XML
i nowoczesne metody zarządzania treścią

Wykład 3: Modelowanie dokumentów XML-owych: XML Schema

Maciej Ogrodniczuk

MIMUW, 15 października 2009

Przykład DTD

```
<!ELEMENT słownik (hasło)+>
```

Przykład DTD

```
<!ELEMENT słownik (hasło)+>

<!ELEMENT hasło (pojęcie, objaśnienie)>
```

Przykład DTD

```
<!ELEMENT słownik (hasło)+>

<!ELEMENT hasło (pojęcie, objaśnienie)>
<!ATTLIST hasło
          id ID #REQUIRED
          dataZapisu NMTOKEN #IMPLIED>
```

Przykład DTD

```
<!ELEMENT słownik (hasło)+>

<!ELEMENT hasło (pojęcie, objaśnienie)>
<!ATTLIST hasło
          id ID #REQUIRED
          dataZapisu NMTOKEN #IMPLIED>

<!ELEMENT pojęcie (#PCDATA)>
```

Przykład DTD

```
<!ELEMENT słownik (hasło)+>

<!ELEMENT hasło (pojęcie, objaśnienie)>
<!ATTLIST hasło
          id ID #REQUIRED
          dataZapisu NMTOKEN #IMPLIED>

<!ELEMENT pojęcie (#PCDATA)>

<!ELEMENT objaśnienie (#PCDATA | link)*>
```

Przykład DTD

```
<!ELEMENT słownik (hasło)+>

<!ELEMENT hasło (pojęcie, objaśnienie)>
<!ATTLIST hasło
          id ID #REQUIRED
          dataZapisu NMTOKEN #IMPLIED>

<!ELEMENT pojęcie (#PCDATA)>

<!ELEMENT objaśnienie (#PCDATA | link)*>

<!ELEMENT link (#PCDATA)>
```

Przykład DTD

```
<!ELEMENT słownik (hasło)+>

<!ELEMENT hasło (pojęcie, objaśnienie)>
<!ATTLIST hasło
          id ID #REQUIRED
          dataZapisu NMTOKEN #IMPLIED>

<!ELEMENT pojęcie (#PCDATA)>

<!ELEMENT objaśnienie (#PCDATA | link)*>

<!ELEMENT link (#PCDATA)>
<!ATTLIST link
          hasło IDREF #REQUIRED>
```

Wady DTD

- Ograniczona kontrola nad strukturą dokumentów.

Wady DTD

- Ograniczona kontrola nad strukturą dokumentów.
- Zbyt „wysokopoziomowe” typy danych: liczby, daty są zawsze reprezentowane jako tekst!

Wady DTD

- Ograniczona kontrola nad strukturą dokumentów.
- Zbyt „wysokopoziomowe” typy danych: liczby, daty są zawsze reprezentowane jako tekst!
- Bardzo ogólne metody definiowania częstości wystąpień. Aby określić, że zwrotka wiersza może mieć od 4 do 8 wersów musimy ją zdefiniować w następujący, skomplikowany sposób:

```
<!ELEMENT zwrotka (wers, wers, wers, wers,
                   wers?, wers?, wers?, wers?)>
```

Wady DTD

- Ograniczona kontrola nad strukturą dokumentów.
- Zbyt „wysokopoziomowe” typy danych: liczby, daty są zawsze reprezentowane jako tekst!
- Bardzo ogólne metody definiowania częstości wystąpień. Aby określić, że zwrotka wiersza może mieć od 4 do 8 wersów musimy ją zdefiniować w następujący, skomplikowany sposób:

```
<!ELEMENT zwrotka (wers, wers, wers, wers,
                   wers?, wers?, wers?, wers?)>
```

- Mało „obiektowe”, nierozszerzalne definicje. Chcielibyśmy np. zdefiniować zwrotkę w jednym DTD, a następnie rozszerzyć ją w innym bez przepisywania całej definicji.

Wady DTD

- Ograniczona kontrola nad strukturą dokumentów.
- Zbyt „wysokopoziomowe” typy danych: liczby, daty są zawsze reprezentowane jako tekst!
- Bardzo ogólne metody definiowania częstości wystąpień. Aby określić, że zwrotka wiersza może mieć od 4 do 8 wersów musimy ją zdefiniować w następujący, skomplikowany sposób:

```
<!ELEMENT zwrotka (wers, wers, wers, wers,
                   wers?, wers?, wers?, wers?)>
```

- Mało „obiektowe”, nierozszerzalne definicje. Chcielibyśmy np. zdefiniować zwrotkę w jednym DTD, a następnie rozszerzyć ją w innym bez przepisywania całej definicji.
- Składnia różna od składni opisywanej zawartości (zaszłości SGML-owe): może udałoby się użyć XML-a do opisu składni dokumentów?

Świat bez XML Schema

Mamy system przetwarzający zamówienia przysyłane jako dokumenty XML-owe zawierające pole odpowiadające liczbie zamawianych produktów.

Świat bez XML Schema

Mamy system przetwarzający zamówienia przysyłane jako dokumenty XML-owe zawierające pole odpowiadające liczbie zamawianych produktów.

Co się stanie, gdy system odbierze zamówienie:

```
<zamówienie>
  <liczba-prod>no kocham go do szaleństwa</liczba-prod>
</zamówienie>
```

Świat bez XML Schema

Mamy system przetwarzający zamówienia przysyłane jako dokumenty XML-owe zawierające pole odpowiadające liczbie zamawianych produktów.

Co się stanie, gdy system odbierze zamówienie:

```
<zamówienie>
  <liczba-prod>no kocham go do szaleństwa</liczba-prod>
</zamówienie>
```

A takie?

```
<zamówienie>
  <liczba-prod>-10000</liczba-prod>
</zamówienie>
```

Świat bez XML Schema

Mamy system przetwarzający zamówienia przysyłane jako dokumenty XML-owe zawierające pole odpowiadające liczbie zamawianych produktów.

Co się stanie, gdy system odbierze zamówienie:

```
<zamówienie>
  <liczba-prod>no kocham go do szaleństwa</liczba-prod>
</zamówienie>
```

A takie?

```
<zamówienie>
  <liczba-prod>-10000</liczba-prod>
</zamówienie>
```

Przydałoby się niskopoziomowe sprawdzanie poprawności — najlepiej wbudowane w sam dokument.

Specyfikacja XML Schema

Najważniejsze dokumenty:

- 1999: dokument W3C opisujący wymagania stawiane przed nowym formatem: mechanizmy tworzenia struktury, typy proste i reguły przetwarzania,

Specyfikacja XML Schema

Najważniejsze dokumenty:

- 1999: dokument W3C opisujący wymagania stawiane przed nowym formatem: mechanizmy tworzenia struktury, typy proste i reguły przetwarzania,
- 2001: XML Schema staje się oficjalną rekomendacją W3C:
  - *XML Schema Part 0: Primer*,
  - *XML Schema Part 1: Structures*,
  - *XML Schema Part 2: Datatypes*.

Specyfikacja XML Schema

Najważniejsze dokumenty:

- 1999: dokument W3C opisujący wymagania stawiane przed nowym formatem: mechanizmy tworzenia struktury, typy proste i reguły przetwarzania,
- 2001: XML Schema staje się oficjalną rekomendacją W3C:
  - *XML Schema Part 0: Primer*,
  - *XML Schema Part 1: Structures*,
  - *XML Schema Part 2: Datatypes*.
- 2004: aktualne, drugie wydanie Specyfikacji.

Wprowadzenie

Najważniejsze informacje:

Wprowadzenie

Najważniejsze informacje:

- XML Schema rozszerza funkcjonalność DTD (możliwa jest automatyczna konwersja DTD do formatu XML Schema).

Wprowadzenie

Najważniejsze informacje:

- XML Schema rozszerza funkcjonalność DTD (możliwa jest automatyczna konwersja DTD do formatu XML Schema).
- Do definicji typu dokumentu w formacie XML Schema wykorzystywana jest standardowa składnia XML-a.

Wprowadzenie

Najważniejsze informacje:

- XML Schema rozszerza funkcjonalność DTD (możliwa jest automatyczna konwersja DTD do formatu XML Schema).
- Do definicji typu dokumentu w formacie XML Schema wykorzystywana jest standardowa składnia XML-a.
- Składniki definicji należą do przestrzeni nazw XML Schema `http://www.w3.org/2001/XMLSchema` (dalej będę się do niej odwoływał prefiksem `xsd`).

Wprowadzenie

Najważniejsze informacje:

- XML Schema rozszerza funkcjonalność DTD (możliwa jest automatyczna konwersja DTD do formatu XML Schema).
- Do definicji typu dokumentu w formacie XML Schema wykorzystywana jest standardowa składnia XML-a.
- Składniki definicji należą do przestrzeni nazw XML Schema `http://www.w3.org/2001/XMLSchema` (dalej będę się do niej odwoływał prefiksem `xsd`).
- Cała definicja zawarta jest w elemencie głównym `<xsd:schema>`, zaś odpowiednikami deklaracji elementów i atrybutów z DTD są elementy `<xsd:element>` i `<xsd:attribute>`.

Typy proste i złożone

XML Schema wprowadza do XML-a typy danych. Pod względem zawartości typy mogą być proste (ang. *simple types*) i złożone (ang. *complex types*).

