Programowanie komunikacji człowiek-komputer

dr inż. Joanna Ochelska-Mierzejewska

2

DTD

Definicja typu dokumentu (ang. *Document Type Definition*)

3

Po co nam formalizacja struktury?

- Dostępność logicznej struktury treści pomaga lepiej przetwarzać dokumenty
- Zamiast pilnować poprawności danych „samemu" (w naszej aplikacji) możemy zlecić sprawdzanie poprawności automatowi, który wymusi poprawność budowy dokumentu
- Przeniesienie zadania sprawdzania poprawności na parser daje spore oszczędności (ponoć aż 60% kodu programów dotyczy weryfikacji poprawności danych)

4

Struktura dokumentu – idea

- Mamy stworzyć dokument opisujący osobę, który ma posiadać element główny `<osoba>`, w którym znajdują się elementy `<imię>`, `<nazwisko>` i `<adres>` w tej właśnie kolejności, z tym że dana osoba może mieć więcej niż jedno imię, natomiast nie musi mieć zdefiniowanego adresu

```
<osoba>
  <imię> Jan </imię>
  <nazwisko> Kowalski </nazwisko>
  <adres> Wólczańska 215 </adres>
</osoba>
```

```
<osoba>
  <imię> Jan </imię>
  <nazwisko> Kowalski </nazwisko>
</osoba>
```

```
<osoba>
  <imię> Jan </imię>
  <imię> Adam </imię>
  <nazwisko> Kowalski </nazwisko>
  <adres> Wólczańska 215 </adres>
</osoba>
```

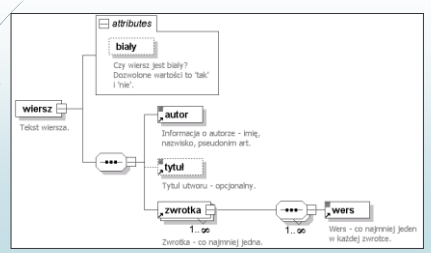
5

Modelowanie struktury logicznej

- Modelowanie struktury dokumentów to
  - Określenie zestawu dopuszczalnych elementów i atrybutów
  - Przypisanie atrybutów do elementów
  - Definiowanie dopuszczalnej zawartości elementów (tekst, inne elementy)

6

Projektowanie struktury abstrakcyjnej



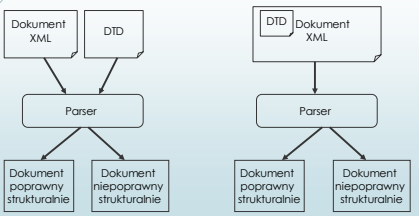
7

Formalizmy opisu struktury

- Metody opisu struktury
  - Brak struktury
  - DTD – Definicja Typu Dokumentu (ang. *Document Type Definition*)
  - XML Schema (więcej możliwości)
  - Inne (Relax NG, Schematron, ...)

8

Poprawność dokumentu a DTD



9

Poprawność dokumentu

- Dokument XML-owy jest
  - Poprawny składniowo (ang. *well-formed*), gdy
    - Ma tylko jeden element główny
    - Znaczniki początkowe i końcowe „pasują do siebie" (każdy element zamknięty, znaczniki nie nakładają się na siebie)
    - Wartości atrybutów są ujęte w cudzysłowy lub apostrofy
    - ... inne wymagania ujęte w specyfikacji
  - Poprawny strukturalnie (ang. *valid*), gdy
    - Jest poprawny składniowo
    - Jest zgodny z ustaloną wcześniej strukturą logiczną
    - Zostały określone wszystkie wymagane atrybuty
    - ... inne wymagania

10

Definicja elementu

```
<!ELEMENT nazwaElementu typElementu>
```

- Definicja składa się z trzech części
  - Słowa kluczowego `ELEMENT`
  - Nazwy danego elementu, czyli nazwy znacznika
  - Słowa lub grupy słów definiujących jego zawartość (`typElementu`)
- Element może
  - Być pusty
  - Zawierać tekst (treść)
  - Zawierać nieokreśloną postać (niewskazany w finalnej postaci DTD, pomocniczy przy wymyślaniu definicji elementów)
  - Zawierać jeden lub więcej elementów-dzieci (podelementy)
  - Zawierać tekst i podelementy (mieszany)
- Każdy element musi być zadeklarowany

11

Definicja elementu pustego

```
<!ELEMENT nazwaElementu typElementu>
```

- `typElementu = EMPTY` deklaruje elementy puste
- Nie używa się nawiasów, ani innych deklaracji elementów
- Element pusty może posiadać atrybuty

```
DTD: <!ELEMENT sesja EMPTY>
```

```
XML: <sesja />
```

