</tablica>
```

## Przykład:

```
<tablica>  
  <napis>Ala ma żółwia</napis>  
  <napis xsi:nil="true"/>  
  <napis/>  
  <napis xsi:nil="true"/>  
</tablica>
```

## Komentarz:

- napis pusty (o zerowej długości) można reprezentować elementem pustym lub brakiem elementu,

## Przykład:

```
<tablica>
  <napis>Ala ma żółwia</napis>
  <napis xsi:nil="true"/>
  <napis/>
  <napis xsi:nil="true"/>
</tablica>
```

## Komentarz:

- napis pusty (o zerowej długości) można reprezentować elementem pustym lub brakiem elementu,
- napis o nieokreślonej wartości można reprezentować wartością `nil` lub brakiem elementu,

## Przykład:

```
<tablica>
  <napis>Ala ma żółwia</napis>
  <napis xsi:nil="true"/>
  <napis/>
  <napis xsi:nil="true"/>
</tablica>
```

## Komentarz:

- napis pusty (o zerowej długości) można reprezentować elementem pustym lub brakiem elementu,
- napis o nieokreślonej wartości można reprezentować wartością `nil` lub brakiem elementu,
- jeśli jednak definiujemy strukturę, w której wystąpienie elementu ma znaczenie (np. dla jej budowy, rozmiaru), brak elementu okazuje się złym reprezentantem czegokolwiek.

- Typy proste (ang. *simple*) i złożone (ang. *complex*):
  - proste — bez struktury elementowej ani atrybutowej,
  - złożone — mogą mieć zawartość elementową i wprowadzać atrybuty.
- Typy jednostkowe (ang. *atomic*) i wielowartościowe (listy i unie):
  - jednostkowe — zawierające niepodzielne wartości,
  - listy — skończone (ew. puste) sekwencje wartości jednostkowych,
  - unie — złożenia typów.
- Typy bazowe (ang. *ur-types*), pierwotne (ang. *primitive*) i pochodne (ang. *derived*):
  - bazowe — „pra-typy”,
  - pierwotne — istniejące „od zawsze”, z rozłącznymi przestrzeniami wartości,
  - pochodne — wywiedzione od innych typów poprzez ograniczenie, stworzenie listy lub unii.

- Typy wbudowane (ang. *built-in*) i zdefiniowane przez użytkownika (ang. *user-derived*):
  - wbudowane — zdefiniowane w specyfikacji,
  - zdefiniowane przez użytkownika — na potrzeby budowy schematu.
- Typy nazwane i anonimowe:
  - nazwane — globalne,
  - anonimowe — o zasięgu lokalnym.

# Typ złożony o zawartości prostej

Problem:

Chcemy zdefiniować element z zawartością tekstową i atrybutem.

## Problem:

Chcemy zdefiniować element z zawartością tekstową i atrybutem.

## Rozwiązanie:

```
<xsd:element name="liczba-słownie">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="wartość"
                      type="xsd:positiveInteger"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

# Rozszerzanie typów (ang. *derivation by extension*)

Rozszerzanie:

Rozszerzanie:

- zawartości prostej: dodawanie atrybutów do typu prostego lub typu złożonego o zawartości prostej,

## Rozszerzanie:

- zawartości prostej: dodawanie atrybutów do typu prostego lub typu złożonego o zawartości prostej,
- zawartości złożonej: dodawanie do typu bazowego dodatkowych elementów lub atrybutów.

Rozszerzanie:

- zawartości prostej: dodawanie atrybutów do typu prostego lub typu złożonego o zawartości prostej,
- zawartości złożonej: dodawanie do typu bazowego dodatkowych elementów lub atrybutów.

Dwie uwagi:

- wartości typu bazowego nie muszą być poprawnymi wartościami typu wyprowadzonego (rozszerzenie może np. dodawać elementy lub atrybuty, które są wymagane),

## Rozszerzanie:

- zawartości prostej: dodawanie atrybutów do typu prostego lub typu złożonego o zawartości prostej,
- zawartości złożonej: dodawanie do typu bazowego dodatkowych elementów lub atrybutów.