Typy proste i złożone

XML Schema wprowadza do XML-a typy danych. Pod względem zawartości typy mogą być proste (ang. *simple types*) i złożone (ang. *complex types*).

Typy proste odpowiadają zawartości bez wewnętrznej struktury (bez podelementów ani atrybutów). Mogą być przypisywane elementom i atrybutom.

Typy proste i złożone

XML Schema wprowadza do XML-a typy danych. Pod względem zawartości typy mogą być proste (ang. *simple types*) i złożone (ang. *complex types*).

Typy proste odpowiadają zawartości bez wewnętrznej struktury (bez podelementów ani atrybutów). Mogą być przypisywane elementom i atrybutom.

Typy złożone mogą mieć zawartość elementową i wprowadzać atrybuty.

Typy proste

Typami prostymi są wszystkie typy wbudowane (np. liczba, napis, wartość logiczna), jak również typy stworzone na ich bazie.

Oto przykład deklaracji elementu o zawartości typu prostego:

```
<xsd:element name="pojęcie" type="xsd:string"/>
```

Typy proste

Typami prostymi są wszystkie typy wbudowane (np. liczba, napis, wartość logiczna), jak również typy stworzone na ich bazie.

Oto przykład deklaracji elementu o zawartości typu prostego:

```
<xsd:element name="pojęcie" type="xsd:string"/>
```

W DTD odpowiada to definicji:

Typy proste

Typami prostymi są wszystkie typy wbudowane (np. liczba, napis, wartość logiczna), jak również typy stworzone na ich bazie.

Oto przykład deklaracji elementu o zawartości typu prostego:

```
<xsd:element name="pojęcie" type="xsd:string"/>
```

W DTD odpowiada to definicji:

```
<!ELEMENT pojęcie (#PCDATA)>
```

Wbudowane typy proste — przegląd

Typy wbudowane:

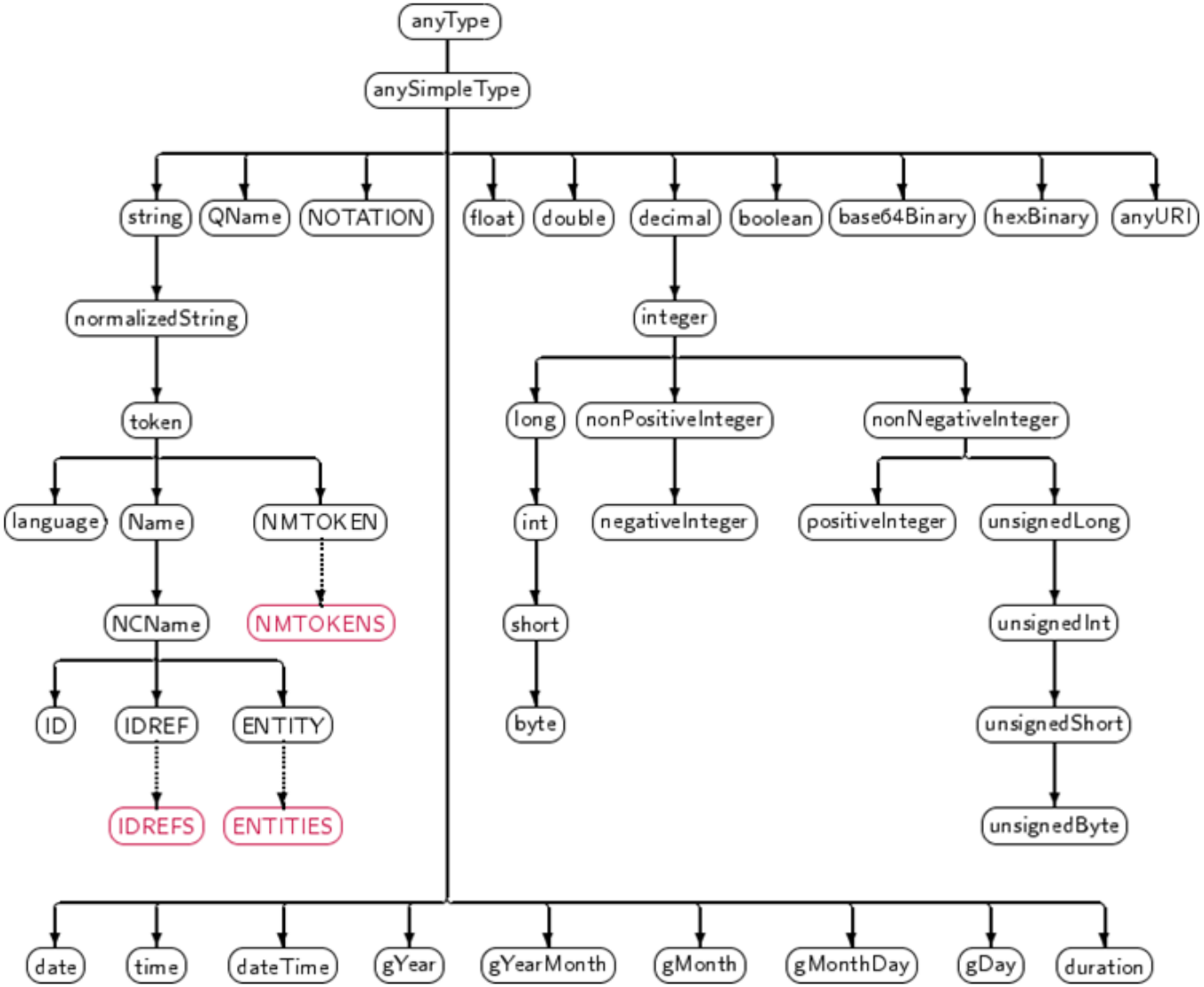
- `string` — ciąg znaków,
- `boolean` — wartość logiczna (`true`, `false`, `0` lub `1`),
- `decimal` — liczby rzeczywiste z plusem lub minusem; kropka oddziela część dziesiętną,
- `float` — 32-bitowa liczba rzeczywista (dopuszcza także wartości: `-INF`, `INF` i `NaN`),
- `double` — 64-bitowa liczba rzeczywista (także `-INF`, `INF` i `NaN`),
- `normalizedString` — napis, w którym każdy biały znak jest podczas przetwarzania zastępowany przez spację,

Wbudowane typy proste — przegląd

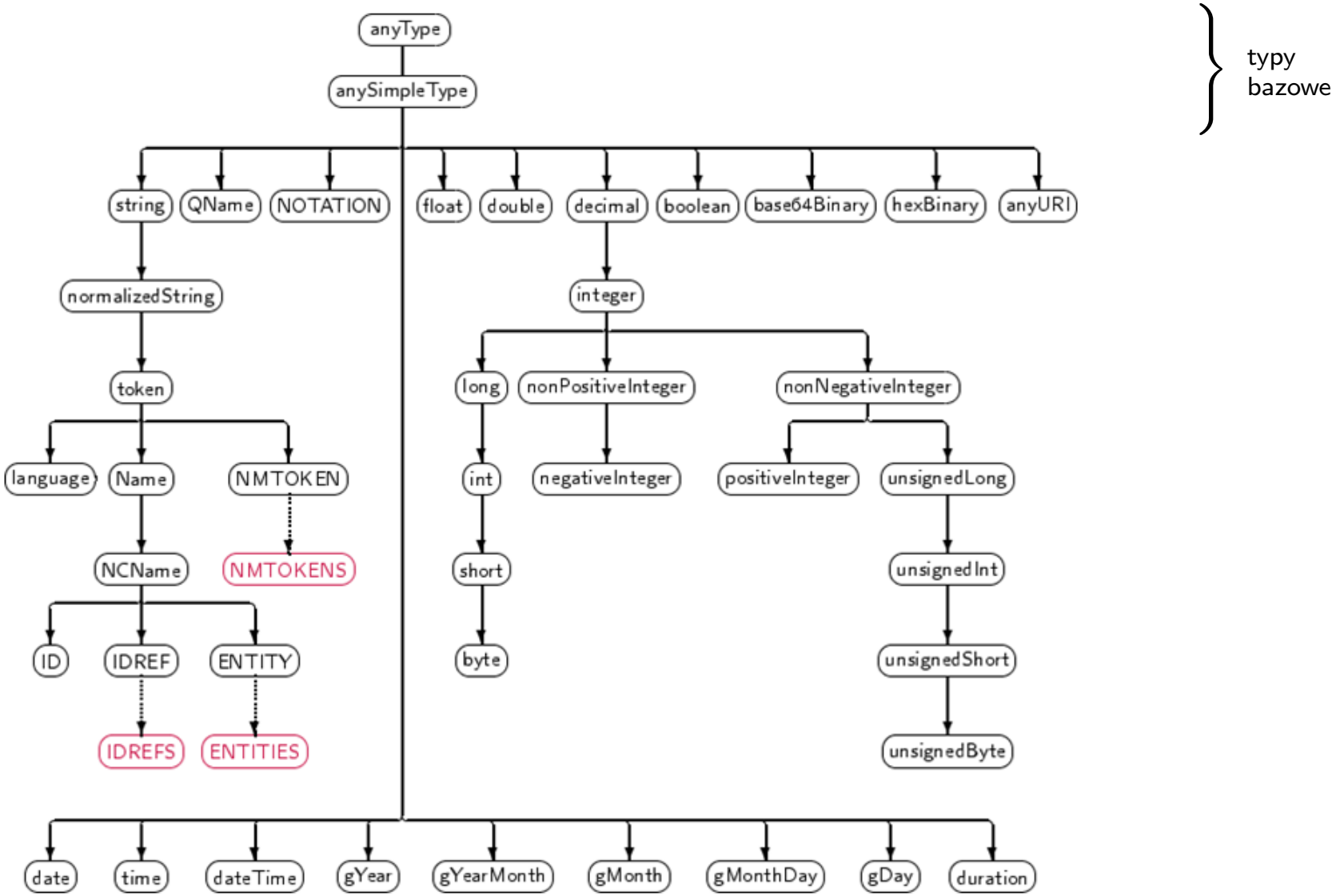
Typy wbudowane:

- `token` — napis, w którym każdy ciąg białych znaków jest podczas przetwarzania zastępowany przez jedną spację, zaś białe znaki na początku i końcu są usuwane,
- `hexBinary` — dane binarne zapisane szesnastkowo,
- `byte` — liczba całkowita z przedziału od `-128` do `127`,
- `integer` — liczba całkowita z przedziału od `-126789` do `126789`,
- `date`, `time`, `dateTime`, `duration`, `gYearMonth`, ...
- `ID`, `IDREF`, `IDREFS`, `ENTITY`, `ENTITIES`, `NOTATION`, `NMTOKEN`, `NMTOKENS`, `CDATA`, `language`, `Name`, `normalizedString`, `token`, `uriReference`...

Drzewo wbudowanych typów prostych

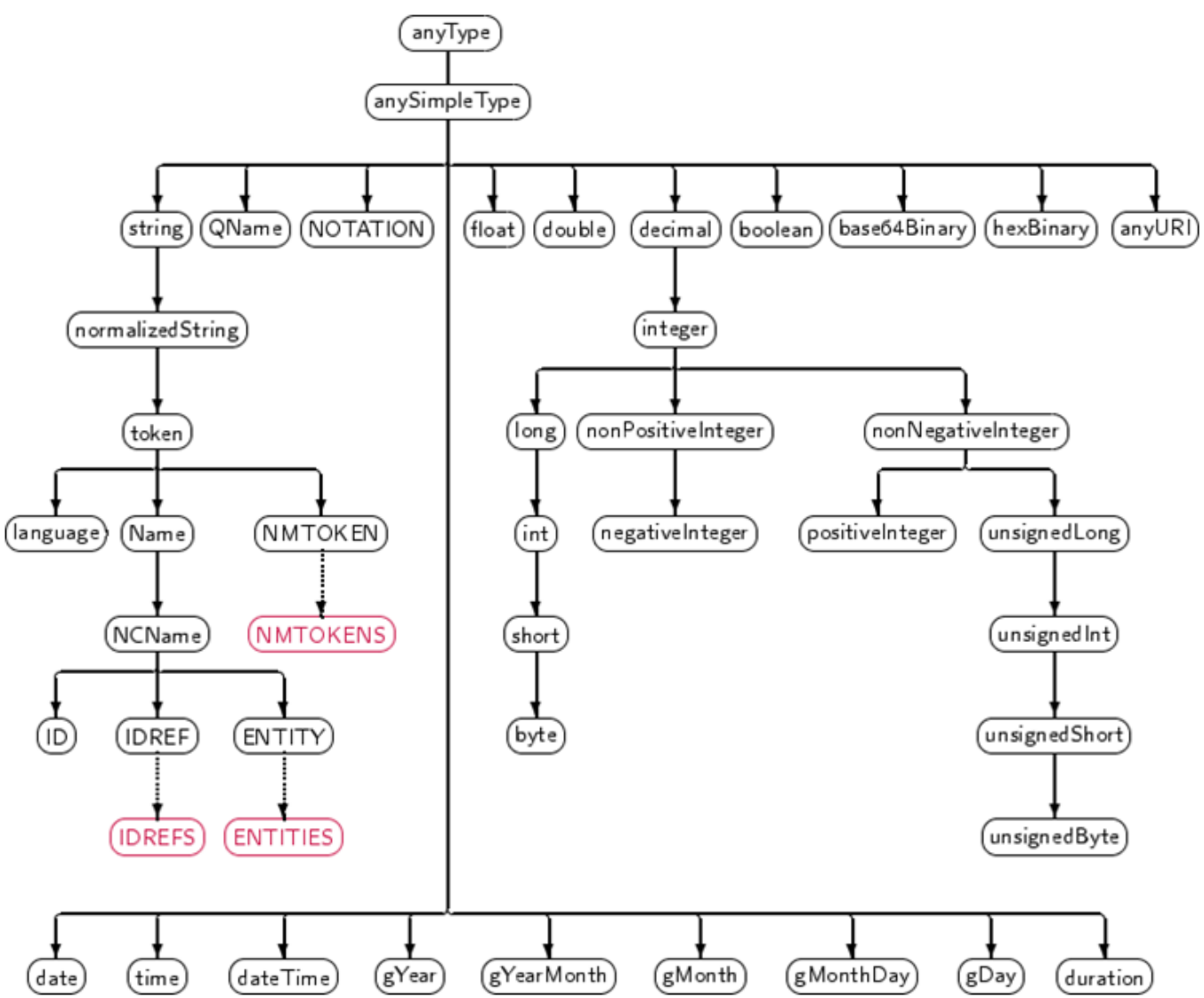


Drzewo wbudowanych typów prostych



typy bazowe

Drzewo wbudowanych typów prostych

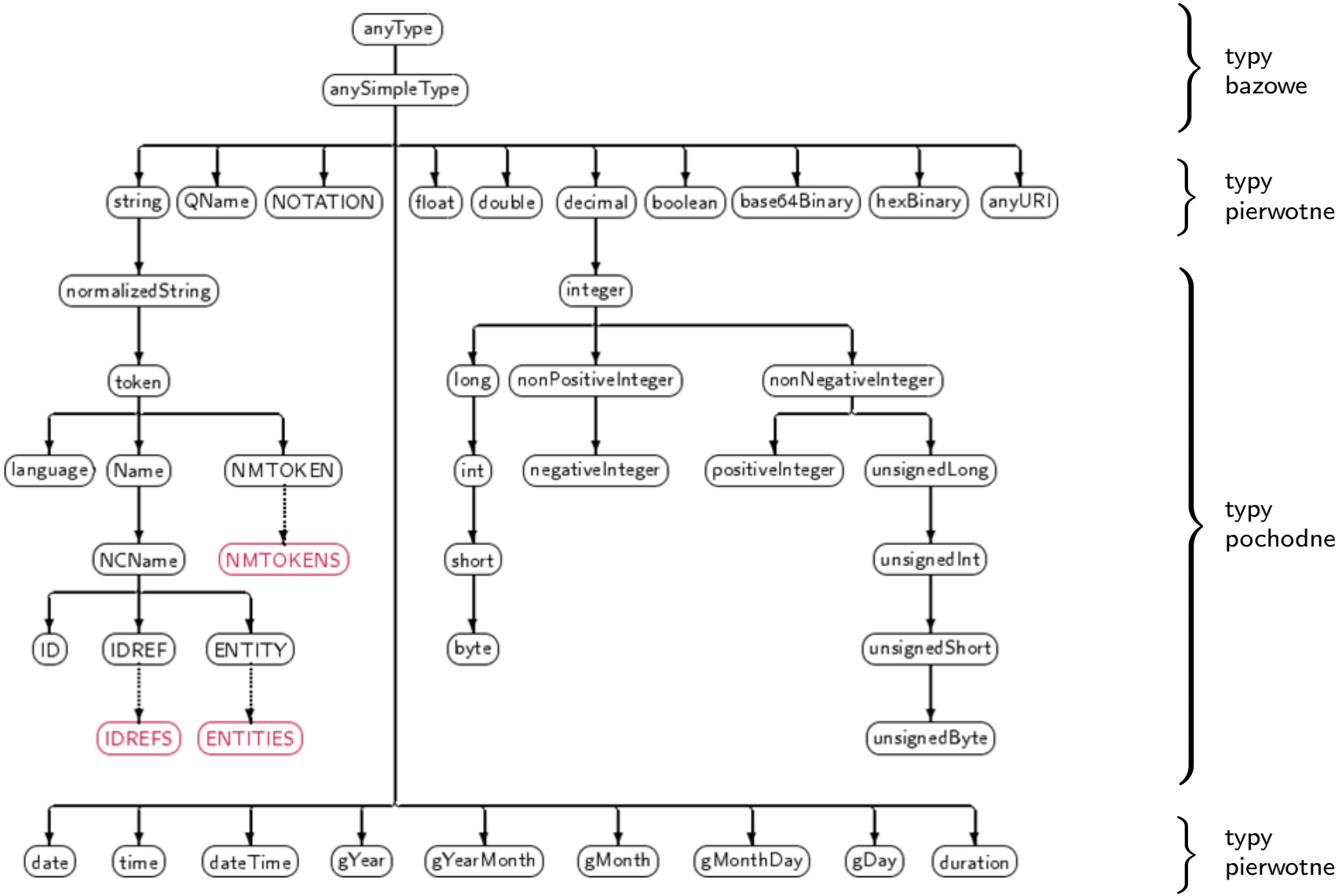


typy bazowe

typy pierwotne

typy pierwotne

Drzewo wbudowanych typów prostych



Typy bazowe xsd:anyType i xsd:anySimpleType

Typ xsd:anyType może być użyty jawnie do definiowania zawartości elementu:

```
<xsd:element name="fragmentKodu" type="xsd:anyType"/>
```

Typy bazowe xsd:anyType i xsd:anySimpleType

Typ xsd:anyType może być użyty jawnie do definiowania zawartości elementu:

```
<xsd:element name="fragmentKodu" type="xsd:anyType"/>
```

Taka definicja zezwala na dowolną zawartość znakową i elementową.

