

XML w sosie własnym

Standard XML wraz z DTD, przestrzenie nazw,
projektowanie struktury dokumentów.

Patryk Czarnik

Instytut Informatyki UW

XML i nowoczesne technologie zarządzania treścią – 2007/08

1 Podstawy XML

- Dokument XML – warstwa logiczna
- Dokument XML – warstwa fizyczna
- Standardy kodowania znaków

2 DTD

- Motywacja dla definiowania struktury dokumentów
- DTD – elementy i atrybuty
- DTD – encje i parametryzacja

3 Przestrzenie nazw

- Motywacja
- Realizacja

Plan

- 1 Podstawy XML
 - Dokument XML – warstwa logiczna
 - Dokument XML – warstwa fizyczna
 - Standardy kodowania znaków

- 2 DTD
 - Motywacja dla definiowania struktury dokumentów
 - DTD – elementy i atrybuty
 - DTD – encje i parametryzacja

- 3 Przestrzenie nazw
 - Motywacja
 - Realizacja

Model dokumentu XML

- Opisany bardziej formalnie m.in. w standardach XML Information Set, XPath, DOM.
- Dokument jest drzewem:
 - elementy – węzły drzewa, posiadają nazwę, mogą posiadać atrybuty,
 - węzły tekstowe – liście drzewa,
 - ponadto komentarze i instrukcje przetwarzania – także liście.
- **Atrybuty** to własności elementów identyfikowane po nazwie, nie są dziećmi elementów.

Model dokumentu XML

- Opisany bardziej formalnie m.in. w standardach XML Information Set, XPath, DOM.
- Dokument jest drzewem:
 - **elementy** – węzły drzewa, posiadają nazwę, mogą posiadać atrybuty,
 - **węzły tekstowe** – liście drzewa,
 - ponadto **komentarze**
i **instrukcje przetwarzania** – także liście.
- **Atrybuty** to własności elementów identyfikowane po nazwie, nie są dziećmi elementów.

Model dokumentu XML

- Opisany bardziej formalnie m.in. w standardach XML Information Set, XPath, DOM.
- Dokument jest drzewem:
 - **elementy** – węzły drzewa, posiadają nazwę, mogą posiadać atrybuty,
 - **węzły tekstowe** – liście drzewa,
 - ponadto **komentarze**
i **instrukcje przetwarzania** – także liście.
- **Atrybuty** to własności elementów identyfikowane po nazwie, nie są dziećmi elementów.

Model dokumentu XML

- Opisany bardziej formalnie m.in. w standardach XML Information Set, XPath, DOM.
- Dokument jest drzewem:
 - **elementy** – węzły drzewa, posiadają nazwę, mogą posiadać atrybuty,
 - **węzły tekstowe** – liście drzewa,
 - ponadto **komentarze**
i **instrukcje przetwarzania** – także liście.
- **Atrybuty** to własności elementów identyfikowane po nazwie, nie są dziećmi elementów.

Model dokumentu XML

- Opisany bardziej formalnie m.in. w standardach XML Information Set, XPath, DOM.
- Dokument jest drzewem:
 - **elementy** – węzły drzewa, posiadają nazwę, mogą posiadać atrybuty,
 - **węzły tekstowe** – liście drzewa,
 - ponadto **komentarze**
i **instrukcje przetwarzania** – także liście.
- **Atrybuty** to własności elementów identyfikowane po nazwie, nie są dziećmi elementów.

Model dokumentu XML

- Opisany bardziej formalnie m.in. w standardach XML Information Set, XPath, DOM.
- Dokument jest drzewem:
 - **elementy** – węzły drzewa, posiadają nazwę, mogą posiadać atrybuty,
 - **węzły tekstowe** – liście drzewa,
 - ponadto **komentarze**
i **instrukcje przetwarzania** – także liście.
- **Atrybuty** to własności elementów identyfikowane po nazwie, nie są dziećmi elementów.

Zapisywanie struktury dokumentu, znaczniki

```
<artykuł id="13/2007" xml:lang="pl">
  <tytuł>Przypisanie w Pascalu i C </tytuł>
  <autor email='ola@fasola.pl'>Ola Fasola</autor>
  <treść>
    W Pascalu przypisanie ma postać <kod>x := 5</kod>,
    natomiast w C <kod>x = 5</kod>.

  </treść>
</artykuł>
```

- Element główny

Zapisywanie struktury dokumentu, znaczniki

```
<artykuł id="13/2007" xml:lang="pl">
  <tytuł>Przypisanie w Pascalu i C </tytuł>
  <autor email='ola@fasola.pl'>Ola Fasola</autor>
  <treść>
    W Pascalu przypisanie ma postać <kod>x := 5</kod>,
    natomiast w C <kod>x = 5</kod>.

  </treść>
</artykuł>
```

- Element

Zapisywanie struktury dokumentu, znaczniki

```
<artykuł id="13/2007" xml:lang="pl">
  <tytuł>Przypisanie w Pascalu i C </tytuł>
  <autor email='ola@fasola.pl'>Ola Fasola</autor>
  <treść>
    W Pascalu przypisanie ma postać <kod>x := 5</kod>,
    natomiast w C <kod>x = 5</kod>.

  </treść>
</artykuł>
```

- Znacznik otwierający

Zapisywanie struktury dokumentu, znaczniki

```
<artykuł id="13/2007" xml:lang="pl">
  <tytuł>Przypisanie w Pascalu i C </tytuł>
  <autor email='ola@fasola.pl'>Ola Fasola</autor>
  <treść>
    W Pascalu przypisanie ma postać <kod>x := 5</kod>,
    natomiast w C <kod>x = 5</kod>.

  </treść>
</artykuł>
```

