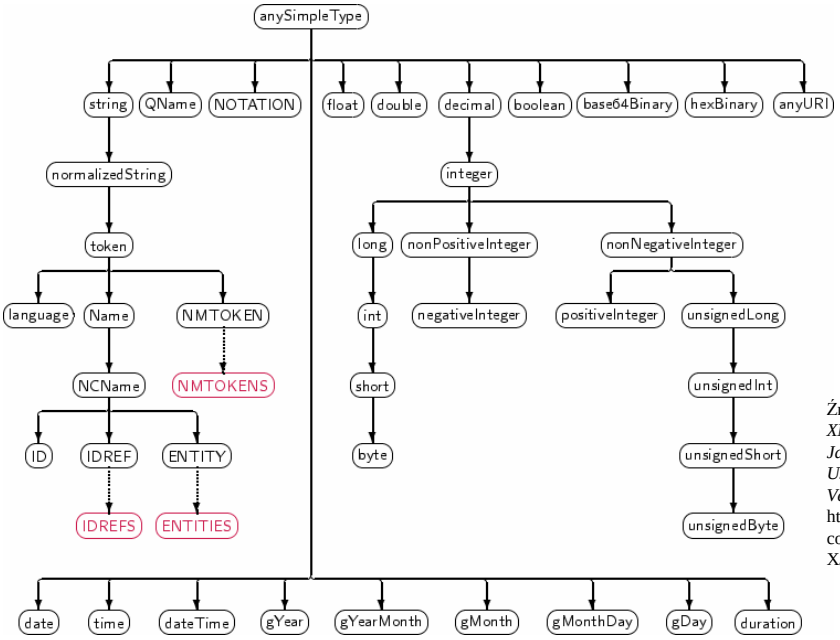
Definiowanie typów dokumentów

Część 3. XML Schema

Wbudowane typy proste



Źródło:
XML Enhancements to Java™ (XJ). User Manual for Release Version 1.0
http://www.research.ibm.com/xj/doc/XJmanual/XJmanual.html

Typy wbudowane tworzą hierarchię – niektóre typy są wyprowadzone z innych, bardziej ogólnych. Pierwszy poziom hierarchii (bezpośrednio pod anySimpleType) to *typy pierwotne* (ang. *primitive types*). Reprezentują one podstawowe abstrakcyjne typy danych. Pozostałe typy nazywamy *wyprowadzonymi typami wbudowanymi* (ang. *derived built-in types*). Są one ograniczeniami typów pierwotnych lub listami.

Wbudowane typy proste (wybrane)

- `string` – napis
- `normalizedString` – napis, w którym każdy biały znak jest podczas przetwarzania zastępowany przez spację
- `token` – napis, w którym każdy ciąg białych znaków jest podczas przetwarzania zastępowany przez jedną spację, zaś białe znaki na początku i końcu są usuwane
- `QName` – nazwa kwalifikowana
- `NCName` – nazwa bez dwukropka
- `base64Binary` – dane binarne zapisane w kodowaniu Base64
- `hexBinary` – dane binarne zapisane szesnastkowo

Typy danych `string`, `normalizedString` i `token` reprezentują ciągi znaków złożone z dowolnych znaków standardu Unicode. Jedyna różnica między tymi typami polega na sposobie traktowania białych znaków.

`QName` – nazwa kwalifikowana – to nazwa należąca do pewnej przestrzeni nazw. Jak pamiętamy, niekoniecznie musi ona zawierać prefiks przestrzeni nazw i dwukropek – nazwa bez prefiksu także może być kwalifikowana, jeśli jest w zasięgu deklaracji domyślnej przestrzeni nazw.

Wbudowane typy proste (wybrane)

Typ	*Poprawne wartości*	
`decimal`	`12.3`	`+000012.300`
`float`, `double`	`+24.3e-3` `NaN`	`12` `-INF`
`QName`	`os:osoba`	`osoba`
`date`	`1968-04-02` `1968-04-02Z`	`1968-04-02-05:00` `-0045-02-02`
`time`	`13:20:00.887`	`13:30:00-05:00`
`dateTime`	`1968-04-02T13:20:00.887`	
`gYearMonth`	`1968-04`	
`gMonthDay`	`--04-02`	
`duration`	`P2Y6M5DT12H35M30S`	

Podczas sprawdzania poprawności dokumentu względem schematu zachodzi czasami konieczność porównywania wartości. Należy pamiętać, że procesor schematów, porównując wartości, nie ogranicza się do porównania napisów, lecz porównuje abstrakcyjne wartości danego typu, reprezentowane przez te napisy. Zatem wartości `12.3` i `+000012.300` typu `decimal` są sobie równe (bo reprezentują tą samą liczbę dziesiętną. Ale te same wartości typu `string` są różne.