Jeśli w definicji elementu nie zostanie podana nazwa typu, typ xsd:anyType zostanie użyty jako domyślny.

Typy bazowe xsd:anyType i xsd:anySimpleType

Typ xsd:anyType może być użyty jawnie do definiowania zawartości elementu:

```
<xsd:element name="fragmentKodu" type="xsd:anyType"/>
```

Taka definicja zezwala na dowolną zawartość znakową i elementową.

Jeśli w definicji elementu nie zostanie podana nazwa typu, typ xsd:anyType zostanie użyty jako domyślny.

Podobną funkcję pełni typ xsd:anySimpleType, który nie dopuszcza zawartości elementowej.

Własne typy danych

Na podstawie typów predefiniowanych można łatwo tworzyć własne typy proste wykorzystując tzw. **aspekty** (ang. *facets*).

Najważniejsze z nich:

- `minInclusive`, `maxInclusive`, `minExclusive`, `maxExclusive` — zawężają zakres dozwolonych wartości liczbowych (wartości minimalna i maksymalna, odpowiednio z włączeniem lub wyłączeniem podanej wartości),
- `pattern` — wzorzec wartości zgodny z podanym wyrażeniem regularnym (`.`, `a?`, `a+`, `a*`, `(a|b)`, `[a-c]`), `(ab){2,}`, ...)
- `enumeration` — typ wyliczeniowy,
- `list` — listy wartości typu prostego (jak `NMTOKENS`),
- `union` — suma teoriomnogościowa wartości kilku typów,
- `length`, `minLength`, `maxLength` — opowiednio wymagana, minimalna lub maksymalna długość napisu lub listy.

Przykład: nowy typ prosty z wykorzystaniem wzorca

Nowy typ prosty tworzymy poprzez użycie elementu `<xsd:simpleType>`.

Skorzystanie z aspektów wymaga zastosowania w definicji elementu `<xsd:restriction>` ograniczającego typ wskazany atrybutem `base` lub podany w treści tego elementu.

Przykład: nowy typ prosty z wykorzystaniem wzorca

Nowy typ prosty tworzymy poprzez użycie elementu `<xsd:simpleType>`.

Skorzystanie z aspektów wymaga zastosowania w definicji elementu `<xsd:restriction>` ograniczającego typ wskazany atrybutem `base` lub podany w treści tego elementu.

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

Przykład: nowy typ prosty z wykorzystaniem wzorca

Nowy typ prosty tworzymy poprzez użycie elementu `<xsd:simpleType>`.

Skorzystanie z aspektów wymaga zastosowania w definicji elementu `<xsd:restriction>` ograniczającego typ wskazany atrybutem `base` lub podany w treści tego elementu.

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

Przykład: nowy typ prosty z wykorzystaniem wzorca

Nowy typ prosty tworzymy poprzez użycie elementu `<xsd:simpleType>`.

Skorzystanie z aspektów wymaga zastosowania w definicji elementu `<xsd:restriction>` ograniczającego typ wskazany atrybutem base lub podany w treści tego elementu.

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

Przykład: nowy typ prosty z wykorzystaniem wzorca

Nowy typ prosty tworzymy poprzez użycie elementu `<xsd:simpleType>`.

Skorzystanie z aspektów wymaga zastosowania w definicji elementu `<xsd:restriction>` ograniczającego typ wskazany atrybutem `base` lub podany w treści tego elementu.

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

Typy anonimowe i typy nazwane

Typ, którego definicję podajemy w miejscu użycia, to **typ anonimowy**.

Typy anonimowe i typy nazwane

Typ, którego definicję podajemy w miejscu użycia,
to **typ anonimowy**.

Poprzez uniezależnienie definicji typu od miejsca jego wystąpienia możemy tworzyć **typy nazwane**:

```
<xsd:simpleType name="typKodPocztowy">
   ...
</xsd:simpleType>
```

Typy anonimowe i typy nazwane

Typ, którego definicję podajemy w miejscu użycia,
to **typ anonimowy**.

Poprzez uniezależnienie definicji typu od miejsca jego wystąpienia możemy tworzyć **typy nazwane**:

```
<xsd:simpleType name="typKodPocztowy">
   ...
</xsd:simpleType>
```

i używać ich (wielokrotnie) w definicjach schematu:

```
<xsd:element name="kodPocztowy" type="typKodPocztowy"/>
```

Przykład: przedział wartości

DTD:

Nie da się ograniczyć tekstowej zawartości elementów!

Przykład: przedział wartości

DTD:

Nie da się ograniczyć tekstowej zawartości elementów!

XML Schema:

```
<xsd:simpleType name="typWiek">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="0"/>
    <xsd:maxInclusive value="120"/>
  </xsd:restriction>
</xsd:simpleType>
```

Przykład: przedział wartości

DTD:

Nie da się ograniczyć tekstowej zawartości elementów!

XML Schema:

```
<xsd:simpleType name="typWiek">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="0"/>
    <xsd:maxInclusive value="120"/>
  </xsd:restriction>
</xsd:simpleType>
```

Przykład: typ wyliczeniowy

DTD:

Nie da się ograniczyć tekstowej zawartości elementów!

Przykład: typ wyliczeniowy

DTD:

Nie da się ograniczyć tekstowej zawartości elementów!

XML Schema:

```
<xsd:simpleType name="typPłeć">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="kobieta"/>
    <xsd:enumeration value="mężczyzna"/>
  </xsd:restriction>
</xsd:simpleType>
```

Przykład: typ wyliczeniowy

DTD:

Nie da się ograniczyć tekstowej zawartości elementów!

XML Schema:

```
<xsd:simpleType name="typPłeć">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="kobieta"/>
    <xsd:enumeration value="mężczyzna"/>
  </xsd:restriction>
</xsd:simpleType>
```

Uwaga na ograniczenia!

Przestrzeń wartości wyprowadzonego typu prostego musi być podzbiorem przestrzeni wartości typu bazowego.

Warto o tym pamiętać ograniczając typ wbudowany, dla którego określono niektóre aspekty. Typ `byte` może np. przyjmować wartości z przedziału -128 – 127.

Uwaga na ograniczenia!

Przestrzeń wartości wyprowadzonego typu prostego musi być podzbiorem przestrzeni wartości typu bazowego.

Warto o tym pamiętać ograniczając typ wbudowany, dla którego określono niektóre aspekty. Typ `byte` może np. przyjmować wartości z przedziału -128 – 127.

Błędne ograniczenie:

```
<xsd:simpleType name="typByte9">
  <xsd:restriction base="xsd:byte">
    <xsd:minInclusive value="-256"/>
    <xsd:maxInclusive value="255"/>
  </xsd:restriction>
</xsd:simpleType>
```

Listy wartości

Lista to ciąg oddzielonych białymi znakami wartości wskazanego typu.

Listy wartości

Lista to ciąg oddzielonych białymi znakami wartości wskazanego typu.

Przykład definicji typu (z atrybutem base):

```
<xsd:simpleType name="typListaLiczb">
  <xsd:list itemType="xsd:integer"/>
</xsd:simpleType>

<xsd:simpleType name="typLista10Liczb">
  <xsd:restriction base="typListaLiczb">
    <xsd:length value="10"/>
  </xsd:restriction>
</xsd:simpleType>
```

Listy wartości

Wariant definicji (z typem ograniczanym podanym jawnie w treści elementu <xsd:restriction>):

```
<xsd:simpleType name="typLista10Liczb">
  <xsd:restriction>
    <xsd:simpleType>
      <xsd:list itemType="xsd:integer"/>
    </xsd:simpleType>
    <xsd:length value="10"/>
  </xsd:restriction>
</xsd:simpleType>
```

Listy wartości

Wariant definicji (z typem ograniczanym podanym jawnie w treści elementu <xsd:restriction>):

```
<xsd:simpleType name="typLista10Liczb">
  <xsd:restriction>
    <xsd:simpleType>
      <xsd:list itemType="xsd:integer"/>
    </xsd:simpleType>
    <xsd:length value="10"/>
  </xsd:restriction>
</xsd:simpleType>
```

Użycie typu listowego:

```
<xsd:element name="ciąg" type="typLista10Liczb"/>
```

Listy wartości

Wariant definicji (z typem ograniczanym podanym jawnie w treści elementu <xsd:restriction>):

```
<xsd:simpleType name="typLista10Liczb">
  <xsd:restriction>
    <xsd:simpleType>
      <xsd:list itemType="xsd:integer"/>
    </xsd:simpleType>
    <xsd:length value="10"/>
  </xsd:restriction>
</xsd:simpleType>
```

Użycie typu listowego:

```
<xsd:element name="ciąg" type="typLista10Liczb"/>
```

```
<ciąg>0 1 1 2 3 5 8 13 21 34</ciąg>
```

Przykład: Lotto

```
<xsd:simpleType name="typLotto">
  <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>
```

Przykład: Lotto

```
<xsd:simpleType name="typLotto">
  <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>
```

Czego brakuje do „prawdziwego” Lotto?