- Znacznik zamykający

Zapisywanie struktury dokumentu, znaczniki

```
<artykuł id="13/2007" xml:lang="pl">
  <tytuł>Przypisanie w Pascalu i C </tytuł>
  <autor email='ola@fasola.pl'>Ola Fasola</autor>
  <treść>
    W Pascalu przypisanie ma postać <kod>x := 5</kod>,
    natomiast w C <kod>x = 5</kod>.

  </treść>
</artykuł>
```

- Atrybuty

Zapisywanie struktury dokumentu, znaczniki

```
<artykuł id="13/2007" xml:lang="pl">
  <tytuł>Przypisanie w Pascalu i C </tytuł>
  <autor email='ola@fasola.pl'>Ola Fasola</autor>
  <treść>
    W Pascalu przypisanie ma postać <kod>x := 5</kod>,
    natomiast w C <kod>x = 5</kod>.

  </treść>
</artykuł>
```

- Węzeł tekstowy

Zapisywanie struktury dokumentu, znaczniki

```
<artykuł id="13/2007" xml:lang="pl">
  <tytuł>Przypisanie w Pascalu i C </tytuł>
  <autor email='ola@fasola.pl'>Ola Fasola</autor>
  <treść>
    W Pascalu przypisanie ma postać <kod>x := 5</kod>,
    natomiast w C <kod>x = 5</kod>.

  </treść>
</artykuł>
```

- Zawartość mieszana

Zapisywanie struktury dokumentu, znaczniki

```
<artykuł id="13/2007" xml:lang="pl">
  <tytuł>Przypisanie w Pascalu i C </tytuł>
  <autor email='ola@fasola.pl'>Ola Fasola</autor>
  <treść>
    W Pascalu przypisanie ma postać <kod>x := 5</kod>,
    natomiast w C <kod>x = 5</kod>.
    <!-- TODO: Dopisać o Javie i Ocamlu -->
  </treść>
</artykuł>
<!--recenzent: podać ciekawszy przykład -->
```

- Komentarze

Zapisywanie struktury dokumentu, znaczniki

```
<?xml-stylesheet type="text/css" href="styl.css"?>
<artykuł id="13/2007" xml:lang="pl">
  <tytuł>Przypisanie w Pascalu i C </tytuł>
  <autor email='ola@fasola.pl'>Ola Fasola</autor>
  <treść><?drukarnia kod ma być wyróżniony?>
    W Pascalu przypisanie ma postać <kod>x := 5</kod>,
    natomiast w C <kod>x = 5</kod>.
    <!-- TODO: Dopisać o Javie i Ocamlu -->
  </treść>
</artykuł>
<!--recenzent: podać ciekawszy przykład -->
```

- Instrukcje przetwarzania

Zapisywanie struktury dokumentu, znaczniki

```
<?xml version="1.0" encoding="iso-8859-2"?>
<?xml-stylesheet type="text/css" href="styl.css"?>
<artykuł id="13/2007" xml:lang="pl">
  <tytuł>Przypisanie w Pascalu i C </tytuł>
  <autor email='ola@fasola.pl'>Ola Fasola</autor>
  <treść><?drukarnia kod ma być wyróżniony?>
    W Pascalu przypisanie ma postać <kod>x := 5</kod>,
    natomiast w C <kod>x = 5</kod>.
    <!-- TODO: Dopisać o Javie i Ocamlu -->
  </treść>
</artykuł>
<!--recenzent: podać ciekawszy przykład -->
```

- Deklaracja XML

Encja

- **Encja** w XML/SGML to uogólnienie pojęcia pliku (także np. wynik działania aplikacji).
- Dokument jest zapisany w co najmniej jednej encji „głównej”.
- Dokument może zawierać referencje do encji
 - zadeklarowane w DTD,
 - „rozwinęte” gdy widzimy dokument w warstwie logicznej,
 - czyli wstawione i sparsowane.
- O deklarowaniu encji w DTD w dalszej części wykładu.

Przykłady referencji do encji

```
<coś warunek="x &lt; 5">  
bla bla &moja_encja; bla bla  
</coś>
```

Encja

- **Encja** w XML/SGML to uogólnienie pojęcia pliku (także np. wynik działania aplikacji).
- Dokument jest zapisany w co najmniej jednej encji „głównej”.
- Dokument może zawierać referencje do encji
 - zadeklarowane w DTD,
 - „rozwinęte” gdy widzimy dokument w warstwie logicznej,
 - czyli wstawione i sparsowane.
- O deklarowaniu encji w DTD w dalszej części wykładu.

Przykłady referencji do encji

```
<coś warunek="x &lt; 5">  
bla bla &moja_encja; bla bla  
</coś>
```

Encje predefiniowane

- Nie trzeba ich deklarować.
- Najwygodniejszy sposób zapisywania znaków specjalnych.

Encje predefiniowane

<code>&lt;</code>	<code><</code>
<code>&gt;</code>	<code>></code>
<code>&amp;</code>	<code>&</code>
<code>&apos;</code>	<code>'</code>
<code>&quot;</code>	<code>"</code>

Referencje do znaków

- Pozwalają na umieszczenie w dokumencie dowolnego znaku Unicode.
- `ϩ` – referencja do znaku o kodzie 1001,
- `ሿ` – referencja do znaku o kodzie szesnastkowym 123F.

Sekcje CDATA

- Pozwalają na umieszczenie w dokumencie fragmentu nie przetwarzanego przez parser.
- Znacznik otwierający: `< [CDATA [`
- Znacznik zamykający: `]] >`
- `]] >` nie może wystąpić literlanie w normalnej treści dokumentu!