Wyprowadzanie typów

- Wyprowadzanie typów prostych:
  - ograniczanie,
  - tworzenie list,
  - tworzenie unii.
- Wyprowadzanie typów złożonych:
  - ograniczanie,
  - rozszerzanie:
    - typów prostych,
    - typów złożonych.

Nowy typ prosty można wyprowadzić wyłącznie z innego typu prostego. Natomiast nowy typ złożony można wyprowadzić z innego typu złożonego lub z typu prostego.

Nie można rozszerzać typu prostego, z wyjątkiem dodawania atrybutów, co powoduje powstanie typu złożonego.

Ograniczanie typów prostych

- Aspekty (ang. *facets*):
  - `minExclusive`, `minInclusive`,
  - `maxExclusive`, `maxInclusive`,
  - `length`, `minLength`, `maxLength`,
  - `totalDigits`, `fractionDigits`,
  - `enumeration`,
  - `pattern`,
  - `whiteSpace`.
- Tylko `pattern` i `enumeration` można użyć wielokrotnie w jednej definicji typu.

Każdy typ prosty jest ograniczeniem (ang. *restriction*) innego typu prostego, nazywanego typem bazowym (ang. *base type*). Przestrzeń wartości typu prostego powstałego w wyniku ograniczenia jest podzbiorem (ograniczeniem) przestrzeni wartości jego typu bazowego.

Ograniczenia definiujemy przy pomocy aspektów (*x* oznacza wartość aspektu):

- `minExclusive`, `minInclusive`, `maxExclusive`, `maxInclusive` – wartość musi być większa / większa lub równa / mniejsza / mniejsza lub równa niż *x*
- `length` – długość wartości musi być równa *x*
- `minLength`, `maxLength` – długość wartości musi być większa lub równa / mniejsza lub równa *x*
- `totalDigits` – liczba znaczących cyfr musi być mniejsza lub równa *x*
- `fractionDigits` – liczba cyfr ułamkowych musi być mniejsza lub równa *x*
- `whiteSpace` – określa sposób traktowania białych znaków
- `enumeration` – *x* jest jedną z poprawnych wartości
- `pattern` – *x* jest wyrażeniem regularnym, do których powinna pasować wartość

Ograniczanie typów prostych – przykłady

```
<xsd:simpleType name="NumerLottoTyp">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="1"/>
    <xsd:maxInclusive value="49"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="DokumentTyp">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="dowód osobisty"/>
    <xsd:enumeration value="paszport"/>
  </xsd:restriction>
</xsd:simpleType>

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

Ograniczanie typów prostych

- Przestrzeń wartości typu wyprowadzonego musi być podzbiorem przestrzeni wartości typu bazowego.
- Typy wbudowane mają określone niektóre aspekty, np.: typ `byte`:
  - `minInclusive`: -128,
  - `maxInclusive`: 127.
- Przykład – niepoprawne wyprowadzenie:

```
<xsd:simpleType name="ExtendedByte">
  <xsd:restriction base="xsd:byte">
    <xsd:minInclusive value="-256"/>
    <xsd:maxInclusive value="255"/>
  </xsd:restriction>
</xsd:simpleType>
```


Definiując ograniczenie należy pamiętać, że musi ono faktycznie ograniczać przestrzeń wartości – nie może jej rozszerzać. Jest to szczególnie istotne w przypadku ograniczania typów wbudowanych, które mają określone wartości niektórych aspektów.

Aspekt `whiteSpace`

- `preserve` – wszystkie białe znaki są pozostawiane bez zmian
- `replace` – każdy biały znak jest podczas przetwarzania zastępowany przez spację
- `collapse` - każdy biały znak jest podczas przetwarzania zastępowany przez spację, a następnie każdy ciąg spacji jest zastępowany przez jedną spację, zaś spacje na początku i na końcu są usuwane

Aspekt `whiteSpace` pozwala na kontrolowanie sposobu traktowania białych znaków. W praktyce tego aspektu używa się rzadko – zamiast niego wygodniej jest posługiwać się wbudowanymi typami `string`, `normalizedString` i `token` oraz ich ograniczeniami. Różnią się one od siebie właśnie wartością aspektu `whiteSpace`.

Tworzenie list