12

Definicja elementu tekstowego (dowolną)

```
<!ELEMENT nazwaElementu typElementu>
```

- `typElementu = (#PCDATA)` opisuje podstawowe elementy, które zawierają analizowane dane znakowe
- Dane te są wierszem tekstu i mogą zawierać odniesienia jednostek, takich jak `>` lub `<`
- Nie mogą zawierać innych znaczników lub elementów potomnych

```
DTD: <!ELEMENT klient (#PCDATA)>
```

```
XML: <klient>Jan Kowalski</klient>
```

13

Definicja elementu nieokreślonego

```
<!ELEMENT nazwaElementu typElementu>
```

- `typElementu = ANY` deklaruje element, który może posiadać dowolny typ zawartości
- Nie używa się nawiasów, ani innych deklaracji elementów

```
DTD: <!ELEMENT mój ANY>
```

```
XML: <mój/>
```

```
XML: <mój>Jan Kowalski</mój>
```

```
XML: <mój>Jan Kowalski<twój>Jan Nowak</twój></mój>
```

14

Definicja elementu z podelementami

```
<!ELEMENT nazwaElementu typElementu>
```

- `typElementu = (grupaElementów)`
- `grupaElementów` zawiera przynajmniej jeden symbol (token) opisujący podelement danego elementu
- W przypadku, gdy model zawiera więcej niż jeden symbol, elementy dzieci mogą być organizowane na dwa sposoby: sekwencyjny i wyboru

```
DTD:
<!ELEMENT zamówienie (klient)>
<!ELEMENT klient (#PCDATA)>
```

```
XML:
<zamówienie><klient>Jan Kowalski</klient></zamówienie>
```

15

Definicja elementu z podelementami

```
DTD:
<!ELEMENT zamówienie (nrZamówienia, klient)>
<!ELEMENT nrZamówienia (#PCDATA)>
<!ELEMENT klient (#PCDATA)>
```

```
XML:
<zamówienie><nrZamówienia>123</nrZamówienia>
            <klient> Jan Kowalski </klient>
</zamówienie>
```

```
DTD:
<!ELEMENT zamówienie (nrZam | nrKlient | NIP)>
<!ELEMENT nrZam (#PCDATA)>
<!ELEMENT nrKlient (#PCDATA)>
<!ELEMENT NIP (#PCDATA)>
```

```
XML:
<zamówienie>
   <nrZam> 123 </nrZam>
</zamówienie>
```

```
XML:
<zamówienie>
   <nrKlient> 456 </nrKlient>
</zamówienie>
```

```
XML:
<zamówienie>
   <NIP> 123-234-45-56 </NIP>
</zamówienie>
```

16

Definicja elementu mieszanego

```
<!ELEMENT nazwaElementu typElementu>
```

- Składa się z treści tekstowej przeplatanej podelementami
- `typElementu = (#PCDATA | element | element2)*`

```
DTD:
<!ELEMENT produkt (#PCDATA | pozycja)*>
<!ELEMENT pozycja (#PCDATA)>
```

```
XML: <produkt>bułki</produkt>
```

```
XML: <produkt><pozycja>12</pozycja></produkt>
```

```
XML: <produkt><pozycja>12</pozycja>bułki</produkt>
```

```
XML: <produkt>bułki<pozycja>12</pozycja>
          masło</produkt>
```

17

Określenie liczby wystąpień elementów

		Element występuje dokładnie raz
?	Opcjonalny	Element występuje raz lub w ogóle nie występuje
+	Wymagany i powtarzalny	Element występuje raz lub wielokrotnie
*	Opcjonalny i powtarzalny	Element nie występuje lub występuje wielokrotnie

18

Przykłady modeli elementów

```
DTD: <!ELEMENT daneKontaktowe (id, (nazwa | telefon | email)) >
```

```
XML:
<daneKontaktowe>
   <id>13</id>
   <nazwa>Jan Kowalski</nazwa>
</daneKontaktowe>
```

```
XML:
<daneKontaktowe>
   <id>13</id>
   <telefon>123456?</telefon>
</daneKontaktowe>
```

```
XML:
<daneKontaktowe>
   <id>13</id>
   <email>jk@wp.pl</email>
</daneKontaktowe>
```

```
DTD: <!ELEMENT daneKontaktowe (id, nazwa?) >
```

```
XML:
<daneKontaktowe>
   <id>13</id>
   <nazwa>Jan Kowalski</nazwa>
</daneKontaktowe>
```

```
XML:
<daneKontaktowe>
   <id>13</id>
</daneKontaktowe>
```

```
DTD: <!ELEMENT daneKontaktowe (id, nazwa*) >
```

```
XML:
<daneKontaktowe>
   <id>13</id>
   <nazwa>Jan Kowalski</nazwa>
</daneKontaktowe>
```

```
XML:
<daneKontaktowe>
   <id>13</id>
   <nazwa>Jan Kowalski</nazwa>
   <nazwa>Jan Nowak</nazwa>
   <nazwa>Jan X</nazwa>
</daneKontaktowe>
```

```
XML:
<daneKontaktowe>
   <id>13</id>
</daneKontaktowe>
```