## Dwie uwagi:

- wartości typu bazowego nie muszą być poprawnymi wartościami typu wyprowadzonego (rozszerzenie może np. dodawać elementy lub atrybuty, które są wymagane),
- definiując rozszerzenie zawartości złożonej, nie trzeba powtarzać modelu zawartości typu bazowego — procesor doda nowy model po modelu zawartości typu bazowego, jak gdyby oba modele były zawarte w grupie `sequence`.

```
<xsd:complexType name="typPublikacja">
  <xsd:sequence>
    <xsd:element name="tytuł" maxOccurs="unbounded"/>
    <xsd:element name="autor" maxOccurs="unbounded"/>
    <xsd:element name="rokPubl" type="xsd:year"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="typKsiążka">
  <xsd:complexContent>
    <xsd:extension base="typPublikacja">
      <xsd:sequence>
        <xsd:element name="ISBN"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

## Ograniczanie zawartości:

- dla zawartości prostej — za pomocą aspektów,
- dla zawartości złożonej — poprzez:
  - ograniczanie liczebności (`minOccurs`, `maxOccurs`),
  - usunięcie elementów opcjonalnych w grupach `sequence` i `all`,
  - wybranie podzbioru elementów w grupie `choice`,
  - ograniczenie typu poszczególnych podelementów.

## Ograniczanie atrybutów:

- ograniczenie typu atrybutu,
- ograniczanie wystąpienia atrybutu (z opcjonalnego na wymagany lub zabroniony),
- dodanie, zmiana lub usunięcie wartości domyślnej,
- dodanie wartości stałej.

```
<xsd:complexType name="typPrzedziałCzasu">
  <xsd:simpleContent>
    <xsd:extension base="xsd:positiveInteger">
      <xsd:attribute name="jednostka"/>
      <xsd:attribute name="milisekund"
                    type="xsd:positiveInteger"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

# Ograniczanie zawartości prostej i atrybutów: przykład

```
<xsd:complexType name="typWiekWina">
  <xsd:simpleContent>
    <xsd:restriction base="typPrzedziałCzasu">
      <xsd:maxInclusive value="300"/>
      <xsd:attribute name="jednostka" fixed="rok"/>
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>
```

# Ograniczanie zawartości prostej i atrybutów: przykład

```
<xsd:complexType name="typWiekWina">
  <xsd:simpleContent>
    <xsd:restriction base="typPrzedziałCzasu">
      <xsd:maxInclusive value="300"/>
      <xsd:attribute name="jednostka" fixed="rok"/>
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="typLiczbaCykli">
  <xsd:simpleContent>
    <xsd:restriction base="typPrzedziałCzasu">
      <xsd:attribute name="jednostka"
        fixed="cykli procesora"/>
      <xsd:attribute name="milisekund" use="prohibited"/>
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>
```

# Ograniczanie zawartości prostej i atrybutów: przykład

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  ...
  <xsd:element name="okres" type="typPrzedziałCzasu"/>
  <xsd:element name="wiekWina" type="typWiekWina"/>
  <xsd:element name="czasWykonaniaInstrukcji"
    type="typLiczbaCykli"/>
  ...
</xsd:schema>
```

# Ograniczanie zawartości prostej i atrybutów: przykład

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  ...
  <xsd:element name="okres" type="typPrzedziałCzasu"/>
  <xsd:element name="wiekWina" type="typWiekWina"/>
  <xsd:element name="czasWykonaniaInstrukcji"
    type="typLiczbaCykli"/>
  ...
</xsd:schema>
```

---

```
<okres jednostka="mrugnięcia okiem"
  milisekund="700">2</okres>
<wiekWina milisekund="9460800000">3</wiekWina>
<czasWykonaniaInstrukcji>5</czasWykonaniaInstrukcji>
```

## Ograniczanie zawartości złożonej: przykład

```
<xsd:complexType name="typPublikacja">
  <xsd:sequence>
    <xsd:element name="tytuł"/>
    <xsd:element name="autor" minOccurs="0"
                  maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="typPublikacjaJednegoAutora">
  <xsd:complexContent>
    <xsd:restriction base="typPublikacja">
      <xsd:sequence>
        <xsd:element name="tytuł"/>
        <xsd:element name="autor"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
```

```
<xsd:complexType name="typPublikacja">
  <xsd:sequence>
    <xsd:element name="tytuł" maxOccurs="unbounded"/>
    <xsd:element name="autor" minOccurs="0" maxOccurs="3"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="typPublikacjaBezAutora">
  <xsd:complexContent>
    <xsd:restriction base="typPublikacja">
      <xsd:sequence>
        <xsd:element name="tytuł" maxOccurs="unbounded"/>
        <xsd:element name="autor" minOccurs="0"
                    maxOccurs="0"/>
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
```

```
<xsd:simpleType name="typDwucyfrowy">  
  <xsd:restriction base="xsd:string">  
    <xsd:pattern value="\d{2}"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