```
<xsd:simpleType name="NumerLottoTyp">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="1"/>
    <xsd:maxInclusive value="49"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="NumeryLottoTyp">
  <xsd:list itemType="NumerLottoTyp"/>
</xsd:simpleType>

<xsd:simpleType name="KuponLottoTyp">
  <xsd:restriction base="NumeryLottoTyp">
    <xsd:length value="6"/>
  </xsd:restriction>
</xsd:simpleType>
```


Wartością typu listy jest lista wartości pewnego typu atomowego (nazywanego *typem pozycji listy*), oddzielonych od siebie białymi znakami. Pamiętajmy, że lista jest typem prostym – pozycje na liście nie są zapisane w podelementach, a jedynie oddzielone białymi znakami, a więc cała wartość jest napisem.

Definiując typ listy podajemy (w atrybucie `itemType` lub wewnątrz elementu `list` – patrz następny slajd) typ pozycji listy.

Typ listy można dalej ograniczać, wyproiwadzając z niego kolejne typy. Pamiętajmy, że atrybuty `length`, `minLength` i `maxLength` w przypadku list odnoszą się do liczby pozycji na liście, a nie do liczby znaków, przy pomocy których lista jest zapisana.

Tworzenie list – inaczej

```
<xsd:simpleType name="KuponLottoTyp">
  <xsd:restriction>
    <xsd:simpleType>
      <xsd:list>
        <xsd:simpleType>
          <xsd:restriction base="xsd:integer">
            <xsd:minInclusive value="1"/>
            <xsd:maxInclusive value="49"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:list>
    </xsd:simpleType>
    <xsd:length value="6"/>
  </xsd:restriction>
</xsd:simpleType>
```


Poprawną wartością typu zdefiniowanego na tym i poprzednim slajdzie może być:

`5 15  9 12 9 43`

Niepoprawnymi wartościami tego typu są np.:

`5 15 9 12`

`5 15 59 12 9 43`

`5 15 abc 12 9 43`

Tworzenie unii

```
<xsd:simpleType name="RozmiarLiczbowyTyp">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="2"/>
    <xsd:maxInclusive value="18"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="RozmiarSMLTyp">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="S"/>
    <xsd:enumeration value="M"/>
    <xsd:enumeration value="L"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="RozmiarTyp">
  <xsd:union memberTypes="RozmiarLiczbowyTyp
                          RozmiarSMLTyp"/>
</xsd:simpleType>
```


Definiując unię, określamy (w atrybucie `memberTypes` lub wewnątrz elementu `union` – patrz kolejny slajd) jej typy składowe. Unia jest sumą teoriomnogościową przestrzeni wartości swoich typów składowych. Inaczej mówiąc, każda poprawna wartość typu składowego, jest poprawną wartością typu unii.

Tworzenie unii – inaczej

```
<xsd:simpleType name="RozmiarTyp">
  <xsd:union>
    <xsd:simpleType>
      <xsd:restriction base="xsd:integer">
        <xsd:minInclusive value="2"/>
        <xsd:maxInclusive value="18"/>
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType>
      <xsd:restriction base="xsd:token">
        <xsd:enumeration value="S"/>
        <xsd:enumeration value="M"/>
        <xsd:enumeration value="L"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>
```


Poprawnymi wartościami typu zdefiniowanego na tym i poprzednim slajdzie są np.:

15

M

18

Niepoprawnymi wartościami są np.:

15M

L 18

19

Z

Rozszerzanie typów

- Wartości typu bazowego nie muszą być poprawnymi wartościami typu wyprowadzonego.
- Rozszerzanie zawartości prostej – dodawanie atrybutów do:
  - typu prostego,
  - typu złożonego o zawartości prostej.
- Rozszerzanie zawartości złożonej:
  - dodawanie do typu bazowego dodatkowych elementów i/lub atrybutów,
  - elementy dodawane w rozszerzeniu występują zawsze na końcu, po elementach zadeklarowanych w typie bazowym.

Rozszerzać można zarówno typy proste, jak i złożone, ale w wyniku rozszerzania zawsze powstaje typ złożony. Rozszerzanie polega na dodawaniu do typu bazowego nowych elementów i/lub atrybutów. Wartości typu bazowego niekoniecznie muszą być poprawnymi wartościami typu rozszerzonego, ponieważ rozszerzenie może dodawać elementy lub atrybuty, które są wymagane.

Rozszerzanie zawartości prostej