19

Przykłady modeli elementów

```
DTD: <!ELEMENT daneKontaktowe
          ((id,nazwa) | (nazwa,telefon,email)) >
```

```
XML:
<daneKontaktowe>
   <id>13</id>
   <nazwa>Jan Kowalski</nazwa>
</daneKontaktowe>
```

```
XML:
<daneKontaktowe>
   <nazwa>Jan Kowalski</nazwa>
   <telefon>12345678</telefon>
   <email>jan.kowalski@wp.pl</email>
</daneKontaktowe>
```

```
DTD: <!ELEMENT daneKontaktowe (id, (nazwa | tel | email)? ) >
```

```
XML:
<daneKontaktowe>
   <id>13</id>
</daneKontaktowe>
```

```
XML:
<daneKontaktowe>
   <id>13</id>
   <nazwa>Jan</nazwa>
</daneKontaktowe>
```

```
XML:
<daneKontaktowe>
   <id>13</id>
   <tel>12345</tel>
</daneKontaktowe>
```

```
XML:
<daneKontaktowe>
   <id>13</id>
   <email>j@wp.pl</email>
</daneKontaktowe>
```

```
DTD:
<!ELEMENT biblioteka (książka*) >
<!ELEMENT książka (autor+, tytuł, podtytuł?)>
<!ELEMENT autor (#PCDATA)>
<!ELEMENT tytuł (#PCDATA)>
<!ELEMENT podtytuł (#PCDATA)>
```

20

Definicja atrybutu

```
<!ATTLIST nazwaElementu nazwaAtrybutu typ zawartość>
```

Typ	Opis
CDATA	Zawiera dane znakowe
ID	Zawiera nazwę XML, która jest niepowtarzającym się identyfikatorem W deklaracji atrybutu dla danego elementu może wystąpi tylko jeden atrybut o typie ID
IDREF	Zawiera odniesienie do niepowtarzającego się identyfikatora Wartość tego atrybutu musi odpowiadać wartości ID w innym elemencie danego dokumentu
IDREFS	Zawiera listę odniesień do niepowtarzających się identyfikatorów (oddzielonych spacjami)
NMTOKEN	Słowo, symbol – nazwa może składa się z liter, cyfr, znaku podkreślenia, myślnika, kropki, dwukropka i nie może zawiera białych znaków
NMTOKENS	Zbiór słów, symboli oddzielonych od siebie znakami spacji

21

Definicja atrybutu

```
<!ATTLIST nazwaElementu nazwaAtrybutu typ zawartość>
```

Typ	Opis
ENTITY	Zawiera nazwę nieanalizowanej jednostki (plik, obraz, tekst) – jest rodzajem zmiennej, pod którą kryje się dalsza informacja
ENTITIES	Zawiera nazwy wielu nieanalizowanych jednostek
NOTATION	Umożliwienie wstawienia instrukcji niezgodnych z językiem XML, a podlegających restrykcjom programu zewnętrznego
wyliczeniowy	Zawiera listę możliwych wartości atrybutu – lista jest umieszczana w nawiasach okrągłych, a elementy rozdzielone są znakiem „|"

22

Definicja atrybutu

```
<!ATTLIST nazwaElementu nazwaAtrybutu typ zawartość>
```

Kwalifikatory zawartości	Znaczenie
#REQUIRED	Atrybut jest obowiązkowy, bez określenia wartości
#IMPLIED	Atrybut jest opcjonalny
"wartość"	Wartością domyślną atrybutu jest "wartość"
#FIXED "wartość"	Wartość stała, zawsze taka sama

