

Czy już warto używać XML?

Tomasz Traczyk
ttraczyk@ia.pw.edu.pl
<http://www.ia.pw.edu.pl/~ttraczyk/>



Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych
Instytut Automatyki i Informatyki Stosowanej

Czy już warto używać XML

- **Przypomnienie: podstawy XML**
- **Przetwarzanie dokumentów w XML**
- **Narzędzia dla XML**
- **Zastosowania XML**
- **Podsumowanie**



Co to jest XML?

Extensible Markup Language

- Metajęzyk
 - umożliwia definiowanie języków znakowania
 - służy do zapisu różnorodnych dokumentów i struktur danych
 - przeznaczony m.in. do użycia w WWW

Geneza XML

- Podzbiór SGML (*Standard Generalised Markup Language*)
- Uproszczony przez usunięcie trudnych i niekoniecznych konstrukcji
- Dostosowany do przetwarzania w WWW

Co zapewnia XML?

- Możliwość definiowania
 - struktury dokumentu
 - znaczników (*tags*)
- Prostotę
 - łatwość opanowania przez użytkowników
 - względnie proste przetwarzanie
- Łatwość interpretacji dokumentu
 - przez przeglądarki
 - przez człowieka
 - nawet bez definicji struktury dokumentu
- Możliwość tworzenia połączeń (*links*)
 - między dokumentami
 - przez sieć



Dokument w XML

Dokument w XML

- Plik tekstowy
- Znaczniki
 - początkowe ujęte w znaki < >
 - końcowe ujęte w </ >
- Budowa dokumentu
 - prolog
 - element główny
 - elementy podrzędne
- Możliwe użycie łączników (*links*)

Znakowanie znaczeniowe

- Znaczniki odzwierciedlają strukturę dokumentu/danych
- Znacznikom nie jest z góry przypisany sposób prezentacji
- Do formatowania wizualnego konieczne użycie *style-sheets*

Podstawowe składniki towarzyszące dokumentowi

- DTD (*Document Type Definition*)
 - definicje elementów
 - określenie możliwego następstwa elementów
 - definicje atrybutów elementów
 - definicje encji (*entities*)
- Arkusze stylistyczne (*style-sheets*)
 - określenie sposobu prezentacji dla elementów
 - możliwa różna prezentacja tego samego dokumentu w zależności od środowiska
 - przetwarzanie, np.
 - ♦ XML → HTML
 - ♦ XML → XML
 - język XSL (*eXtensible Style Language*)



Dokument w XML — przykład

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<!DOCTYPE eres_konpekty SYSTEM "konpekty.dtd">
<?xml-stylesheet type="text/xsl" href="konpekty.xsl"?>
<eres_konpekty>
  <przedmiot id="KBD2" wersja="1">
    <konpekt>
      <czesc_konpektu id="Streszczenie">
        <P> Monograficzny przedmiot poświęcony bazie danych i
          narzędziom Oracle. </P>
      </czesc_konpektu>
      <czesc_konpektu id="Treść przedmiotu">
        <P> Omawiane są podstawowe zagadnienia związane z użytkowaniem
          RDBMS Oracle8 i Oracle8<I>i</I>. </P>
        <P> Przedstawiane są także narzędzia do budowy aplikacji: </P>
        <UL>
          <LI> Oracle Forms, </LI>
          <LI> Oracle Reports </LI>
        </UL>
        <P>oraz narzędzia do tworzenia dynamicznych serwisów WWW.</P>
      </czesc_konpektu>
    </konpekt>
  </przedmiot>
</eres_konpekty>
```



Dokument — po kolei

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<!DOCTYPE eres_konpekty SYSTEM "konpekty.dtd">
<?xml-stylesheet type="text/xsl" href="konpekty.xsl"?>
<eres_konpekty>
  <przedmiot id="KBD2" wersja="1">
    <konpekt>
      <czesc_konpektu id="Streszczenie">
        <P> Monograficzny przedmiot poświęcony bazie danych i
          narzędziom Oracle. </P>
      </czesc_konpektu>
      <czesc_konpektu id="Treść przedmiotu">
        <P> Omawiane są podstawowe zagadnienia związane z użytkowaniem
          RDBMS Oracle8 i Oracle8<I>i</I>. </P>
        <P> Przedstawiane są także narzędzia do budowy aplikacji: </P>
        <UL>
          <LI> Oracle Forms, </LI>
          <LI> Oracle Reports </LI>
        </UL>
        <P>oraz narzędzia do tworzenia dynamicznych serwisów WWW.</P>
      </czesc_konpektu>
    </konpekt>
  </przedmiot>
</eres_konpekty>
```