```
<xsd:complexType name="MasaTyp">
  <xsd:simpleContent>
    <xsd:extension base="xsd:decimal">
      <xsd:attribute name="jm" type="xsd:string"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:complexType name="MasaNiedokładnaTyp">
  <xsd:simpleContent>
    <xsd:extension base="MasaTyp">
      <xsd:attribute name="dokładność"
                     type="xsd:decimal"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>
```


Rozszerzając typ prosty lub typ złożony o zawartości prostej, możemy do niego dodawać wyłącznie atrybuty. Wynikiem takiego rozszerzenia jest typ złożony o zawartości prostej.

Rozszerzanie zawartości złożonej

```
<xsd:complexType name="OsobaTyp">
  <xsd:sequence>
    <xsd:element name="imie" type="xsd:string"/>
    <xsd:element name="nazwisko" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="OsobaZDokumentemTyp">
  <xsd:complexContent>
    <xsd:extension base="OsobaTyp">
      <xsd:choice>
        <xsd:element name="dowód-os"
                     type="DowódTyp"/>
        <xsd:element name="paszport"
                     type="PaszportTyp"/>
      </xsd:choice>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>
```


Rozszerzając zawartość złożoną można dodawać podelementy na końcu modelu zawartości typu bazowego. Można także dodawać atrybuty. Nie można jednak modyfikować ani usuwać istniejących atrybutów.

Definiując rozszerzenie zawartości złożonej, nie trzeba powtarzać modelu zawartości typu bazowego. Procesor obsługuje rozszerzenia zawartości złożonej dodając nowy model zawartości po modelu zawartości typu bazowego, jak gdyby oba modele były zawarte w grupie `sequence`.

(Priscilla Walmsley, *Wszystko o XML Schema*, WNT, 2008)

Typy

- Typy wg zasięgu definicji:
  - typy nazwane,
  - typy anonimowe.
- Typy wg zawartości:
  - typy proste,
  - typy złożone o zawartości:
    - prostej,
    - elementowej,
    - mieszanej,
    - pustej.
- Typy wg pochodzenia:
  - typy wbudowane,
  - typy zdefiniowane w schemacie:
    - rozszerzenia innych typów,
    - ograniczenia innych typów,
    - listy i unie.

Inne metody modularyzacji schematów

- Ograniczanie typów złożonych.
- Grupy do wielokrotnego wykorzystania (*reusable groups*):
  - nazwane grupy modeli,
  - grupy atrybutów.
- Grupy zamienne (*substitution groups*).

Ograniczanie typów złożonych

- Przestrzeń wartości typu wyprowadzonego musi być podzbiorem przestrzeni wartości typu bazowego.
- Dozwolone operacje:
  - ograniczenie zawartości prostej (tak jak typu prostego),
  - ograniczenie atrybutu:
    - ograniczenie typu atrybutu,
    - zmiana atrybutu opcjonalnego na wymagany (`required`) lub zabroniony (`prohibited`),
    - dodanie, zmiana lub usunięcie wartości domyślnej,
    - dodanie wartości ustalonej, jeśli jej nie było.
  - ograniczenie modelu zawartości, np.:
    - ściślejsze ograniczenia liczebności (`minOccurs`, `maxOccurs`),
    - usunięcie elementów opcjonalnych w grupach `sequence` i `all`,
    - wybranie podzbioru elementów w grupie `choice`,
    - ograniczenie typu poszczególnych podelementów.

Grupy do wielokrotnego wykorzystania

- Grupy używanych wspólnie deklaracji elementów lub atrybutów.