Przykład

```
<coś>  
Coś nie do parsowania:  
  <![CDATA[x > 0 && x < 100]]>  
</coś>
```


Sekcje CDATA

- Pozwalają na umieszczenie w dokumencie fragmentu nie przetwarzanego przez parser.
- Znacznik otwierający: `< [CDATA [`
- Znacznik zamykający: `]] >`
- `]] >` nie może wystąpić literlanie w normalnej treści dokumentu!

Przykład

```
<coś>  
Coś nie do parsowania:  
  <![CDATA[x > 0 && x < 100]]>  
</coś>
```

Unicode i Universal Character Set (UCS)

- Tabela przyporządkowująca numerom „abstrakcyjne znaki”
 - np. ogonek tworzący ą z a .
- Obecnie (Unicode 5.0, ISO/IEC 10646:2003 z dodatkami) około 100 tys. znaków, miejsce na ponad milion.
- Unicode ponadto opisuje „metadane” znaków, np.:
 - kolejność liter w alfabetach,
 - wartość liczbowa cyfry.
- Podstawowy zestaw znaków (Unicode 2.0) daje się zapisać za pomocą dwóch bajtów.

Popularne sposoby zapisywania Unicode

● UTF-16

- znaki z zestawu podstawowego zapisywane „po prostu”,
- istotna kolejność bajtów w słowie (dwie wersje),
- znaki spoza zestawu podstawowego jako para dwubajtowych słów.

● UTF-8

- znaki o kodach 0–127 zapisywane w jednym bajcie zgodnie z ASCII i ISO-8859-_,
- znaki z zakresu podstawowego zajmują do czterech bajtów,
- pozostałe znaki zapisywane do sześciu bajtów,
- optymalizacje ułatwiające znajdowanie początku znaku w środku tekstu.

Popularne sposoby zapisywania Unicode

● UTF-16

- znaki z zestawu podstawowego zapisywane „po prostu”,
- istotna kolejność bajtów w słowie (dwie wersje),
- znaki spoza zestawu podstawowego jako para dwubajtowych słów.

● UTF-8

- znaki o kodach 0–127 zapisywane w jednym bajcie zgodnie z ASCII i ISO-8859-__,
- znaki z zakresu podstawowego zajmują do czterech bajtów,
- pozostałe znaki zapisywane do sześciu bajtów,
- optymalizacje ułatwiające znajdowanie początku znaku w środku tekstu.

Inne standardy

Standardy jendobajtowe

Popularne standardy jednobajtowe

(Windows-1250/Latin-2, ISO-8859-2 i inne):

- Kody 0–127 zgodne z ASCII, Unicode i UTF-8.
- Kody 128–255 mapowane na inne znaki Unicode.

XML

- XML może być zapisany w dowolnym standardzie dającym się zmapować na Unicode (odpowiedzialność parsera).
- Każde narzędzie XML-owe musi obsługiwać UTF-8.

Inne standardy

Standardy jendobajtowe

Popularne standardy jednobajtowe

(Windows-1250/Latin-2, ISO-8859-2 i inne):

- Kody 0–127 zgodne z ASCII, Unicode i UTF-8.
- Kody 128–255 mapowane na inne znaki Unicode.

XML

- XML może być zapisany w dowolnym standardzie dającym się zmapować na Unicode (odpowiedzialność parsera).
- Każde narzędzie XML-owe musi obsługiwać UTF-8.

Plan

- 1 Podstawy XML
 - Dokument XML – warstwa logiczna
 - Dokument XML – warstwa fizyczna
 - Standardy kodowania znaków

- 2 DTD
 - Motywacja dla definiowania struktury dokumentów
 - DTD – elementy i atrybuty
 - DTD – encje i parametryzacja

- 3 Przestrzenie nazw
 - Motywacja
 - Realizacja

XML jako metajęzyk

- XML, SGML – metajęzyki:
 - języki opisane w standardach (XHTML, DocBook XML, WSDL, SVG, ...),
 - możliwość definiowania w własnych języków.
- Definiowanie języków (zastosowań, struktury dokumentów, typów dokumentów):
 - określanie zestawu dopuszczalnych elementów, atrybutów, ... ,
 - definiowanie dopuszczalnej zawartości elementów (tekst, inne elementy),
 - przypisywanie atrybutów do elementów,
 -

XML jako metajęzyk

- XML, SGML – metajęzyki:
 - języki opisane w standardach (XHTML, DocBook XML, WSDL, SVG, ...),
 - możliwość definiowania w własnych języków.
- Definiowanie języków (zastosowań, struktury dokumentów, typów dokumentów):
 - określanie zestawu dopuszczalnych elementów, atrybutów, ... ,
 - definiowanie dopuszczalnej zawartości elementów (tekst, inne elementy),
 - przypisywanie atrybutów do elementów,
 -

Dwa stopnie poprawności dokumentu

- Dokument XML **poprawny składniowo** (ang. *well-formed*):
 - poprawne nazwy,
 - każdy element musi być zamknięty,
 - nie ma nakładających się elementów (struktura drzewa),
 - wartości atrybutów w apostrofach lub cudzysłowach.
 - ...
- Dokument XML poprawny strukturalnie (ang. *valid*):
 - zgodny z definicją struktury,
 - standard XML – zgodność z DTD,
 - to samo określenie – poprawność względem innych standardów (XML Schema, Relax NG, ...).

Dwa stopnie poprawności dokumentu

- Dokument XML poprawny składniowo (ang. *well-formed*):
 - poprawne nazwy,
 - każdy element musi być zamknięty,
 - nie ma nakładających się elementów (struktura drzewa),
 - wartości atrybutów w apostrofach lub cudzysłowach.
 - ...
- Dokument XML **poprawny strukturalnie** (ang. *valid*):
 - zgodny z definicją struktury,
 - standard XML – zgodność z DTD,
 - to samo określenie – poprawność względem innych standardów (XML Schema, Relax NG, ...).