Unie, czyli kombinacje

Unia to połączenie zakresów wartości kilku typów w jeden nowy zakres. Instancja może mieć przypisaną wartość należącą do dowolnego z połączonych zakresów wartości.

Unie, czyli kombinacje

Unia to połączenie zakresów wartości kilku typów w jeden nowy zakres. Instancja może mieć przypisaną wartość należącą do dowolnego z połączonych zakresów wartości.

Przykład definicji typu:

```
<xsd:simpleType name="typRozmiar">
  <xsd:union>
    <xsd:simpleType>
      <xsd:restriction base="xsd:integer">
        <xsd:minInclusive value="34"/>
        <xsd:maxInclusive value="62"/>
      </xsd:restriction>
    </xsd:simpleType>
```

Unie, czyli kombinacje

```
      <xsd:simpleType>
        <xsd:restriction base="xsd:token">
          <xsd:enumeration value="S"/>
          <xsd:enumeration value="M"/>
          <xsd:enumeration value="L"/>
          <xsd:enumeration value="XL"/>
          <xsd:enumeration value="XXL"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:union>
</xsd:simpleType>
```

Unie, czyli kombinacje

```
      <xsd:simpleType>
        <xsd:restriction base="xsd:token">
          <xsd:enumeration value="S"/>
          <xsd:enumeration value="M"/>
          <xsd:enumeration value="L"/>
          <xsd:enumeration value="XL"/>
          <xsd:enumeration value="XXL"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:union>
</xsd:simpleType>
```

Użycie:

```
<xsd:element name="rozmiar" type="typRozmiar"/>
<rozmiar>48</rozmiar>
<rozmiar>XL</rozmiar>
```

Unie — wariant definicji

```
<xsd:simpleType name="typLiczbowy">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="34"/>
    <xsd:maxInclusive value="62"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="typLiterowy">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="S"/>
    ...
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="typRozmiar">
  <xsd:union memberTypes="typLiczbowy typLiterowy"/>
</xsd:simpleType>
```

Stałe i domyślne wartości elementów

W schemacie XML Schema można (w DTD nie można) podawać stałą i domyślną zawartość elementów używając wykluczających się wzajemnie atrybutów `fixed` i `default`.

Stałe i domyślne wartości elementów

W schemacie XML Schema można (w DTD nie można) podawać stałą i domyślną zawartość elementów używając wykluczających się wzajemnie atrybutów `fixed` i `default`.

Jeśli w schemacie określono domyślną zawartość elementu, to:

- jeśli w dokumencie wystąpił element o niepustej zawartości, przysłoni ona wartość domyślną podaną w schemacie,

Stałe i domyślne wartości elementów

W schemacie XML Schema można (w DTD nie można) podawać stałą i domyślną zawartość elementów używając wykluczających się wzajemnie atrybutów `fixed` i `default`.

Jeśli w schemacie określono domyślną zawartość elementu, to:

- jeśli w dokumencie wystąpił element o niepustej zawartości, przysłoni ona wartość domyślną podaną w schemacie,
- jeśli wystąpił element o zawartości pustej, wynikowa zawartość elementu zostanie wypełniona wartością domyślną ze schematu,

Stałe i domyślne wartości elementów

W schemacie XML Schema można (w DTD nie można) podawać stałą i domyślną zawartość elementów używając wykluczających się wzajemnie atrybutów `fixed` i `default`.

Jeśli w schemacie określono domyślną zawartość elementu, to:

- jeśli w dokumencie wystąpił element o niepustej zawartości, przysłoni ona wartość domyślną podaną w schemacie,
- jeśli wystąpił element o zawartości pustej, wynikowa zawartość elementu zostanie wypełniona wartością domyślną ze schematu,
- jeśli element w ogóle nie wystąpił, wartość domyślna nie zostanie użyta w ogóle (parser nie stworzy elementu).

Stałe i domyślne wartości elementów

W schemacie XML Schema można (w DTD nie można) podawać stałą i domyślną zawartość elementów używając wykluczających się wzajemnie atrybutów `fixed` i `default`.

Jeśli w schemacie określono domyślną zawartość elementu, to:

- jeśli w dokumencie wystąpił element o niepustej zawartości, przysłoni ona wartość domyślną podaną w schemacie,
- jeśli wystąpił element o zawartości pustej, wynikowa zawartość elementu zostanie wypełniona wartością domyślną ze schematu,
- jeśli element w ogóle nie wystąpił, wartość domyślna nie zostanie użyta w ogóle (parser nie stworzy elementu).

Koncepcja wartości stałej jest identyczna: jeśli w dokumencie podano jakąś wartość, musi być ona równa wartości stałej, jeśli wartości nie podano, zostanie ona pobrana ze schematu.

Typy złożone

Typy złożone umożliwiają definiowanie modeli zawartości oraz dołączanie atrybutów.

Definicję typu złożonego tworzy element `<xsd:complexType>`, w którego treści zawarte są podelementy odpowiadające modelowi zawartości:

- `<xsd:sequence>` — sekwencja wystąpień elementów (przecinek z DTD),
- `<xsd:choice>` — alternatywa (| z DTD),
- `<xsd:group>` — grupa definicji (`(...)` z DTD).

Sekwencja i kontrola liczby wystąpień elementów

Określanie liczby wystąpień elementów odbywa się za pomocą atrybutów `minOccurs` i `maxOccurs`, które mogą przyjmować wartości naturalne rozszerzone o specjalną wartość `unbounded` oznaczającą nieograniczoną liczbę wystąpień.

Uwaga: wartość domyślna obu atrybutów: `1`.

Sekwencja i kontrola liczby wystąpień elementów

DTD:

```
<!ELEMENT osoba (tytuł?, imię+, nazwisko, adres*)>
```

Sekwencja i kontrola liczby wystąpień elementów

DTD:

```
<!ELEMENT osoba (tytuł?, imię+, nazwisko, adres*)>
```

XML Schema:

```
<xsd:element name="osoba">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="tytuł" minOccurs="0"/>
      <xsd:element name="imię"
                   maxOccurs="unbounded"/>
      <xsd:element name="nazwisko"/>
      <xsd:element name="adres" minOccurs="0"
                   maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Sekwencja i kontrola liczby wystąpień elementów

DTD:

```
<!ELEMENT osoba (tytuł?, imię+, nazwisko, adres*)>
```

XML Schema:

```
<xsd:element name="osoba">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="tytuł" minOccurs="0"/>
      <xsd:element name="imię"
                   maxOccurs="unbounded"/>
      <xsd:element name="nazwisko"/>
      <xsd:element name="adres" minOccurs="0"
                   maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Sekwencja i kontrola liczby wystąpień elementów

DTD:

```
<!ELEMENT osoba (tytuł?, imię+, nazwisko, adres*)>
```

XML Schema:

```
<xsd:element name="osoba">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="tytuł" minOccurs="0"/>
      <xsd:element name="imię"
                   maxOccurs="unbounded"/>
      <xsd:element name="nazwisko"/>
      <xsd:element name="adres" minOccurs="0"
                   maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Sekwencja i kontrola liczby wystąpień elementów

DTD:

```
<!ELEMENT osoba (tytuł?, imię+, nazwisko, adres*)>
```

XML Schema:

```
<xsd:element name="osoba">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="tytuł" minOccurs="0"/>
      <xsd:element name="imię"
                   maxOccurs="unbounded"/>
      <xsd:element name="nazwisko"/>
      <xsd:element name="adres" minOccurs="0"
                   maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Definicje elementów: przykłady

1. `<xsd:element name="a" minOccurs="2" maxOccurs="5"/>`

Definicje elementów: przykłady

1. `<xsd:element name="a" minOccurs="2" maxOccurs="5"/>`
   Element może wystąpić od 2 do 5 razy, może zawierać dowolną wartość.

Definicje elementów: przykłady

1. `<xsd:element name="a" minOccurs="2" maxOccurs="5"/>`
   Element może wystąpić od 2 do 5 razy, może zawierać dowolną wartość.
2. `<xsd:element name="b"/>`

Definicje elementów: przykłady

1. `<xsd:element name="a" minOccurs="2" maxOccurs="5"/>`
   Element może wystąpić od 2 do 5 razy, może zawierać dowolną wartość.
2. `<xsd:element name="b"/>`
   Element musi wystąpić raz, może zawierać dowolną wartość.

Definicje elementów: przykłady

1. `<xsd:element name="a" minOccurs="2" maxOccurs="5"/>`
   Element może wystąpić od 2 do 5 razy, może zawierać dowolną wartość.
2. `<xsd:element name="b"/>`
   Element musi wystąpić raz, może zawierać dowolną wartość.
3. `<xsd:element name="c" minOccurs="0" fixed="24"/>`

Definicje elementów: przykłady

1. `<xsd:element name="a" minOccurs="2" maxOccurs="5"/>`
   Element może wystąpić od 2 do 5 razy, może zawierać dowolną wartość.
2. `<xsd:element name="b"/>`
   Element musi wystąpić raz, może zawierać dowolną wartość.
3. `<xsd:element name="c" minOccurs="0" fixed="24"/>`
   Element może wystąpić co najwyżej raz. Jeśli nie wystąpi, parser nie stworzy elementu. Jeśli wystąpi jako element pusty, jego zawartość zostanie wypełniona wartością 24. Jeśli wystąpi jako element niepusty, jego zawartością musi być tekst 24.