```
<xsd:group name="ImięNazwisko">
  <xsd:sequence>
    <xsd:element name="imie" type="xsd:string"/>
    <xsd:element name="nazwisko" type="xsd:string"/>
  </xsd:sequence>
</xsd:group>

<xsd:attributeGroup name="Biurokracja">
  <xsd:attribute name="NIP" type="NIPTyp"/>
  <xsd:attribute name="PESEL" type="PESELTyp"/>
</xsd:attributeGroup>
```


Są dwa rodzaje grup do wielokrotnego wykorzystania:

•nazwane grupy modeli (`group`),

•grupy atrybutów (`attributeGroup`).

Jest to prosta metoda modularyzacji schematu. Jeśli dany fragment modelu zawartości lub zestaw atrybutów jest często używany, można zdefiniować go jako grupę, a następnie odwoływać się do tej grupy w wielu definicjach typów złożonych.

Grupy do wielokrotnego wykorzystania

- Odwołania do grup:

```
<xsd:complexType name="OsobaTyp">
  <xsd:sequence>
    <xsd:group ref="ImięNazwisko"/>
    <xsd:element name="obywatelstwo"
                 type="xsd:token"/>
  </xsd:sequence>
  <xsd:attributeGroup ref="Biurokracja"/>
</xsd:complexType>
```


Odwołanie do grupy jest interpretowane w taki sposób, jakby zawartość grupy była wklejona w miejscu, gdzie występuje odwołanie.

Grupy zamienne

- Możliwość deklaracji zamienników dla wybranych elementów.

```
<xsd:element name="katalog">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="produkt"
                   maxOccurs="unbounded"/>
    <xsd:sequence>
  <xsd:complexType>
</xsd:element>
<xsd:element name="produkt" type="ProduktTyp"/>
<xsd:element name="komputer" type="KomputerTyp"
             substitutionGroup="produkt"/>
<xsd:element name="ubranie" type="UbranieTyp"
             substitutionGroup="produkt"/>
```


Przy pomocy grup zamiennych można wskazać, że pewne elementy mogą być używane w egzemplarzach jako zamienniki pewnego elementu, zwanego *głową* grupy zamiennej. Nie trzeba przy tym w żaden sposób modyfikować deklaracji głowy ani dodawać czegokolwiek w definicji typu, w której głowa występuje. Wystarczy w deklaracjach zamienników (*członków grupy zamiennej*) wskazać, jaki element mogą one zastępować. Dlatego grupy zamienne uważa się za bardzo elastyczny i wygodny sposób rozszerzania schematów.

Zauważmy, że przy pomocy grup zamiennych można rozszerzyć schemat, nie modyfikując jego bazowych deklaracji i definicji. Jest to istotne np. w sytuacjach, gdy chcemy rozszerzyć schemat branżowy, nie mając praw do jego modyfikacji.

Element A można zadeklarować jako zamiennik elementu B, pod warunkiem, że typ elementu A jest typem wyprowadzonym z typu elementu B, tzn. rozszerzeniem lub ograniczeniem typu B.

Grupy zamienne

- Wykorzystanie grup zamiennych (egzemplarz):

```
<katalog>
  <produkt>...</produkt>
  <komputer>...</komputer>
  <ubranie>...</ubranie>
  <produkt>...</produkt>
</katalog>
```


W egzemplarzu nie ma potrzeby oznaczania zamienników w żaden szczególny sposób. Zamienników używamy tak, jakby były zadeklarowane bezpośrednio w modelu zawartości elementu `katalog`.

Ograniczenia integralności

- Ograniczenia integralności (*identity constraints*):
  - `unique` – wartości w ramach określonego zakresu muszą być unikatowe,
  - `key` – wartości w ramach określonego zakresu muszą być unikatowe i nie mogą być puste,
  - `keyref` – odwołanie do wartości wskazanego klucza.
- Odpowiednik bazodanowych kluczy pierwotnych i kluczy zewnętrznych.

Ograniczenia unikatowości (`unique`) pozwalają wymusić, aby wartość (lub kombinacja wartości) była unikatowa w określonym zasięgu. Przy pomocy takiego ograniczenia można na przykład określić, że wszystkie numery produktów w katalogu muszą być unikatowe.

Klucze (`key`) także zapewniają unikatowość wartości, dodatkowo wymuszając, aby wszystkie wartości występowały w dokumencie. Przy pomocy klucza można na przykład określić, że każdy produkt musi mieć numer, który jest unikatowy w katalogu.

Odwołania do kluczy (`keyref`) wymuszają, aby dla każdej wartości (lub kombinacji wartości) istniała odpowiednia wartość klucza lub ograniczenia unikatowości. Przy pomocy odwołania można na przykład określić, że dla każdego numeru produktu, jaki pojawia się w pozycji zamówienia, musi istnieć odpowiedni numer produktu w sekcji zawierającej opisy produktów.

(Priscilla Walmsley, *Wszystko o XML Schema*, WNT, 2008)

Przykład