Zalety kontroli poprawności strukturalnej

(za pomocą gotowych narzędzi, zwykle podczas parsowania)

- Można przyjąć założenia dotyczące struktury dokumentu
np. występowanie danego elementu.
- Nie trzeba „ręcznie” badać struktury.
- Metody definiowania struktury:
 - DTD – Document Type Definition,
 - XML Schema (rekomendacja W3C z 2 maja 2001),
 - Relax NG,
 - Schematron,
 - (?). . . ,
 - dokument XML bez formalnej definicji struktury.

Zalety kontroli poprawności strukturalnej

(za pomocą gotowych narzędzi, zwykle podczas parsowania)

- Można przyjąć założenia dotyczące struktury dokumentu
np. występowanie danego elementu.
- Nie trzeba „ręcznie” badać struktury.
- Metody definiowania struktury:
 - DTD – Document Type Definition,
 - XML Schema (rekomendacja W3C z 2 maja 2001),
 - Relax NG,
 - Schematron,
 - (?). . . ,
 - dokument XML bez formalnej definicji struktury.

Zalety kontroli poprawności strukturalnej

(za pomocą gotowych narzędzi, zwykle podczas parsowania)

- Można przyjąć założenia dotyczące struktury dokumentu
np. występowanie danego elementu.
- Nie trzeba „ręcznie” badać struktury.
- Metody definiowania struktury:
 - DTD – Document Type Definition,
 - XML Schema (rekomendacja W3C z 2 maja 2001),
 - Relax NG,
 - Schematron,
 - (?). . . ,
 - dokument XML bez formalnej definicji struktury.

Modelowanie jako proces analityczny

Wieloetapowy proces analityczno-projektowy:

- analiza struktury modelowanych bytów,
- analiza przykładowych dokumentów,
- analiza potencjalnych zastosowań oraz przypadków użycia,
- abstrakcyjny projekt struktury,
- kodowanie projektu struktury np. przy pomocy XML Schema,
- testowanie,
- pielęgnacja i uaktualnianie.

Uwaga

Definicja zastosowania XML powinna określać nie tylko składnię (DTD, XML Schema), ale i semantykę – znaczenie poszczególnych elementów i atrybutów.

Modelowanie jako proces analityczny

Wieloetapowy proces analityczno-projektowy:

- analiza struktury modelowanych bytów,
- analiza przykładowych dokumentów,
- analiza potencjalnych zastosowań oraz przypadków użycia,
- abstrakcyjny projekt struktury,
- kodowanie projektu struktury np. przy pomocy XML Schema,
- testowanie,
- pielęgnacja i uaktualnianie.

Uwaga

Definicja zastosowania XML powinna określać nie tylko składnię (DTD, XML Schema), ale i **semantykę** – znaczenie poszczególnych elementów i atrybutów.

Położenie DTD

- **Wewnątrz** dokumentu.
- Na zewnątrz (w encji zewnętrznej)
 - identyfikator systemowy,
 - identyfikator publiczny.
- częściowo wewnątrz, częściowo na zewnątrz,
 - jako pierwsza przetwarzana część wewnętrzna.

Przykład

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE greeting [
  <!ELEMENT greeting (#PCDATA)>
]>
<greeting>Hello, world!</greeting>
```

Położenie DTD

- Wewnątrz dokumentu.
- **Na zewnątrz** (w encji zewnętrznej)
 - **identyfikator systemowy**,
 - identyfikator publiczny.
- częściowo wewnątrz, częściowo na zewnątrz,
 - jako pierwsza przetwarzana część wewnętrzna.

Przykład

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE greeting
  SYSTEM "hello.dtd">

<greeting>Hello, world!</greeting>
```

Położenie DTD

- Wewnątrz dokumentu.
- **Na zewnątrz** (w encji zewnętrznej)
 - identyfikator systemowy,
 - **identyfikator publiczny.**
- częściowo wewnątrz, częściowo na zewnątrz,
 - jako pierwsza przetwarzana część wewnętrzna.

Przykład

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE greeting
  PUBLIC "-//W3C//GREETEING 1.0//EN" "hello.dtd">

<greeting>Hello, world!</greeting>
```

Położenie DTD

- Wewnątrz dokumentu.
- Na zewnątrz (w encji zewnętrznej)
 - identyfikator systemowy,
 - identyfikator publiczny.
- **częściowo wewnątrz, częściowo na zewnątrz,**
 - jako pierwsza przetwarzana część wewnętrzna.

Przykład

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE greeting SYSTEM "hello.dtd" [
  <!ATTLIST greeting words CDATA #IMPLIED>
]>
<greeting words="2">Hello, world!</greeting>
```

Deklaracja elementu

```
<!ELEMENT nazwa-elementu typ-zawartosci-elementu >
```

Deklaracja elementu

```
<!ELEMENT nazwa-elementu typ-zawartosci-elementu >
```

Typy zawartości elementów

- **ANY** – dowolna zawartość,
- **EMPTY** – element pusty,
- **#PCDATA** – zawartość tekstowa,
- **(#PCDATA | elem1 | elem2 ...)**
– zawartość mieszana:
 - tekst z zanurzonymi podelementami,
 - brak kontroli kolejności i liczności podelementów.
- wyrażenie regularne – tylko podelementy bez tekstu.

Deklaracja elementu

```
<!ELEMENT nazwa-elementu typ-zawartosci-elementu >
```

Typy zawartości elementów

- ANY – dowolna zawartość,
- **EMPTY** – element pusty,
- #PCDATA – zawartość tekstowa,
- (#PCDATA | *elem1* | *elem2* ...)
– zawartość mieszana:
 - tekst z zanurzonymi podelementami,
 - brak kontroli kolejności i liczności podelementów.
- wyrażenie regularne – tylko podelementy bez tekstu.