Definicje elementów: przykłady

1. `<xsd:element name="a" minOccurs="2" maxOccurs="5"/>`
   Element może wystąpić od 2 do 5 razy, może zawierać dowolną wartość.
2. `<xsd:element name="b"/>`
   Element musi wystąpić raz, może zawierać dowolną wartość.
3. `<xsd:element name="c" minOccurs="0" fixed="24"/>`
   Element może wystąpić co najwyżej raz. Jeśli nie wystąpi, parser nie stworzy elementu. Jeśli wystąpi jako element pusty, jego zawartość zostanie wypełniona wartością 24. Jeśli wystąpi jako element niepusty, jego zawartością musi być tekst 24.
4. `<xsd:element name="d" maxOccurs="unbounded" default="24"/>`

Definicje elementów: przykłady

1. `<xsd:element name="a" minOccurs="2" maxOccurs="5"/>`
   Element może wystąpić od 2 do 5 razy, może zawierać dowolną wartość.
2. `<xsd:element name="b"/>`
   Element musi wystąpić raz, może zawierać dowolną wartość.
3. `<xsd:element name="c" minOccurs="0" fixed="24"/>`
   Element może wystąpić co najwyżej raz. Jeśli nie wystąpi, parser nie stworzy elementu. Jeśli wystąpi jako element pusty, jego zawartość zostanie wypełniona wartością 24. Jeśli wystąpi jako element niepusty, jego zawartością musi być tekst 24.
4. `<xsd:element name="d" maxOccurs="unbounded" default="24"/>`
   Element musi wystąpić co najmniej raz. Jeśli wystąpi jako element pusty, jego zawartość zostanie wypełniona wartością 24; jeśli jako niepusty, wynikiem będzie podana wartość.

Alternatywa elementów (jeden z listy)

DTD:

```
<!ELEMENT pojazd (pociąg | samolot | samochód)>
```

Alternatywa elementów (jeden z listy)

DTD:

```
<!ELEMENT pojazd (pociąg | samolot | samochód)>
```

XML Schema:

```
<xsd:element name="pojazd">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element name="pociąg"/>
      <xsd:element name="samolot"/>
      <xsd:element name="samochód"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Alternatywa elementów (jeden z listy)

DTD:

```
<!ELEMENT pojazd (pociąg | samolot | samochód)>
```

XML Schema:

```
<xsd:element name="pojazd">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element name="pociąg"/>
      <xsd:element name="samolot"/>
      <xsd:element name="samochód"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Dowolna kombinacja elementów

DTD:

```
<!ELEMENT pojazd (pociąg | samolot | samochód)*>
```

Dowolna kombinacja elementów

DTD:

```
<!ELEMENT pojazd (pociąg | samolot | samochód)*>
```

XML Schema:

```
<xsd:element name="pojazd">
  <xsd:complexType>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="pociąg"/>
      <xsd:element name="samolot"/>
      <xsd:element name="samochód"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Dowolna kombinacja elementów

DTD:

```
<!ELEMENT pojazd (pociąg | samolot | samochód)*>
```

XML Schema:

```
<xsd:element name="pojazd">
  <xsd:complexType>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="pociąg"/>
      <xsd:element name="samolot"/>
      <xsd:element name="samochód"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Dowolna kolejność elementów

DTD:

Nie da się określić dowolnej kolejności elementów!

Dowolna kolejność elementów

DTD:

Nie da się określić dowolnej kolejności elementów!

XML Schema:

```
<xsd:element name="książka">
  <xsd:complexType>
    <xsd:all>
      <xsd:element name="tytuł"/>
      <xsd:element name="autor"/>
    </xsd:all>
  </xsd:complexType>
</xsd:element>
```

Dowolna kolejność elementów

DTD:

Nie da się określić dowolnej kolejności elementów!

XML Schema:

```
<xsd:element name="książka">
  <xsd:complexType>
    <xsd:all>
      <xsd:element name="tytuł"/>
      <xsd:element name="autor"/>
    </xsd:all>
  </xsd:complexType>
</xsd:element>
```

Dowolna kolejność elementów

Ograniczenia:

- nie może zawierać innych grup (tylko deklaracje elementów i odwołania do elementów),
- każdy element może wystąpić co najwyżej raz,
- grupa `all` nie może być zagnieżdżona w innej grupie,
- grupę można łączyć z zawartością mieszaną.

Zawartość pusta

DTD:

```
<!ELEMENT hr EMPTY>
```

Zawartość pusta

DTD:

```
<!ELEMENT hr EMPTY>
```

XML Schema:

```
<xsd:element name="hr">
  <xsd:complexType/>
</xsd:element>
```

Dowolna zawartość

Dopuszczenie zawartości dowolnej ma zwykle na celu włączenie do dokumentu elementów z innej przestrzeni nazw (np. zagnieżdżenie fragmentu XHTML-a, MathML-a czy SVG).

Dowolna zawartość

Dopuszczenie zawartości dowolnej ma zwykle na celu włączenie do dokumentu elementów z innej przestrzeni nazw (np. zagnieżdżenie fragmentu XHTML-a, MathML-a czy SVG).

XML Schema udostępnia do tego celu element `<xsd:any>` z dwoma „specjalizowanymi” atrybutami:

- `namespace`, określającym przestrzeń nazw włączanej zawartości,
- `processContents`, określającym poziom walidacji włączanej zawartości i przyjmującym wartości:
  - `skip` — nie waliduj,
  - `lax` — waliduj wyłącznie elementy, dla których określono przestrzenie nazw,
  - `strict` — wykonaj pełną walidację (wartość domyślna).

Dowolna zawartość

DTD:

```
<!ELEMENT kodXHTML ANY>
```

Dowolna zawartość

DTD:

```
<!ELEMENT kodXHTML ANY>
```

XML Schema:

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

Dowolne atrybuty

Istnieje także możliwość dopuszczenia wystąpienia w elemencie dowolnych atrybutów (dzięki czemu można np. skorzystać z zaawansowanych mechanizmów linkowania oferowanych przez standard XLink).

Przykład użycia:

```
<xsd:element name="osoba">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="stronaWWW">
        <xsd:complexType>
          <xsd:anyAttribute
              namespace="http://www.w3.org/1999/xlink"/>
        </xsd:complexType>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Definicja atrybutów

Kontrola użycia atrybutów realizowana jest przy pomocy opcjonalnego atrybutu `use` o dopuszczalnych wartościach:

- `required` — atrybut jest wymagany,
- `optional` (domyślnie) — atrybut nie musi wystąpić,
- `prohibited` — atrybut nie może wystąpić (przy ograniczaniu definicji typów, o czym na następnym wykładzie).

Definicja atrybutów

DTD:

```
<!ELEMENT osoba EMPTY>
<!ATTLIST osoba pesel CDATA #REQUIRED
                nip CDATA #IMPLIED>
                gatunek CDATA #FIXED "Homo sapiens">
```

Definicja atrybutów

DTD:

```
<!ELEMENT osoba EMPTY>
<!ATTLIST osoba pesel CDATA #REQUIRED
                nip CDATA #IMPLIED>
                gatunek CDATA #FIXED "Homo sapiens">
```

XML Schema:

```
<xsd:element name="osoba">
  <xsd:complexType>
    <xsd:attribute name="pesel" use="required"/>
    <xsd:attribute name="nip"/>
    <xsd:attribute name="gatunek"
                   fixed="Homo sapiens"/>
  </xsd:complexType>
</xsd:element>
```

Definicja atrybutów

DTD:

```
<!ELEMENT osoba EMPTY>
<!ATTLIST osoba pesel CDATA #REQUIRED
                nip CDATA #IMPLIED>
                gatunek CDATA #FIXED "Homo sapiens">
```

XML Schema:

```
<xsd:element name="osoba">
  <xsd:complexType>
    <xsd:attribute name="pesel" use="required"/>
    <xsd:attribute name="nip"/>
    <xsd:attribute name="gatunek"
                   fixed="Homo sapiens"/>
  </xsd:complexType>
</xsd:element>
```

Definicja atrybutów

DTD:

```
<!ELEMENT osoba EMPTY>
<!ATTLIST osoba pesel CDATA #REQUIRED
                nip CDATA #IMPLIED>
                gatunek CDATA #FIXED "Homo sapiens">
```

XML Schema:

```
<xsd:element name="osoba">
  <xsd:complexType>
    <xsd:attribute name="pesel" use="required"/>
    <xsd:attribute name="nip"/>
    <xsd:attribute name="gatunek"
                   fixed="Homo sapiens"/>
  </xsd:complexType>
</xsd:element>
```

Definicja atrybutów

DTD:

```
<!ELEMENT osoba EMPTY>
<!ATTLIST osoba pesel CDATA #REQUIRED
                nip CDATA #IMPLIED>
                gatunek CDATA #FIXED "Homo sapiens">
```

XML Schema:

```
<xsd:element name="osoba">
  <xsd:complexType>
    <xsd:attribute name="pesel" use="required"/>
    <xsd:attribute name="nip"/>
    <xsd:attribute name="gatunek"
                   fixed="Homo sapiens"/>
  </xsd:complexType>
</xsd:element>
```

Definicja atrybutów

DTD:

```
<!ATTLIST osoba płeć (kobieta | mężczyzna) #REQUIRED>
```

Definicja atrybutów

DTD:

```
<!ATTLIST osoba płeć (kobieta | mężczyzna) #REQUIRED>
```

XML Schema:

```
<xsd:element name="osoba">
  <xsd:complexType>
    <xsd:attribute name="płeć" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="kobieta"/>
          <xsd:enumeration value="mężczyzna"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
</xsd:element>
```

Definicja atrybutów

DTD:

```
<!ATTLIST osoba płeć (kobieta | mężczyzna) #REQUIRED>
```

XML Schema:

```
<xsd:element name="osoba">
  <xsd:complexType>
    <xsd:attribute name="płeć" use="required">
      <xsd:simpleType>
        <xsd:restriction base="xsd:NMTOKEN">
          <xsd:enumeration value="kobieta"/>
          <xsd:enumeration value="mężczyzna"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:attribute>
  </xsd:complexType>
</xsd:element>
```

Stałe i domyślne wartości atrybutów

Podobnie jak w przypadku elementów, do określania stałych i domyślnych wartości atrybutów używane są wykluczające się wzajemnie atrybuty `fixed` i `default`.

Stałe i domyślne wartości atrybutów

Podobnie jak w przypadku elementów, do określania stałych i domyślnych wartości atrybutów używane są wykluczające się wzajemnie atrybuty `fixed` i `default`.

Jeśli w schemacie określono atrybut domyślny, to:

- jeśli w dokumencie wartość tego atrybutu została podana, przysłoni ona wartość podaną w schemacie,
- jeśli wartość atrybutu nie została podana, parser pobierze wartość domyślną ze schematu.

Wartość domyślną można podać wyłącznie dla atrybutów opcjonalnych.

Stałe i domyślne wartości atrybutów

Podobnie jak w przypadku elementów, do określania stałych i domyślnych wartości atrybutów używane są wykluczające się wzajemnie atrybuty `fixed` i `default`.

Jeśli w schemacie określono atrybut domyślny, to:

- jeśli w dokumencie wartość tego atrybutu została podana, przysłoni ona wartość podaną w schemacie,
- jeśli wartość atrybutu nie została podana, parser pobierze wartość domyślną ze schematu.

Wartość domyślną można podać wyłącznie dla atrybutów opcjonalnych.

Koncepcja wartości stałej jest identyczna: jeśli w dokumencie podano jakąś wartość, musi być ona równa wartości stałej, jeśli wartości nie podano, zostanie ona pobrana ze schematu.

Definicje atrybutów: przykłady

1. `<xsd:attribute name="a" use="required"/>`

Definicje atrybutów: przykłady

1. `<xsd:attribute name="a" use="required"/>`
   Atrybut musi wystąpić, może przyjąć dowolną wartość.

Definicje atrybutów: przykłady

1. `<xsd:attribute name="a" use="required"/>`
   Atrybut musi wystąpić, może przyjąć dowolną wartość.

2. `<xsd:attribute name="b"/>`

Definicje atrybutów: przykłady

1. `<xsd:attribute name="a" use="required"/>`
   Atrybut musi wystąpić, może przyjąć dowolną wartość.

2. `<xsd:attribute name="b"/>`
   Atrybut jest opcjonalny, może przyjąć dowolną wartość.

Definicje atrybutów: przykłady

1. `<xsd:attribute name="a" use="required"/>`
   Atrybut musi wystąpić, może przyjąć dowolną wartość.

2. `<xsd:attribute name="b"/>`
   Atrybut jest opcjonalny, może przyjąć dowolną wartość.

3. `<xsd:attribute name="c" fixed="24"/>`

Definicje atrybutów: przykłady

1. `<xsd:attribute name="a" use="required"/>`
   Atrybut musi wystąpić, może przyjąć dowolną wartość.

2. `<xsd:attribute name="b"/>`
   Atrybut jest opcjonalny, może przyjąć dowolną wartość.

3. `<xsd:attribute name="c" fixed="24"/>`
   Atrybut jest opcjonalny. Jeśli nie zostanie podany w dokumencie, jego wartość wyniesie 24; jeśli zostanie podany, musi mieć wartość 24.

Definicje atrybutów: przykłady

1. `<xsd:attribute name="a" use="required"/>`
   Atrybut musi wystąpić, może przyjąć dowolną wartość.

2. `<xsd:attribute name="b"/>`
   Atrybut jest opcjonalny, może przyjąć dowolną wartość.

3. `<xsd:attribute name="c" fixed="24"/>`
   Atrybut jest opcjonalny. Jeśli nie zostanie podany w dokumencie, jego wartość wyniesie 24; jeśli zostanie podany, musi mieć wartość 24.

4. `<xsd:attribute name="d" default="24"/>`

Definicje atrybutów: przykłady

1. `<xsd:attribute name="a" use="required"/>`
   Atrybut musi wystąpić, może przyjąć dowolną wartość.

2. `<xsd:attribute name="b"/>`
   Atrybut jest opcjonalny, może przyjąć dowolną wartość.

3. `<xsd:attribute name="c" fixed="24"/>`
   Atrybut jest opcjonalny. Jeśli nie zostanie podany w dokumencie, jego wartość wyniesie 24; jeśli zostanie podany, musi mieć wartość 24.

4. `<xsd:attribute name="d" default="24"/>`
   Atrybut jest opcjonalny. Jeśli wartość atrybutu nie zostanie podana, zostanie mu przypisana wartość 24. Jeśli wartość zostanie podana, wynikowa wartość atrybutu będzie taka jak w dokumencie.

Model mieszany

DTD:

```
<!ELEMENT tekst (#PCDATA | wyróżnienie)*>
```

Model mieszany

DTD:

```
<!ELEMENT tekst (#PCDATA | wyróżnienie)*>
```

XML Schema:

```
<xsd:element name="tekst">
  <xsd:complexType mixed="true">
    <xsd:choice>
      <xsd:element name="wyróżnienie" minOccurs="0"
                   maxOccurs="unbounded"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Model mieszany

DTD:

```
<!ELEMENT tekst (#PCDATA | wyróżnienie)*>
```

XML Schema:

```
<xsd:element name="tekst">
  <xsd:complexType mixed="true">
    <xsd:choice>
      <xsd:element name="wyróżnienie" minOccurs="0"
                   maxOccurs="unbounded"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Model mieszany: co wolno w XML Schema?

Trochę więcej niż w DTD — można określać:

- porządek wystąpienia elementów
  (nie tylko `<xsd:choice>`, ale także `<xsd:sequence>`),
- liczbę wystąpień elementów w treści mieszanej
  (używając `minOccurs` i `maxOccurs`).

Deklaracje globalne i lokalne

Deklaracje będące bezpośrednimi podwęzłami elementu głównego `<xsd:schema>` są globalne, pozostałe — lokalne.

Deklaracje globalne i lokalne

Deklaracje będące bezpośrednimi podwęzłami elementu głównego <xsd:schema> są globalne, pozostałe — lokalne.

Deklaracji globalnych można używać w dowolnym miejscu schematu podając odwołanie do definicji w atrybucie `ref`:

```
<xsd:element name="przypis">
  <xsd:complexType>
    <xsd:attribute name="nr"/>
  </xsd:complexType>
</xsd:element>
<xsd:element name="tytuł">
  <xsd:complexType mixed="true">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element ref="przypis"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Grupy elementów

DTD:

```
<!ENTITY % elem-formatowania 'b | i | u'>
```

Grupy elementów

DTD:

```
<!ENTITY % elem-formatowania 'b | i | u'>
```

XML Schema:

```
<xsd:group name="elem-formatowania">
  <xsd:choice>
    <xsd:element ref="b"/>
    <xsd:element ref="i"/>
    <xsd:element ref="u"/>
  </xsd:choice>
</xsd:group>
```

Grupy elementów

DTD:

```
<!ENTITY % elem-formatowania 'b | i | u'>
```

XML Schema:

```
<xsd:group name="elem-formatowania">
  <xsd:choice>
    <xsd:element ref="b"/>
    <xsd:element ref="i"/>
    <xsd:element ref="u"/>
  </xsd:choice>
</xsd:group>
```

Grupy elementów

DTD:

```
<!ENTITY % elem-formatowania 'b | i | u'>
```

XML Schema:

```
<xsd:group name="elem-formatowania">
  <xsd:choice>
    <xsd:element ref="b"/>
    <xsd:element ref="i"/>
    <xsd:element ref="u"/>
  </xsd:choice>
</xsd:group>
```

Uwaga: Grupy muszą być nazwane i globalne.