Prolog

- Wersja XML (obowiązkowa) i strona kodowa (opcjonalna)*
- Odwołanie do DTD (opcjonalne)
- Odwołanie do arkusza stylistycznego (opcjonalne)*

* Instrukcje przetwarzania



Dokument — po kolei

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<!DOCTYPE eres_konspekty SYSTEM "konspekty.dtd">
<?xml-stylesheet type="text/xsl" href="konspekty.xsl"?>
<eres_konspekty>
  <przedmiot id="KBD2" wersja="1">
    <konspekt>
      <czesc_konspektu id="Streszczenie">
        <P> Monograficzny przedmiot poświęcony bazie danych i
          narzędziom Oracle.
        </czesc_konspektu>
      <czesc_konspektu id="
        <P> Omawiane są pod
          RDBMS Oracle8 i Ora
        <P> Przedstawiane s
        <UL>
          <LI> Oracle Forms,
          <LI> Oracle Reports
        </UL>
      </czesc_konspektu>
      <P>oraz narzędzia do tworzenia dynamicznych serwisów WWW.</P>
    </konspekt>
  </przedmiot>
</eres_konspekty>
```

Elementy

- Element główny — dokładnie jeden
- Elementy podrzędne
- Jawne zamykanie znaczników



Dokument — po kolei

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<!DOCTYPE eres_konspekty SYSTEM "konspekty.dtd">
<?xml-stylesheet type="text/xsl" href="konspekty.xsl"?>
<eres_konspekty>
  <przedmiot id="KBD2" wersja="1">
    <konspekt>
      <czesc_konspektu id="Streszczenie">
        <P> Monograficzny przedmiot poświęcony bazie danych i
          narzędziom Oracle. </P>
      </czesc_konspektu>
      <czesc_konspektu id="Treść przedmiotu">
        <P> Omawiane są pod
          RDBMS Oracle8 i Or
        <P> Przedstawiane
        <UL>
          <LI> Oracle Form
          <LI> Oracle Repo
        </UL>
        <P>oraz narzędzia do tworzenia dynamicznych serwisów WWW.</P>
      </czesc_konspektu>
    </konspekt>
  </przedmiot>
</eres_konspekty>
```

Atrybuty

- Uzupełniają elementy
- Wartości ujęte w cudzysłów



Dokument — inne składniki

Instrukcje przetwarzania

- Służą do sterowania aplikacjami przetwarzającymi
- Składnia
`<?nazwa atrybuty?>`
- Nazwy rozpoczynające się od `xml` — zastrzeżone

Sekcje CDATA

- W tekście wewnątrz sekcji CDATA nie są rozpoznawane znaki specjalne
- Stosowane do umieszczania fragmentów zawierających dużo znaków specjalnych
- Składnia
`<![CDATA[tekst]]>`

Komentarze

- Składnia
`<!-- tekst komentarza -->`

Encje (jednostki, *entities*)

- Rodzaj makrodefinicji
- Stosowane do
 - włączania części dokumentu (np. powtarzalnych, z plików zewnętrznych)
 - umieszczania znaków specjalnych w dokumencie
- Składnia odwołania do encji: `&nazwa;`
- Encje predefiniowane:
 - znaki specjalne, np. `<`;
 - referencje znakowe, np. `℞` (Unicode)
- Encje definiowane w DTD:
`<!ENTITY nazwa "tekst">`
lub
`<!ENTITY nazwa SYSTEM "URL">`