Deklaracja elementu

```
<!ELEMENT nazwa-elementu typ-zawartosci-elementu >
```

Typy zawartości elementów

- ANY – dowolna zawartość,
- EMPTY – element pusty,
- #PCDATA – zawartość tekstowa,
- (#PCDATA | *elem1* | *elem2* ...)
– zawartość mieszana:
 - tekst z zanurzonymi podelementami,
 - brak kontroli kolejności i liczności podelementów.
- wyrażenie regularne – tylko podelementy bez tekstu.

Deklaracja elementu

```
<!ELEMENT nazwa-elementu typ-zawartości-elementu >
```

Typy zawartości elementów

- ANY – dowolna zawartość,
- EMPTY – element pusty,
- #PCDATA – zawartość tekstowa,
- (#PCDATA | *elem1* | *elem2* ...)
– zawartość mieszana:
 - tekst z zanurzonymi podelementami,
 - brak kontroli kolejności i liczności podelementów.
- wyrażenie regularne – tylko podelementy bez tekstu.

Deklaracja elementu

```
<!ELEMENT nazwa-elementu typ-zawartości-elementu >
```

Typy zawartości elementów

- ANY – dowolna zawartość,
- EMPTY – element pusty,
- #PCDATA – zawartość tekstowa,
- (#PCDATA | *elem1* | *elem2* ...)
– zawartość mieszana:
 - tekst z zanurzonymi podelementami,
 - brak kontroli kolejności i liczności podelementów.
- **wyrażenie regularne** – tylko podelementy bez tekstu.

Wyrażenia regularne w typie zaw. elem.

- Nazwy elementów (atomy).
- Nawiasy.
- Spójniki pomiędzy nazwami lub grupami w nawiasach:
 - , – sekwencja,
 - | – wybór.
- Modyfikatory za nazwą lub nawiasem zamykającym:
 - ? – opcjonalny,
 - * – zero lub więcej,
 - + – jeden lub więcej.

Przykład

```
<!ELEMENT osoby (osoba)* >
```

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

Wyrażenia regularne w typie zaw. elem.

- Nazwy elementów (atomy).
- Nawiasy.
- Spójniki pomiędzy nazwami lub grupami w nawiasach:
 - , – sekwencja,
 - | – wybór.
- Modyfikatory za nazwą lub nawiasem zamykającym:
 - ? – opcjonalny,
 - * – zero lub więcej,
 - + – jeden lub więcej.

Przykład

```
<!ELEMENT osoby (osoba)* >
```

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

Wyrażenia regularne w typie zaw. elem.

- Nazwy elementów (atomy).
- Nawiasy.
- Spójniki pomiędzy nazwami lub grupami w nawiasach:
 - , – sekwencja,
 - | – wybór.
- Modyfikatory za nazwą lub nawiasem zamykającym:
 - ? – opcjonalny,
 - * – zero lub więcej,
 - + – jeden lub więcej.

Przykład

```
<!ELEMENT osoby (osoba)* >
```

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

Wyrażenia regularne w typie zaw. elem.

- Nazwy elementów (atomy).
- Nawiasy.
- Spójniki pomiędzy nazwami lub grupami w nawiasach:
 - , – sekwencja,
 - | – wybór.
- Modyfikatory za nazwą lub nawiasem zamykającym:
 - ? – opcjonalny,
 - * – zero lub więcej,
 - + – jeden lub więcej.

Przykład

```
<!ELEMENT osoby (osoba)* >
```

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

Deklaracja listy atrybutów

```
<!ATTLIST nazwa-elementu  
  nazwa-atrybutu typ-zawartości-atrybutu opis-domyślnej  
  nazwa-atrybutu typ-zawartości-atrybutu opis-domyślnej  
  ...>
```

Może być wiele list atrybutów dla danego elementu, są akumulowane, ważniejsza jest wcześniejsza.

Deklaracja listy atrybutów

```
<!ATTLIST nazwa-elementu  
  nazwa-atrybutu typ-zawartości-atrybutu opis-domyślnej  
  nazwa-atrybutu typ-zawartości-atrybutu opis-domyślnej  
  ...>
```

Może być wiele list atrybutów dla danego elementu, są akumulowane, ważniejsza jest wcześniejsza.

Opis domyślnej

- **#REQUIRED** – atrybut obowiązkowy,
- **#IMPLIED** – atrybut opcjonalny,
- **"wartość"** – wartość domyślna,
- **#FIXED "wartość"** – wartość ustalona.

Deklaracja listy atrybutów

```
<!ATTLIST nazwa-elementu
  nazwa-atrybutu typ-zawartości-atrybutu opis-domyślnej
  nazwa-atrybutu typ-zawartości-atrybutu opis-domyślnej
  ...>
```

Może być wiele list atrybutów dla danego elementu, są akumulowane, ważniejsza jest wcześniejsza.

Opis domyślnej

- #REQUIRED – atrybut obowiązkowy,
- #IMPLIED – atrybut opcjonalny,
- "wartość" – wartość domyślna,
- #FIXED "wartość" – wartość ustalona.

Deklaracja listy atrybutów

```
<!ATTLIST nazwa-elementu
  nazwa-atrybutu typ-zawartości-atrybutu opis-domyślnej
  nazwa-atrybutu typ-zawartości-atrybutu opis-domyślnej
  ...>
```

Może być wiele list atrybutów dla danego elementu, są akumulowane, ważniejsza jest wcześniejsza.

Opis domyślnej

- #REQUIRED – atrybut obowiązkowy,
- #IMPLIED – atrybut opcjonalny,
- "**wartość**" – wartość domyślna,
- #FIXED "**wartość**" – wartość ustalona.