Grupy atrybutów

DTD:

```
<!ENTITY % atr-czasowe 'ważne-od CDATA #IMPLIED
                        ważne-do CDATA #IMPLIED'>

<!ENTITY % atr-wspólne 'status CDATA #IMPLIED
                        %atr-czasowe;'>
```

Grupy atrybutów

DTD:

```
<!ENTITY % atr-czasowe 'ważne-od CDATA #IMPLIED
                        ważne-do CDATA #IMPLIED'>

<!ENTITY % atr-wspólne 'status CDATA #IMPLIED
                        %atr-czasowe;'>
```

XML Schema:

```
<xsd:attributeGroup name="atr-czasowe">
  <xsd:attribute name="obowiązuje-od"/>
  <xsd:attribute name="obowiązuje-do"/>
</xsd:attributeGroup

<xsd:attributeGroup name="atr-wspólne">
  <xsd:attribute name="status"/>
  <xsd:attributeGroup ref="atr-czasowe"/>
</xsd:attributeGroup
```

Grupy atrybutów

DTD:

```
<!ENTITY % atr-czasowe 'ważne-od CDATA #IMPLIED
                        ważne-do CDATA #IMPLIED'>

<!ENTITY % atr-wspólne 'status CDATA #IMPLIED
                        %atr-czasowe;'>
```

XML Schema:

```
<xsd:attributeGroup name="atr-czasowe">
  <xsd:attribute name="obowiązuje-od"/>
  <xsd:attribute name="obowiązuje-do"/>
</xsd:attributeGroup

<xsd:attributeGroup name="atr-wspólne">
  <xsd:attribute name="status"/>
  <xsd:attributeGroup ref="atr-czasowe"/>
</xsd:attributeGroup
```

Definiowanie wartości niepowtarzalnych

Dwa sposoby:

Definiowanie wartości niepowtarzalnych

Dwa sposoby:

1. za pomocą atrybutów typu `xsd:ID` i `xsd:IDREF` — jak w DTD,

Definiowanie wartości niepowtarzalnych

Dwa sposoby:

1. za pomocą atrybutów typu `xsd:ID` i `xsd:IDREF` — jak w DTD,
2. za pomocą ograniczeń integralności (ang. *identity constraints*):

Definiowanie wartości niepowtarzalnych

Dwa sposoby:

1. za pomocą atrybutów typu `xsd:ID` i `xsd:IDREF` — jak w DTD,
2. za pomocą ograniczeń integralności (ang. *identity constraints*):
   - kluczy `<xsd:key>`,

Definiowanie wartości niepowtarzalnych

Dwa sposoby:

1. za pomocą atrybutów typu `xsd:ID` i `xsd:IDREF` — jak w DTD,
2. za pomocą ograniczeń integralności (ang. *identity constraints*):
   - kluczy `<xsd:key>`,
   - odwołań do kluczy `<xsd:keyref>`,

Definiowanie wartości niepowtarzalnych

Dwa sposoby:

1. za pomocą atrybutów typu `xsd:ID` i `xsd:IDREF` — jak w DTD,
2. za pomocą ograniczeń integralności (ang. *identity constraints*):
   - kluczy `<xsd:key>`,
   - odwołań do kluczy `<xsd:keyref>`,
   - wartości unikatowych `<xsd:unique>`.

Więzy integralności: definicja

Definicja ograniczenia integralności składa się z trzech części:

Więzy integralności: definicja

Definicja ograniczenia integralności składa się z trzech części:

1. **zasięg** (ang. *scope*) jest określony przez element, w którego deklaracji zdefiniowano ograniczenie,

Więzy integralności: definicja

Definicja ograniczenia integralności składa się z trzech części:

1. **zasięg** (ang. *scope*) jest określony przez element, w którego deklaracji zdefiniowano ograniczenie,
2. **selektor** (ang. *selector*) pozwala wskazać węzły, których dotyczy ograniczenie,

Więzy integralności: definicja

Definicja ograniczenia integralności składa się z trzech części:

1. **zasięg** (ang. *scope*) jest określony przez element, w którego deklaracji zdefiniowano ograniczenie,
2. **selektor** (ang. *selector*) pozwala wskazać węzły, których dotyczy ograniczenie,
3. definicja jednego lub więcej **pól** (ang. *fields*) wskazuje elementy i atrybuty, których wartości muszą być unikatowe wśród węzłów wybranych selektorem. Uwaga: w każdym węźle wskazanym przez selektor może być co najwyżej jeden egzemplarz każdego pola.

Więzy integralności: definicja

Definicja ograniczenia integralności składa się z trzech części:

1. **zasięg** (ang. *scope*) jest określony przez element, w którego deklaracji zdefiniowano ograniczenie,
2. **selektor** (ang. *selector*) pozwala wskazać węzły, których dotyczy ograniczenie,
3. definicja jednego lub więcej **pól** (ang. *fields*) wskazuje elementy i atrybuty, których wartości muszą być unikatowe wśród węzłów wybranych selektorem. Uwaga: w każdym węźle wskazanym przez selektor może być co najwyżej jeden egzemplarz każdego pola.

Selektory i pola określa się przy pomocy ograniczonego podzbioru wyrażeń XPath.

Zostań guru XPath w 30 sekund

XPath to język do zapisu ścieżek do elementów i atrybutów.

Zostań guru XPath w 30 sekund

XPath to język do zapisu ścieżek do elementów i atrybutów.

Składnia — prawie jak w systemie plików:

Zostań guru XPath w 30 sekund

XPath to język do zapisu ścieżek do elementów i atrybutów.

Składnia — prawie jak w systemie plików:

- `książka/rozdział` — wskaż/wybierz podelementy typu `rozdział` z elementu `książka` będącego podelementem bieżącego elementu,

Zostań guru XPath w 30 sekund

XPath to język do zapisu ścieżek do elementów i atrybutów.

Składnia — prawie jak w systemie plików:

- `książka/rozdział` — wskaż/wybierz podelementy typu `rozdział` z elementu `książka` będącego podelementem bieżącego elementu,
- `*/rozdział` — podelementy typu `rozdział` z dowolnych podelementów bieżącego elementu,

Zostań guru XPath w 30 sekund

XPath to język do zapisu ścieżek do elementów i atrybutów.

Składnia — prawie jak w systemie plików:

- `książka/rozdział` — wskaż/wybierz podelementy typu `rozdział` z elementu `książka` będącego podelementem bieżącego elementu,
- `*/rozdział` — podelementy typu `rozdział` z dowolnych podelementów bieżącego elementu,
- `@nr` — wartość atrybutu `nr` bieżącego elementu.

Ograniczenia integralności: unikalny identyfikator

```
<księgowość>
  <zamówienia>
    <zamówienie>
      <numer>125</numer>
      ...
    </zamówienie>
    <zamówienie>
      <numer>665</numer>
      ...
    </zamówienie>
  </zamówienia>
  <faktury>
  ...
  </faktury>
</księgowość>
```

Ograniczenia integralności: unikalny identyfikator

```
<księgowość>
  <zamówienia>
    <zamówienie>
      <numer>125</numer>
      ...
    </zamówienie>
    <zamówienie>
      <numer>665</numer>
      ...
    </zamówienie>
  </zamówienia>
  <faktury>
  ...
  </faktury>
</księgowość>
```

Unikalność numerów zamówień w całym dokumencie; numery zamówień bedą przywoływane na fakturach:

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

Ograniczenia integralności: unikalny identyfikator

```
<księgowość>
  <zamówienia>
    <zamówienie>
      <numer>125</numer>
      ...
    </zamówienie>
    <zamówienie>
      <numer>665</numer>
      ...
    </zamówienie>
  </zamówienia>
  <faktury>
  ...
  </faktury>
</księgowość>
```

Unikalność numerów zamówień w całym dokumencie; numery zamówień bedą przywoływane na fakturach:

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

Ograniczenia integralności: unikalny identyfikator

```
<księgowość>
  <zamówienia>
    <zamówienie>
      <numer>125</numer>
      ...
    </zamówienie>
    <zamówienie>
      <numer>665</numer>
      ...
    </zamówienie>
  </zamówienia>
  <faktury>
  ...
  </faktury>
</księgowość>
```

Unikalność numerów zamówień w całym dokumencie; numery zamówień bedą przywoływane na fakturach:

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

Ograniczenia integralności: unikalne numery i odwołania

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

Ograniczenia integralności: unikalne numery i odwołania

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

Unikalność numerów faktur
w danym roku w całym dokumencie:

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

Ograniczenia integralności: unikalne numery i odwołania

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

Unikalność numerów faktur
w danym roku w całym dokumencie:

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

Ograniczenia integralności: unikalne numery i odwołania

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

Unikalność numerów faktur
w danym roku w całym dokumencie:

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

Ograniczenia integralności: unikalne numery i odwołania

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

Odwołanie do numeru zamówienia:

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

Ograniczenia integralności: unikalne numery i odwołania

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

Odwołanie do numeru zamówienia:

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

Ograniczenia integralności: unikalne numery i odwołania

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

Odwołanie do numeru zamówienia:

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

Ograniczenia integralności: cała definicja

```
<xsd:element name="księgowość" type="typKsięgowość">
  <xsd:unique name="unikalnyNrFaktury">
    <xsd:selector xpath="faktury/faktura"/>
    <xsd:field xpath="@nr"/>
    <xsd:field xpath="rok"/>
  </xsd:unique>
  <xsd:keyref name="odwołanieDoZamówienia"
              refer="nrZamówienia">
    <xsd:selector xpath="faktury/faktura/do-zamówienia"/>
    <xsd:field xpath="@nr"/>
  </xsd:keyref>
  <xsd:key name="nrZamówienia">
    <xsd:selector xpath="zamówienia/zamówienie"/>
    <xsd:field xpath="numer"/>
  </xsd:key>
</xsd:element>
```