DTD – przykład

```
<!ELEMENT eres_konspekty (przedmiot)+ >
<!ELEMENT przedmiot (konspekt) >
<!ELEMENT konspekt (czesc_konspektu)+ >
<!ELEMENT czesc_konspektu (P|UL)+ >
<!ELEMENT P (#PCDATA|I)* >
<!ELEMENT I (#PCDATA) >
<!ELEMENT UL (LI)+ >
<!ELEMENT LI (#PCDATA) >

<!ATTLIST przedmiot id ID #REQUIRED
wersja CDATA #IMPLIED >
<!ATTLIST czesc_konspektu id ID #REQUIRED >
```



DTD — po kolei

```
<!ELEMENT eres_konpekty (przedmiot)+ >
<!ELEMENT przedmiot (konpekt) >
<!ELEMENT konpekt (czesc_konpektu)+ >
<!ELEMENT czesc_konpektu (P|UL)+ >
<!ELEMENT P (#PCDATA|I)* >
<!ELEMENT I (#PCDATA) >
<!ELEMENT UL (LI)+ >
<!ELEMENT LI (#PCDATA) >
```

```
<!ATTLIST c
<!ATTLIST p
```

Definicje elementów

- Definiują nazwy elementów
- Określają zawieranie i następstwo
- Określają zawartość elementu



DTD — po kolei

```
<!ELEMENT eres_konpekty (przedmiot)+ >
<!ELEMENT przedmiot (konpekt) >
<!ELEMENT konpekt (czesc_konpektu)+ >
```

```
<!ELEMENT
<!ELEMENT
<!ELEMENT
<!ELEMENT
<!ELEMENT
```

Definicje atrybutów

- Definiują nazwy atrybutów
- Określają typ atrybutu i ew. listę dopuszczalnych wartości
- Określają obowiązkowość atrybutów
- Podają wartości domyślne atrybutów opcjonalnych

```
<!ATTLIST czesc_konpektu id ID #REQUIRED >
<!ATTLIST przedmiot id ID #REQUIRED
wersja CDATA #IMPLIED >
```



DTD a schematy

DTD

- Definiuje składnię dokumentu
 - zawiera definicje elementów i atrybutów
 - określa ich dozwolone zagnieżdżanie i następstwo
- Nie określa typów danych
 - wszystkie atrybuty i elementy są tekstowe
 - możliwe jedynie określenie listy dopuszczalnych wartości dla atrybutu
- Składnia DTD jest dość dziwaczna

Schemat (*XML Schema*)

- Określa składnię dokumentu, podobnie jak DTD
- Precyzyjnie definiuje typy danych dopuszczalne w poszczególnych „komórkach” dokumentu
- Jest sam zapisany w XML
- Przypuszczalnie schematy wyprą w przyszłości DTD
- Na razie specyfikacja nie jest jeszcze ustabilizowana
 - *XML Schema* na etapie *Candidate Recommendation*
 - istnieją konkurencyjne propozycje
- Więcej szczegółów → **PL OUG Zakopane’2001**



Przestrzenie nazw

Motywacja

- Powstaje wiele słowników dziedzinowych
- Autorzy dokumentów powinni móc korzystać z wielu słowników i dodawać własne
- Powstaje ryzyko konfliktów nazw

Rozwiązanie – przestrzenie nazw (*XML namespaces*)

- Nazwy elementów poprzedzane prefiksem przestrzeni nazw i dwukropkiem
- Przestrzeń nazw definiowana przez podanie URI domeny, która zarządza danym słownikiem
- Odwołanie do przestrzeni nazw
 - przez podanie specjalnego atrybutu `xmlns:prefiks_przestrzeni`
 - wartość tego atrybutu wskazuje
 - URN słownika
 - URI domeny zarządzającej słownikiem
 - prefiks zdefiniowany w elemencie nadrzędnym może być wykorzystywany w podrzędnych
 - w jednym dokumencie można użyć wielu przestrzeni nazw