Deklaracja listy atrybutów

```
<!ATTLIST nazwa-elementu  
  nazwa-atrybutu typ-zawartości-atrybutu opis-domyślnej  
  nazwa-atrybutu typ-zawartości-atrybutu opis-domyślnej  
  ...>
```

Może być wiele list atrybutów dla danego elementu, są akumulowane, ważniejsza jest wcześniejsza.

Opis domyślnej

- #REQUIRED – atrybut obowiązkowy,
- #IMPLIED – atrybut opcjonalny,
- "wartość" – wartość domyślna,
- #FIXED "wartość" – wartość ustalona.

Typy zawartości atrybutów

- **CDATA** – dowolny tekst,
- **NMTOKEN** – „token” (tylko litery, cyfry i znaki _ - :),
- **NMTOKENS** – tokeny rozdzielone białymi znakami,
- **ID** – identyfikator unikalny w skali dokumentu,
 - musi być nazwą,
 - dla danego typu elementu max jeden atrybut typu ID.
- **IDREF** i **IDREFS**
 - wartość równa wartości pewnego atrybutu typu ID w tym samym dokumencie.

Przykład

```
<!ATTLIST osoba płeć (K | M) #REQUIRED  
email CDATA #IMPLIED>
```

Typy zawartości atrybutów

- CDATA – dowolny tekst,
- **NMTOKEN** – „token” (tylko litery, cyfry i znaki _ - :),
- NMTOKENS – tokeny rozdzielone białymi znakami,
- ID – identyfikator unikalny w skali dokumentu,
 - musi być nazwą,
 - dla danego typu elementu max jeden atrybut typu ID.
- IDREF i IDREFS
 - wartość równa wartości pewnego atrybutu typu ID w tym samym dokumencie.

Przykład

```
<!ATTLIST osoba płeć (K | M) #REQUIRED  
email CDATA #IMPLIED>
```

Typy zawartości atrybutów

- CDATA – dowolny tekst,
- NMTOKEN – „token” (tylko litery, cyfry i znaki _ - :),
- **NMTOKENS** – tokeny rozdzielone białymi znakami,
- ID – identyfikator unikalny w skali dokumentu,
 - musi być nazwą,
 - dla danego typu elementu max jeden atrybut typu ID.
- IDREF i IDREFS
 - wartość równa wartości pewnego atrybutu typu ID w tym samym dokumencie.

Przykład

```
<!ATTLIST osoba płeć (K | M) #REQUIRED  
email CDATA #IMPLIED>
```

Typy zawartości atrybutów

- CDATA – dowolny tekst,
- NMTOKEN – „token” (tylko litery, cyfry i znaki _ - :),
- NMTOKENS – tokeny rozdzielone białymi znakami,
- ID – identyfikator unikalny w skali dokumentu,
 - musi być nazwą,
 - dla danego typu elementu max jeden atrybut typu ID.
- IDREF i IDREFS
 - wartość równa wartości pewnego atrybutu typu ID w tym samym dokumencie.

Przykład

```
<!ATTLIST osoba płeć (K | M) #REQUIRED  
email CDATA #IMPLIED>
```

Typy zawartości atrybutów

- CDATA – dowolny tekst,
- NMTOKEN – „token” (tylko litery, cyfry i znaki _ - :),
- NMTOKENS – tokeny rozdzielone białymi znakami,
- ID – identyfikator unikalny w skali dokumentu,
 - musi być nazwą,
 - dla danego typu elementu max jeden atrybut typu ID.
- IDREF i IDREFS
 - wartość równa wartości pewnego atrybutu typu ID w tym samym dokumencie.

Przykład

```
<!ATTLIST osoba płeć (K | M) #REQUIRED  
                email CDATA #IMPLIED>
```


Encja ogólna: wewnętrzna i zewnętrzna

- **Encja ogólna** – referencje mogą występować w węzłach tekstowych i atrybutach
 - w atrybutach tylko encje zawierające tekst.
- **Encja wewnętrzna** – treść podana bezpośrednio w DTD.
- **Encja zewnętrzna** – treść znajduje się w zasobie zewnętrznym (pliku itp.).

Przykład

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE coś [
  <!ELEMENT coś (#PCDATA)>
  <!ENTITY wewnętrzna "Wartość" >
  <!ENTITY zewnętrzna SYSTEM "plik.xml" >
]>
<coś>
&wewnętrzna;
&zewnętrzna;
</coś>
```

Encja ogólna: wewnętrzna i zewnętrzna

- **Encja ogólna** – referencje mogą występować w węzłach tekstowych i atrybutach
 - w atrybutach tylko encje zawierające tekst.
- **Encja wewnętrzna** – treść podana bezpośrednio w DTD.
- **Encja zewnętrzna** – treść znajduje się w zasobie zewnętrznym (pliku itp.).

Przykład

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE coś [
  <!ELEMENT coś (#PCDATA)>
  <!ENTITY wewnętrzna "Wartość" >
  <!ENTITY zewnętrzna SYSTEM "plik.xml" >
]>
<coś>
&wewnętrzna;
&zewnętrzna;
</coś>
```

Encja ogólna: wewnętrzna i zewnętrzna

- **Encja ogólna** – referencje mogą występować w węzłach tekstowych i atrybutach
 - w atrybutach tylko encje zawierające tekst.
- **Encja wewnętrzna** – treść podana bezpośrednio w DTD.
- **Encja zewnętrzna** – treść znajduje się w zasobie zewnętrznym (pliku itp.).