```
<xsd:element name="elementTrzycyfrowy">  
  <xsd:simpleType>  
    <xsd:restriction base="typDwucyfrowy">  
      <xsd:pattern value="\d{3}"/>  
    </xsd:restriction>  
  </xsd:simpleType>  
</xsd:element>
```

## Ograniczanie i zawężanie jednocześnie

Chcemy zdefiniować element `<kodPocztowy>` o wzorcu `XX-XXX` i stałym atrybucie `PL` dla strefy kodowej.

## Ograniczanie i zawężanie jednocześnie

Chcemy zdefiniować element `<kodPocztowy>` o wzorcu `XX-XXX` i stałym atrybucie `PL` dla strefy kodowej.

Jednocześnie się nie da, trzeba w dwóch krokach:

# Ograniczanie i zawężanie jednocześnie

Chcemy zdefiniować element `<kodPocztowy>` o wzorcu `XX-XXX` i stałym atrybucie `PL` dla strefy kodowej.

Jednocześnie się nie da, trzeba w dwóch krokach:

```
<xsd:simpleType name="typKodPocztowy">  
  <xsd:restriction base="xsd:string">  
    <xsd:pattern value="\d{2}-\d{3}"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

# Ograniczanie i zawężanie jednocześnie

Chcemy zdefiniować element <kodPocztowy> o wzorcu XX-XXX i stałym atrybucie PL dla strefy kodowej.

Jednocześnie się nie da, trzeba w dwóch krokach:

```
<xsd:simpleType name="typKodPocztowy">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{2}-\d{3}"/>
  </xsd:restriction>
</xsd:simpleType>
<xsd:complexType name="typKodPocztowyPolski">
  <xsd:simpleContent>
    <xsd:extension base="typKodPocztowy">
      <xsd:attribute name="strefa" fixed="PL"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```

Dwa atrybuty:

- `final` — do kontroli nad tworzeniem,
- `block` — do kontroli nad użyciem.

Trzy wartości obu atrybutów:

- `extension` — zabraniamy rozszerzania,
- `restriction` — zabraniamy ograniczania,
- `#all` — zabraniamy rozszerzania i ograniczania.

Globalna kontrola nad całym schematem:

- atrybutem `finalDefault` — jak byśmy podali wartość z atrybutu przy każdym komponencie schematu,
- atrybutem `blockDefault` — j.w.

Użycie:

```
<xsd:complexType name="typPublikacja" final="#all">  
<xsd:schema ... finalDefault="restriction">
```

W przypadku typów prostych można zabronić zmiany wartości aspektów używając atrybutu `fixed`:

```
<simpleType name="typKodPocztowy">  
  <restriction base="string">  
    <length value="7" fixed="true"/>  
  </restriction>  
</simpleType>
```

Korzystając z hierarchii typów możemy elastyczniej budować dokumenty:

- przypisując elementowi typ bazowy w schemacie,
- korzystając z typu wyprowadzonego w dokumencie.

Korzystając z hierarchii typów możemy elastyczniej budować dokumenty:

- przypisując elementowi typ bazowy w schemacie,
- korzystając z typu wyprowadzonego w dokumencie.

Użycie typu wyprowadzonego wymaga jego jawnego wskazania w atrybucie `xsi:type` z przestrzeni nazw dla egzemplarzy dokumentów `http://www.w3.org/2001/XMLSchema-instance`.

Korzystając z hierarchii typów możemy elastyczniej budować dokumenty:

- przypisując elementowi typ bazowy w schemacie,
- korzystając z typu wyprowadzonego w dokumencie.

Użycie typu wyprowadzonego wymaga jego jawnego wskazania w atrybucie `xsi:type` z przestrzeni nazw dla egzemplarzy dokumentów `http://www.w3.org/2001/XMLSchema-instance`.

Uwaga: typy bazowe mogą być jawnie oznaczone w schemacie jako abstrakcyjne (ustawiając wartość atrybutu `abstract` jako `true`) i wówczas w dokumencie musi zostać użyty – i wskazany atrybutem `xsd:type` – któryś z typów pochodnych).

# Użycie typów wyprowadzonych: przykład

## Schemat:

```
<xsd:complexType name="typAdres">
  <xsd:sequence>
    <xsd:element name="nazwa" type="xsd:string"/>
    <xsd:element name="miasto" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="typAdresUSA">
  <xsd:complexContent>
    <xsd:extension base="typAdres">
      <xsd:sequence>
        <xsd:element name="stan" type="typStanyUSA"/>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```

# Użycie typów wyprowadzonych: przykład

## Schemat:

```
<xsd:element name="adres" type="typAdres"/>
```

## Dokument:

```
<adres>  
  <nazwa>Jan Kowalski</nazwa>  
  <miasto>Warszawa</miasto>  
</adres>
```

---

```
<adres xsi:type="typAdresUSA">  
  <nazwa>George W. Bush</nazwa>  
  <miasto>Dallas</miasto>  
  <stan>Texas</stan>  
</adres>
```