23

Przykłady modeli atrybutów

```
DTD:
<!ELEMENT spisKsiazek (tytuł, autor)>
<!ELEMENT tytuł (#PCDATA)>
<!ATTLIST tytuł pozycja NMTOKENS #IMPLIED>
<!ELEMENT autor (#PCDATA)>
<!ATTLIST autor imię NMTOKEN #REQUIRED>
```

```
XML:
<spisKsiazek>
  <tytuł pozycja="półka 5 po lewej stronie od biurka">Mitologia</tytuł>
  <autor imię="Jan">Parandowski</autor>
</spisKsiazek>
```

```
XML:
<spisKsiazek>
  <tytuł>Mitologia</tytuł>
  <autor imię="Jan">Parandowski</autor>
</spisKsiazek>
```

24

Przykłady modeli atrybutów

```
DTD:
<!ELEMENT spisKsiazek (tytuł*)>
<!ELEMENT tytuł (#PCDATA)>
<!ATTLIST tytul gatunek
      (romans | kryminał | horror | klasyka) "klasyka">
```

```
XML:
<spisKsiazek>
  <tytuł gatunek="klasyka"> Mitologia </tytuł>
  <tytuł gatunek="romans"> Ptaki ciernistych krzewów </tytuł>
  <tytuł> Zemsta </tytuł>
</spisKsiazek>
```

```
<spisKsiazek>
  <tytul gatunek="klasyka">Mitologia</tytul>
  <tytul gatunek="romans">Ptaki ciernistych krzewow</tytul>
  <tytul gatunek="klasyka">Zemsta</tytul>
</spisKsiazek>
```

25

Przykłady modeli atrybutów

```
DTD:
<!ELEMENT biblioteka (autorzy, książki, czytelnicy,
                      raport)>

<!ELEMENT autorzy (autor*)>
<!ELEMENT autor (#PCDATA)>
<!ATTLIST autor idAutora ID #REQUIRED>

<!ELEMENT książki (książka+)>
<!ELEMENT książka (tytuł+)>
<!ATTLIST książka idKsiazka ID #REQUIRED
                  idAutora IDREF #IMPLIED>

<!ELEMENT czytelnicy (czytelnik+)>
<!ELEMENT czytelnik (#PCDATA)>
<!ATTLIST czytelnik idOsoby ID #REQUIRED>

<!ELEMENT raport (#PCDATA)>
<!ATTLIST raport idKsiazkaIdAutorIdOsoba IDREFS #REQUIRED>
```

```
XML:
<biblioteka>
  <autorzy>
    <autor idAutora="A4">J. Parandowski</autor>
  </autorzy>
  <książki>
    <książka idKsiazka="K1" idAutora="A4">
      <tytuł>Mitologia</tytuł>
    </książka>
  </książki>
  <czytelnicy>
    <czytelnik idOsoby="O6">Jan Kowalski</czytelnik>
  </czytelnicy>
  <raport idKsiazkaIdAutorIdOsoba="K1 A4 O6"/>
</biblioteka>
```

26

Przykłady modeli atrybutów

```
DTD:
<!ELEMENT spisKsiazek (ksiazka*)>
<!ELEMENT ksiazka EMPTY>
<!ATTLIST ksiazka
    tytul     CDATA  #REQUIRED
    autor     CDATA  #REQUIRED
    liczbaStron    NMTOKEN       #IMPLIED
    oprawa    NMTOKEN       #FIXED "twarda"
    gatunek (romans | kryminal | horror | klasyka|inny) "inny"
    wlasciciel     CDATA  "Biblioteka PL" >
```

```
XML:
<spisKsiazek>
  <ksiazka tytul="Mitologia" autor="Jan Parandowski" />
  <ksiazka tytul="Tozsamosc Bourne'a" autor="Robert Ludlum" liczbaStron="450" />
  <ksiazka tytul="Raport Pelikana" autor="John Grisham" liczbaStron="300"
           oprawa="twarda" gatunek="kryminal" wlasciciel="JOH" />
</spisKsiazek>
```

27

Przykłady modeli atrybutów

- Zawiera nazwę nieanalizowanej jednostki (plik, obraz, tekst) – jest rodzajem zmiennej, pod którą kryje się dalsza informacja

```
DTD:
<!ATTLIST film plik ENTITY #REQUIRED>
```

```
XML:
<film plik="X-Man-trailer.avi"/>
```

```
DTD:
<!ATTLIST pokaz slajdy ENTITIES #REQUIRED>
```

```
XML:
<pokaz slajdy="slajd1 slajd2 slajd4 slajd7"/>
```

28

Przykłady modeli atrybutów

- Umożliwienie wstawienia instrukcji niezgodnych z językiem XML, a podlegających restrykcjom programu zewnętrznego

```
DTD:
<!NOTATION gif SYSTEM "image/gif">
<!NOTATION tiff SYSTEM "image/tiff">
<!NOTATION jpeg SYSTEM "image/jpeg">
<!NOTATION png SYSTEM "image/png">
<!ATTLIST image type NOTATION (gif | tiff | jpeg | png) #REQUIRED>
```