```
<xsd:element name="zamówienia-i-faktury"
             type="ZamówieniaFakturyTyp">
  <xsd:unique name="fakturaId">
    <xsd:selector xpath="faktury/faktura"/>
    <xsd:field xpath="@nr"/>
    <xsd:field xpath="rok"/>
  </xsd:unique>
  <xsd:keyref name="zamówienieRef"
              refer="zamówienieId">
    <xsd:selector
      xpath="faktury/faktura/do-zamówienia"/>
    <xsd:field xpath="@nr"/>
  </xsd:keyref>
  <xsd:key name="zamówienieId">
    <xsd:selector xpath="zamówienia/zamówienie"/>
    <xsd:field xpath="numer"/>
  </xsd:key>
</xsd:element>
```


Definicja ograniczenia integralności składa się z trzech części:

•*Zasięg* (ang. *scope*) to element, w którego deklaracji jest zawarte ograniczenie. Dwa zamówienia o tym samym numerze będą najzupełniej poprawne, jeśli będą umieszczone w dwóch różnych elementach `zamówienia-i-faktury`.

•*Selektor* (ang. *selector*) pozwala wskazać węzły, których dotyczy ograniczenie.

•Jedno lub więcej *pól* (ang. *fields*) to wartości elementów i atrybutów, które muszą być unikatowe wśród wskazanych węzłów. W każdym węźle wskazanym przez selektor może być co najwyżej jeden egzemplarz każdego pola.

Selektory i pola zapisuje się przy pomocy wyrażeń XPath (nie można używać zupełnie dowolnych wyrażeń XPath, rekomendacja XML Schema definiuje podzbiór XPath, który jest dozwolony w ograniczeniach integralności).

Przykład – egzemplarz (1)

```
<xsd:unique name="fakturaId">
  <xsd:selector
   xpath="faktury/faktura"/>
  <xsd:field xpath="@nr"/>
  <xsd:field xpath="rok"/>
</xsd:unique>
```

```
<zamówienia-i-faktury>
  <faktury>
    <faktura nr="123">
      <rok>2007</rok>
      <do-zamówienia nr="125"/>
      ...
    </faktura>
    <faktura nr="123">
      <rok>2006</rok>
      <do-zamówienia nr="665"/>
      ...
    </faktura>
    <faktura nr="127/2007">
      <do-zamówienia nr="125"/>
      ...
    </faktura>
  </faktury>
```

```
<xsd:keyref name="zamówienieRef"
            refer="zamówienieId">
  <xsd:selector xpath=
    "faktury/faktura/do-zamówienia"/>
  <xsd:field xpath="@nr"/>
</xsd:keyref>
```


Zwróćmy uwagę, że nie jest wymagane, aby wartości pól odwołania do klucza były unikatowe.

Przykład – egzemplarz (2)

```
<zamówienia>
  <zamówienie>
    <numer>125</numer>
    ...
  </zamówienie>
  <zamówienie>
    <numer>665</numer>
    ...
  </zamówienie>
  <zamówienie>
    <numer>233</numer>
    ...
  </zamówienie>
</zamówienia>
</zamówienia-i-faktury>
```

```
<xsd:key name="zamówienieId">
  <xsd:selector
    xpath="zamówienia/zamówienie"/>
  <xsd:field xpath="numer"/>
</xsd:key>
```