Przykład

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE coś [
  <!ELEMENT coś (#PCDATA)>
  <!ENTITY wewnętrzna "Wartość" >
  <!ENTITY zewnętrzna SYSTEM "plik.xml" >
]>
<coś>
&wewnętrzna;
&zewnętrzna;
</coś>
```

Encje parametryczne

- **Encja parametryczna** – referencje mogą występować wewnątrz DTD.
- Brak konfliktu nazw z encjami ogólnymi.
- Wewnętrzne lub zewnętrzne.

Przykład

```
<!ENTITY % TOsoba "(imię+, nazwisko)">
```

```
<!ELEMENT student %TOsoba;>
```

```
<!ELEMENT pracownik %TOsoba;>
```

Encje nieprzetwarzane

- Pozwalają na dołączanie do dokumentów danych binarnych.
- Referencje tylko w specjalnie zadeklarowanych atrybutach.
- Wymagają zadeklarowania **notacji**.
- Pozostałość SGML, praktycznie nieużywane w nowoczesnych zastosowaniach XML.

Rodzaje encji – podsumowanie

	wewnętrzne		zewnętrzne	
	przetw.	nieprzetw.	przetw.	nieprzetw.
ogólne	TAK	NIE	TAK	TAK
parametryczne	TAK	NIE	TAK	NIE

Sekcje warunkowe

```
<![INCLUDE [ ... ]]>
```

```
<![IGNORE [ ... ]]>
```

- Powodują przetworzenie lub nie danego fragmentu DTD.
- W połączeniu z encjami parametrycznymi pozwalają tworzyć sparametryzowane DTD.

Sekcje warunkowe – zastosowania

Zewnętrzna część DTD

```
<!ENTITY % usos "IGNORE">
...
<![%usos;[
<!ATTLIST przedmiot usos-id CDATA #REQUIRED>
]]>
```

Dokument

```
<!DOCTYPE przedmioty SYSTEM "przedmioty.dtd" [
  <!ENTITY % usos "INCLUDE">
]>
...
<przedmiot usos-id="1000-2M01XM">XML</przedmiot>
...
```


Sekcje warunkowe – zastosowania

Zewnętrzna część DTD

```
<!ENTITY % usos "IGNORE">
...
<![%usos;[
<!ATTLIST przedmiot usos-id CDATA #REQUIRED>
]]>
```

Dokument

```
<!DOCTYPE przedmioty SYSTEM "przedmioty.dtd" [
  <!ENTITY % usos "INCLUDE">
]>
...
<przedmiot usos-id="1000-2M01XM">XML</przedmiot>
...
```

Plan

- 1 Podstawy XML
 - Dokument XML – warstwa logiczna
 - Dokument XML – warstwa fizyczna
 - Standardy kodowania znaków

- 2 DTD
 - Motywacja dla definiowania struktury dokumentów
 - DTD – elementy i atrybuty
 - DTD – encje i parametryzacja

- 3 Przestrzenie nazw
 - Motywacja
 - Realizacja

Konflikt nazw

- Problem: ta sama nazwa oznacza dwa różne byty w różnych dokumentach,
dokumenty te są powiązane (np. wspólnie przetwarzane, jeden zanurzony w drugim, itp.)
- Rozwiązanie: przestrzeń nazw (*namespace*) – grupa nazw oddzielona (składniowo i semantycznie) od innych nazw.

Jednoznaczny identyfikator

W dokumencie:

- deklaracje przestrzeni nazw – przypisanie prefiksom identyfikatorów przestrzeni nazw,
- nazwy składają się z prefiksu i części lokalnej.
- Zamiast nazwy **para**:
 - unikalny globalnie identyfikator przestrzeni nazw,
 - nazwa lokalna.
- W wersji 1.0 za id. prz. nazw przyjęto **URI**:
 - *Uniform Resource Identifier*,
 - `http://www.w3.org/1999/xhtml` (przykładowy URL),
 - `urn:isbn:0-395-36341-1` (przykładowy URN).
- W wersji 1.1 standardu zmiana na **IRI**
 - *Internationalized Resource Identifier*,
 - uogólnienie URI dopuszczające znaki spoza ASCII.

Jednoznaczny identyfikator

W dokumencie:

- deklaracje przestrzeni nazw – przypisanie prefiksom identyfikatorów przestrzeni nazw,
- **nazwy składają się z prefiksu i części lokalnej.**
- Zamiast nazwy **para**:
 - unikalny globalnie identyfikator przestrzeni nazw,
 - nazwa lokalna.
- W wersji 1.0 za id. prz. nazw przyjęto **URI**:
 - *Uniform Resource Identifier*,
 - `http://www.w3.org/1999/xhtml` (przykładowy URL),
 - `urn:isbn:0-395-36341-1` (przykładowy URN).
- W wersji 1.1 standardu zmiana na **IRI**
 - *Internationalized Resource Identifier*,
 - uogólnienie URI dopuszczające znaki spoza ASCII.

Prefiksy i przestrzenie nazw

```
< os:osoba
  xmlns:os="http://szcz.mimuw.edu.pl/osoby"
  xmlns:inst="http://szcz.mimuw.edu.pl/institucje"
  xmlns:xhtml="http://www.w3.org/1999/xhtml">
  < os:imie>Jan</ os:imie>
  < os:nazwisko>Kowalski</ os:nazwisko>
  < os:NIP>123-456-78-90</ os:NIP>
  < os:opis>To jest < xhtml:b>bardzo</ xhtml:b> fajny
  facet!</ os:opis>
  < os:pracuje-w>
    < inst:firma>
      < inst:nazwa>Business Consulting</ inst:nazwa>
      < inst:NIP>987-654-32-10</ inst:NIP>
    </ inst:firma>
  </ os:pracuje-w>
</ os:osoba>
```

Domyślna przestrzeń nazw