29

Dwie twarze XML-a

- Dwa zastosowania XML-a
  - Zarządzanie treścią (dokumenty tworzone przez człowieka i przeznaczone dla człowieka, o długim czasie życia), np. Wielka Encyklopedia Powszechna PWN
  - Elektroniczna wymiana danych – komunikacja między aplikacjami (dokumenty tworzone oraz przetwarzane automatycznie, zazwyczaj kończące życie wraz z końcem komunikacji), np. komunikaty o błędach
- Tym dwu zastosowaniom odpowiadają dwa modele zawartości XML-owej: „tekstowy" i „bazodanowy"

30

XML „tekstowy" a XML „bazodanowy"

```
<hasło> <osoba> William Szekspir </osoba> (ang. <wariant>
William Shakespeare </wariant>; ur. prawdopodobnie
<data-ur> 23 kwietnia 1564 </data-ur> w
<miejsce-ur> Stratford-upon-Avon </miejsce-ur>, zm.
<data-śm> 23 kwietnia 1616 </data-śm> tamże) - angielski
poeta, dramaturg, aktor. </hasło>
```

```
<zamówienie id="1">
  <pozycja>
    <nazwa> cyklotron </nazwa>
    <jednostka> sztuka </jednostka>
    <ilość> 3 </ilość>
  </pozycja>
  <zamawiający id="2" />
</zamówienie>
```

31

XML „tekstowy" a XML „bazodanowy"
Model „tekstowy znormalizowany"

```
<hasło>
  <osoba> William Szekspir </osoba>
  <tekst> (ang. </tekst>
  <wariant> William Shakespeare </wariant>
  <tekst> ; ur. prawdopodobnie </tekst>
  <data-ur> 23 kwietnia 1564 </data-ur>
  <tekst> w </tekst>
  <miejsce-ur> Stratford-upon-Avon </miejsce-ur>
  <tekst> , zm. </tekst>
  <data-śm> 23 kwietnia 1616 </data-śm>
  <tekst> tamże) – angielski poeta, dramaturg, aktor. </tekst>
</hasło>
```

32

Dylemat projektanta – element czy atrybut?

```
<zamówienie>
  <nr> 12345 </nr>
  <zamawiający> FTIMS </zamawiający>
  <kodTowaru> 987512 </kodTowaru>
  <kodTowaru> 657493 </kodTowaru>
</zamówienie>
```

```
<zamówienie nr="12345" zamawiający="FTIMS" kodTowaru="987512 657493" />
```

```
<zamówienie>
  Zamówienie <nr> 12345 </nr>
  dla <jednostka> FTIMS </jednostka>
  na dostawę towaru <towar kod="987512"> kserokopiarki </towar>
  i <towar kod="657493"> papier ksero </towar>
</zamówienie>
```

33

Dylemat projektanta – element czy atrybut?

- Warto pamiętać o pewnych dobrych praktykach
  - W elementach warto przechowywać dane, w atrybutach metadane
  - W elementach zapisujemy tekst wizualizowany, w atrybutach jego parametry
  - Elementy zagnieżdżone nadają się świetnie do przechowywania informacji szczegółowych (opisy, listy, pozycje, ...), bo atrybuty nie mogą zawierać wewnętrznej struktury (niby można użyć atrybutów typu NMTOKENS i IDREFS, ale ...)
  - Kolejność atrybutów nie ma znaczenia, natomiast w przypadku elementów jest ważna
  - W DTD tylko atrybutom można nadawać wartości domyślne, stałe, określać typ wyliczeniowy

34

Dylemat projektanta – element czy atrybut?

- Zalety elementów
  - Mogą zawierać inne elementy i atrybuty
  - Mogą się powtarzać
  - Można bezpiecznie rozszerzać ich model zawartości
  - Można dokładnie sterować sposobem ich występowania - kolejność, liczba wystąpień
- Zalety atrybutów
  - Szybsze do definiowania
  - Mogą mieć wartości domyślne

35

Związanie dokumentu XML-owego z DTD – wariant 1

- Można zapisać DTD bezpośrednio w dokumencie (**wewnętrzne DTD**), w którym będzie użyte, podając po słowie kluczowym DOCTYPE nazwę elementu głównego, a następnie definicje elementów i atrybutów
- Zalety
  - Stosowanie jest wygodne, gdy ograniczymy się do danego dokumentu
  - Potrzeba testowania przykładowego dokumentu pod kątem zgodności DTD dla złożonego projektu

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE opis [
      <!ELEMENT opis (#PCDATA)>
]>
<opis>To jest pierwszy dokument z wewnętrznym DTD</opis>
```

36

Wewnętrzny DTD

```
<?xml version="1.0" ?>
<!DOCTYPE zamówienie [
          <!ELEMENT zamówienie (klient)>
          <!ELEMENT klient (id_konta, nazwa)>
          <!ELEMENT id_konta (#PCDATA)>
          <!ELEMENT nazwa (imię, nazwisko)>
          <!ELEMENT imię (#PCDATA)>
          <!ELEMENT nazwisko (#PCDATA)>
]>
<zamówienie>
 <klient>
  <id_konta>1234556</id_konta>
  <nazwa>
   <imię>Jan</imię>
   <nazwisko>Kowalski</nazwisko>
  </nazwa>
 </klient>
</zamówienie>
```

```
▼<zamówienie>
 ▼<klient>
    <id_konta>1234556</id_konta>
  ▼<nazwa>
     <imię>Jan</imię>
     <nazwisko>Kowalski</nazwisko>
   </nazwa>
 </klient>
</zamówienie>
```

37

Związanie dokumentu XML-owego z DTD – wariant 2

- Można zdefiniować DTD osobno i użyć go podając URI pliku
- Zalety
  - Możliwość zastosowania do wielu dokumentów
- Rodzaje definicji
  - **Publiczne DTD** są definicjami, które zostały poddane standaryzacji i zapewniają ogólnie dostępny zestaw reguł tworzenia określonych typów dokumentów XML
  - **Niepubliczne DTD** są definicjami utworzonymi przez organizacje prywatne lub pojedyncze osoby

38

Publiczne DTD

```
<!DOCTYPE spec PUBLIC
      "-//W3C//DTD Specification V2.0//EN"
      "/XML/1998/06/xmlspec-v20.dtd">
```

- Opis elementów
  - Znak minus wskazuje, że DTD nie jest rozpoznanym standardem
    Znak plus oznaczałby, że DTD jest standardem rozpoznanym
  - `W3C` określa właściciela DTD, w tym wypadku W3C
  - `DTD Specification V2.0` jest to typ dokumentu DTD i numeru wersji. Etykieta może zawierać dowolne znaki standardowe za wyjątkiem podwójnego ukośnika (//)
  - `EN` dwuliterowy skrót języka dokumentu XML, do którego dana definicja DTD jest stosowana
    - Pełna lista skrótów określających języki znajduje się w specyfikacji ISO 639-1

39

Niepubliczne DTD

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

<opis>
      To jest pierwszy dokument z wewnętrznym DTD
</opis>
```

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

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE opis SYSTEM "http://www.przyklad.pl/opisDTD.dtd">

<opis>
      To jest pierwszy dokument z wewnętrznym DTD
</opis>
```

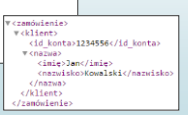
```
<!ELEMENT opis (#PCDATA)>
```

40

Niepubliczne DTD

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE zamówienie SYSTEM "zewnetrznyDTD.dtd">
<zamówienie>
  <klient>
    <id_konta>1234556</id_konta>
    <nazwa>
      <imię>Jan</imię>
      <nazwisko>Kowalski</nazwisko>
    </nazwa>
  </klient>
</zamówienie>
```

```
<!ELEMENT zamówienie (klient)>
<!ELEMENT klient (id_konta, nazwa)>
<!ELEMENT id_konta (#PCDATA)>
<!ELEMENT nazwa (imię, nazwisko)>
<!ELEMENT imię (#PCDATA)>
<!ELEMENT nazwisko (#PCDATA)>
```



41

Związanie dokumentu XML-owego z DTD – wariant 3

- Można połączyć obie metody (gdy chce się uzupełnić/zmienić istniejące DTD)

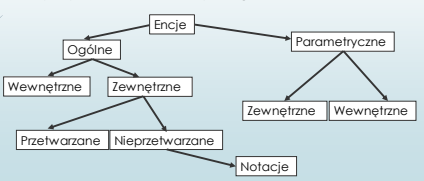
```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE zamówienie SYSTEM "zewnetrznyDTD.dtd" [
  <!ATTLIST klient nr CDATA #IMPLEID>
]>
<zamówienie>
  <klient nr="1">
    <id_konta>1234556</id_konta>
    <nazwa>
      <imię>Jan</imię>
      <nazwisko>Kowalski</nazwisko>
    </nazwa>
  </klient>
</zamówienie>
```

```
<!ELEMENT zamówienie (klient)>
<!ELEMENT klient (id_konta, nazwa)>
<!ELEMENT id_konta (#PCDATA)>
<!ELEMENT nazwa (imię, nazwisko)>
<!ELEMENT imię (#PCDATA)>
<!ELEMENT nazwisko (#PCDATA)>
```