```
< osoba xmlns="http://szcz.mimuw.edu.pl/osoby"
  xmlns:inst="http://szcz.mimuw.edu.pl/instytucje"
  xmlns:xhtml="http://www.w3.org/1999/xhtml">
  < imie>Jan</ imie>
  < nazwisko>Kowalski</ nazwisko>
  < NIP>123-456-78-90</ NIP>
  < opis>To jest < xhtml:b>bardzo</ xhtml:b> fajny
  facet!</ opis>
  < pracuje-w>
  < inst:firma>
    < inst:nazwa>Business Consulting</ inst:nazwa>
    < inst:NIP>987-654-32-10</ inst:NIP>
  </ inst:firma>
</ pracuje-w>
</ osoba>
```

Przeddefiniowanie prefiksu

```
< pre:osoba xmlns:pre="http://szcz.mimuw.edu.pl/osoby"
  xmlns:inst="http://szcz.mimuw.edu.pl/institucje"
  xmlns:xhtml="http://www.w3.org/1999/xhtml">
  < pre:imie>Jan</ pre:imie>
  < pre:nazwisko>Kowalski</ pre:nazwisko>
  < pre:NIP>123-456-78-90</ pre:NIP>
  < pre:opis>To jest < xhtml:b>bardzo</ xhtml:b> fajny
  facet!</ pre:opis>
  < pre:pracuje-w>
    < pre:firma xmlns:pre="http://szcz.mimuw.edu.pl/institucje">
      < pre:nazwa>Business Consulting</ pre:nazwa>
      < pre:NIP>987-654-32-10</ pre:NIP>
    </ pre:firma>
  </ pre:pracuje-w>
</ pre:osoba>
```


Domyślna przestrzeń nazw a atrybuty

- Domyślna przestrzeń nazw nie dotyczy atrybutów.
- Atrybuty bez prefiksu nie należą do żadnej przestrzeni nazw.
- **Znaczenie** atrybutu bez prefiksu zależy od tego w jakim znajduje się elemencie.

Przestrzenie nazw a nazwy atrybutów

Poprawne

```
<x xmlns:n1="http://szcz.mimuw.edu.pl/n1"
  xmlns:n2="http://szcz.mimuw.edu.pl/n2">
  <good n1:a="1" n2:a="2"/>
</x>
```

Niepoprawne

```
<x xmlns:n1="http://szcz.mimuw.edu.pl/n1"
  xmlns:n2="http://szcz.mimuw.edu.pl/n1">
  <bad n1:a="1" n2:a="2"/>
</x>
```

Poprawne

```
<x xmlns:n1="http://szcz.mimuw.edu.pl/n1"
  xmlns="http://szcz.mimuw.edu.pl/n1">
  <good n1:a="1" a="2"/>
</x>
```

Przestrzenie nazw a nazwy atrybutów

Poprawne

```
<x xmlns:n1="http://szcz.mimuw.edu.pl/n1"
  xmlns:n2="http://szcz.mimuw.edu.pl/n2">
  <good n1:a="1" n2:a="2"/>
</x>
```

Niepoprawne

```
<x xmlns:n1="http://szcz.mimuw.edu.pl/n1"
  xmlns:n2="http://szcz.mimuw.edu.pl/n1">
  <bad n1:a="1" n2:a="2"/>
</x>
```

Poprawne

```
<x xmlns:n1="http://szcz.mimuw.edu.pl/n1"
  xmlns="http://szcz.mimuw.edu.pl/n1">
  <good n1:a="1" a="2"/>
</x>
```

Przestrzenie nazw a nazwy atrybutów

Poprawne

```
<x xmlns:n1="http://szcz.mimuw.edu.pl/n1"
  xmlns:n2="http://szcz.mimuw.edu.pl/n2">
  <good n1:a="1" n2:a="2"/>
</x>
```

Niepoprawne

```
<x xmlns:n1="http://szcz.mimuw.edu.pl/n1"
  xmlns:n2="http://szcz.mimuw.edu.pl/n1">
  <bad n1:a="1" n2:a="2"/>
</x>
```

Poprawne

```
<x xmlns:n1="http://szcz.mimuw.edu.pl/n1"
  xmlns="http://szcz.mimuw.edu.pl/n1">
  <good n1:a="1" a="2"/>
</x>
```

Przestrzenie nazw a standard XML

- Sam standard XML (i DTD) nie wspiera przestrzeni nazw.
- Nowe standardy, jak XML Schema, XSLT czy XLink, wspierają przestrzenie nazw.
- Namespaces in XML 1.0 – rekomendacja W3C z 1999 r obecnie dotyczy dokumentów XML 1.0.
- Namespaces in XML 1.1 – rekomendacja W3C z 2006 r dotyczy dokumentów XML 1.1, możliwość oddeklarowania prefiksu, IRI zamiast URI.
- Na dokument XML można patrzeć na dwa sposoby:
 - nie uwzględniając przestrzeni nazw,
 - uwzględniając przestrzenie nazw (mniej dokumentów poprawnych).
- Parsery zwykle dają wybór trybu parsowania.

Przestrzenie nazw a standard XML

- Sam standard XML (i DTD) nie wspiera przestrzeni nazw.
- Nowe standardy, jak XML Schema, XSLT czy XLink, wspierają przestrzenie nazw.
- Namespaces in XML 1.0 – rekomendacja W3C z 1999 r obecnie dotyczy dokumentów XML 1.0.
- Namespaces in XML 1.1 – rekomendacja W3C z 2006 r dotyczy dokumentów XML 1.1, możliwość oddeklarowania prefiksu, IRI zamiast URI.
- Na dokument XML można patrzeć na dwa sposoby:
 - nie uwzględniając przestrzeni nazw,
 - uwzględniając przestrzenie nazw (mniej dokumentów poprawnych).
- Parsery zwykle dają wybór trybu parsowania.