42

Fizyczna struktura dokumentu – encje

- Umożliwiają one nazywanie fragmentów tekstu (znaków, ciągów znaków, zawartości plików) w celu ich późniejszego wykorzystania
- Dzięki temu, poprzez odwołanie do tych nazw, możliwe jest wielokrotne wykorzystanie raz zadeklarowanych fragmentów



43

Encje ogólne wewnętrzne

- Skrót odniesień do tekstu lub danych, które w dokumencie powinny zostać zamienione
- Są zawsze wartościami tekstowymi
- Znaki zarezerwowane dla składni XML-owej daje się uzyskać używając tzw. encji predefiniowanych (`&` dla &,`<` dla <, `>` dla >, `&apos;` dla ', `&quot;` dla ")
- Odwołania do znaków
  - `&#kod;` - kod dziesiętny znaku
  - `&#xkod;` - kod szesnastkowy znaku

```
<!ENTITY nazwaJednostki zawartośćJednostki>
```

```
&nazwaJednostki;
```

44

Encje ogólne wewnętrzne

```
<?xml version="1.0" ?>
<!DOCTYPE student [
  <!ENTITY ftims "Fizyka Techniczna, Informatyka i Matematyka Stosowana">
  <!ENTITY OiZ "Organizacja i zarządzanie">
  <!ELEMENT student (imię, nazwisko)>
  <!ATTLIST student wydział CDATA "&ftims;">
  <!ELEMENT imię (#PCDATA)>
  <!ELEMENT nazwisko (#PCDATA)>
]>
<student>
  <imię>Jan</imię>
  <nazwisko>Kowalski</nazwisko>
</student>
```

```
<student wydział="Fizyka Techniczna, Informatyka i Matematyka Stosowana">
  <imię>Jan</imię>
  <nazwisko>Kowalski</nazwisko>
</student>
```

```
<student wydział="&OiZ;">
  <imię>Jan</imię>
  <nazwisko>Kowalski</nazwisko>
</student>
```

```
<student wydział="Organizacja i zarządzanie">
  <imię>Jan</imię>
  <nazwisko>Kowalski</nazwisko>
</student>
```

45

Encje ogólne zewnętrzne

- Mogą być tekstem lub innym typem danych (np. obraz) i są zawsze przechowywane w zewnętrznym pliku
  - Przetwarzane – mogą być one prywatne (zadeklarowane poprzez słowo `SYSTEM`) lub publiczne (zadeklarowane poprze słowo `PUBLIC`)
  - Nieprzetwarzane jednostki są przesyłane do aplikacji obsługującej dokument (dane)

```
<!ENTITY nazwaJednostki SYSTEM "plik.ent">
```

```
&nazwaJednostki;
```

```
<!ENTITY nazwaJednostki PUBLIC "plik.ent">
```

```
&nazwaJednostki;
```

46

Encje ogólne zewnętrzne

```
<?xml version="1.0" ?>
<!DOCTYPE student [
  <!ENTITY ftims SYSTEM "ftims.ent">
  <!ELEMENT student (imię, nazwisko, wydział)>
  <!ELEMENT imię (#PCDATA)>
  <!ELEMENT nazwisko (#PCDATA)>
  <!ELEMENT wydział (#PCDATA)>
]>
<student>
  <imię>Jan</imię>
  <nazwisko>Kowalski</nazwisko>
  <wydział>&ftims;</wydział>
</student>
```

```
<?xml version="1.0" ?>
<!DOCTYPE student (view source for full doctype...)>
<student>
  <imię>Jan</imię>
  <nazwisko>Kowalski</nazwisko>
- <wydział>
    <ftims>Fizyka Techniczna, Informatyka i Matematyka Stosowana</ftims>
    <przedmiot>Programowanie komunikacji człowiek-komputer</przedmiot>
    <semestr>7</semestr>
  </wydział>
</student>
```

```
<ftims>Fizyka Techniczna, Informatyka i Matematyka Stosowana</ftims>
<przedmiot>Programowanie komunikacji człowiek-komputer</przedmiot>
<semestr>7</semestr>
```

47

Encje w atrybutach i notacje

- Nazwa encji nieprzetwarzanej (ang. *unparser external entity*) może być też wartością atrybutu typu ENTITY
- Dostarczają informacji dotyczących nieanalizowanej części dokumentu oraz takich, które są najczęściej wykorzystywane razem z zewnętrzną nieanalizowaną encją ogólną
- Analizatory XML nie wykorzystują notacji
- Są natychmiast przesyłane do aplikacji obsługującej dokument

```
<!NOTATION nazwaTypDanych SYSTEM adresDoDanych>
```

48

Encje w atrybutach i notacje

```
DTD:
<!DOCTYPE rysunek [
  <!ELEMENT rysunek EMPTY>
  <!ATTLIST rysunek plik ENTITY #REQUIRED>
  <!ENTITY mójRysunek SYSTEM
      "http://mojaStrona/rys1.jpg" NDATA jpeg>
  <!NOTATION jpeg SYSTEM "image/jpg">
]>
```

Aby móc użyć takiego zapisu, encji musi zostać przypisana **notacja** – **poprzez określenie formatu encji oraz osobną deklarację notacji** wraz z dodatkową informacją dla aplikacji zewnętrznej

```
XML:
<rysunek plik="mójRysunek">
```

Element rysunek odwołuje się do fotografii http://mojaStrona/rys1.jpg

49

Encje parametryczne wewnętrzne

```
<!ENTITY % nazwaJednostki zawartoscJednostki>
%nazwaJednostki;
```

- Uniknięcie powtarzania części definicji DTD
- Daje to jedną lokalizację dla wspólnych i powtarzalnych elementów

```
<!ELEMENT firma (nazwa, ulica, miasto, kod,        telefon*, email*, web*, inne*)>
<!ELEMENT kontakt (nazwa, ulica, miasto, kod,      telefon*, email*, web*, inne*)>
<!ELEMENT osoba  (nazwa, ulica, miasto, kod,       telefon*, email*, web*, inne*)>
```

```
<!ENTITY % dane_kontaktowe "nazwa, ulica, miasto, kod">
<!ENTITY % dane_opcjonalne "telefon*, email*, web*, inne*">
```

```
<!ELEMENT firma (%dane_kontaktowe;, %dane_opcjonalne;)>
<!ELEMENT kontakt (%dane_kontaktowe;, %dane_opcjonalne;)>
<!ELEMENT osoba (%dane_kontaktowe;, %dane_opcjonalne;)>
```

50

Encje parametryczne zewnętrzne

- Umożliwienie dołączania jednej DTD do innej
- Deklaracja i wykorzystanie są bardzo podobne do deklaracji wewnętrznych jednostek parametrycznych
  - Deklaracja rozpoczyna się słowem kluczowym `ENTITY`, po którym znajduje się znak procentu (`%`)
  - Następnie w deklaracji umieszcza się słowo kluczowe `SYSTEM` lub `PUBLIC`, które wskazuje że źródło jednostki znajduje się w pliku zewnętrznym
  - Na końcu, wewnątrz znaków cudzysłowu, umieszczony zostaje URI

```
<!ENTITY % nazwaJednostki SYSTEM plikDTD>
%nazwaJednostki;
```

51

Ograniczenia DTD

- Brak możliwości ograniczenia tekstowej treści elementów
- Brak wartości domyślnych dla elementów
- Słabe możliwości kontroli typów atrybutów
- Brak możliwości deklarowania typów zawartości wykorzystywanych dla wielu elementów i atrybutów
- Brak deklaracji dowolnego elementu lub atrybutu, dla którego użytkownik sam dobiera nazwę w dokumencie XML
- Brak możliwości zmiany kolejności elementów
- Brak możliwości ustalenia kolejności atrybutów
- DTD zawarte w standardzie XML 1.0 nie wspiera przestrzeni nazw

52

Ograniczenia DTD

- Słaba modułowość
- Brak rozbudowanych możliwości dokumentowania
- Brak możliwości uzależnienia treści lub wartości atrybutów od zawartości elementów lub atrybutów
- Brak możliwości kontroli typów w zależności od miejsca wystąpienia podelementu
- Typ ID dla atrybutów ma zbyt wiele ograniczeń
- Syntaktyka DTD jest inna niż zwykłego dokumentu XML

53

DTD vs XML Schema

- Wywodzi się z SGML-a
- Specyficzna składnia
- 10 typów danych
- Brak kontroli tekstowej zawartości elementów
- Typowy mieszany model zawartości

- Zaprojektowany na potrzeby XML
- Składnia XML
- 44 wbudowane typy proste
- Zaawansowana kontrola tekstowej zawartości elementów
- Możliwość definiowania własnych typów danych