Poprawność dokumentu w XML

Dokument dobrze sformułowany (*well-formed*)

- Warunki
 - niepusta zawartość
 - wszystkie konstrukcje poprawne w sensie *well-formed* (wg specyfikacji języka)
 - wszystkie konstrukcje rozpoczęte muszą być jawnie zakończone
 - każda konstrukcja zawarta w innej musi być w niej zawarta w całości
 - tylko jedna konstrukcja (element główny) nie zawarta w żadnej innej
 - brak odwołań do niestandardowych definicji nie zawartych w tym dokumencie
- Nie musi mieć DTD (dokument *standalone*)
- Mimo braku DTD może być wyświetlany przez przeglądarki
- Przeznaczenie: tylko wyświetlanie

Dokument poprawny (*valid*)

- Warunki
 - jest *well-formed*
 - powiązany z istniejącym DTD
 - pełna zgodność zawartości z DTD
 - wszystkie konstrukcje poprawne w sensie *valid* (wg specyfikacji języka)
- Przeznaczenie: przetwarzanie
- Do przetwarzania na ogół potrzebny jest dostęp do DTD



XML a HTML

XML

- Przeznaczenie
 - metajęzyk
 - służy do definiowania języków
- Realizacja
 - podzbiór SGML
 - brak predefiniowanych znaczników
- Składnia
 - element niepusty musi być jawnie zakończony
 - element pusty musi być oznaczony `/>`
 - wrażliwy na wielkość liter w nazwach elementów i atrybutów
 - wartości atrybutów muszą być ujęte w cudzysłów

HTML

- Przeznaczenie
 - gotowy język
 - służy do zapisu stron WWW
- Realizacja
 - konkretyzacja SGML
 - zestaw predefiniowanych znaczników
- Składnia
 - niektóre elementy niepuste nie muszą być zakończone jawnie
 - elementy puste nie muszą być wyróżniane
 - niewrażliwy na wielkość liter
 - wartości atrybutów mogą, ale nie muszą być ujęte w cudzysłów



➤ **Przypomnienie: podstawy XML**

➤ **Przetwarzanie dokumentów w XML**

➤ **Narzędzia dla XML**

➤ **Zastosowania XML**

➤ **Podsumowanie**



Arkusze stylistyczne i XSL

Motywacja

- XML nie zawiera żadnych informacji o formatowaniu — znakowanie znaczeniowe
- Informacja o formatowaniu musi być zawarta w arkuszach stylistycznych

XSL (*eXtensible Stylesheet Language*)

- Uzupełnienie XML
- Służy do formatowania dokumentów XML — arkusze stylistyczne
- Może służyć do transformacji dokumentów XML i HTML

Zasada formatowania za pomocą XSL

- Transformacja drzewa znaczników na drzewo wynikowe
- Transformacją steruje skrypt w XSL
- Drzewo wynikowe może zawierać obiekty formatujące, które mają przypisane sposoby prezentacji
- Obiekty formatujące nie są na razie implementowane w typowych przeglądarkach

XSLT (*XSL Transformations*)

- Specyfikacja części XML służącej do transformacji XML
- Może być użyta do przekształcania np.:
 - XML → XML
 - XML → HTML
 - XML → tekst
- Transformacje są szczególnie przydatne do przetwarzania danych w s.i.



Arkuszyk stylistyczny w XSL – przykład

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>

<xsl:template match="/">
  <HTML><HEAD><TITLE>Konspekty</TITLE></HEAD>
  <BODY><H1>Konspekty przedmiotów</H1>
    <xsl:apply-templates select="eres_konspekty/przedmiot">
      <xsl:sort select="@id" />
    </xsl:apply-templates>
  </BODY>
</HTML>
</xsl:template>

<xsl:template match="przedmiot">
  <H2><xsl:value-of select="@id" /><xsl:value-of select="@wersja" /></H2>
  <xsl:apply-templates />
  <HR />
</xsl:template>

<xsl:template match="konspekt">
  <H3>Konspekt</H3>
  <xsl:apply-templates />
</xsl:template>
```



Arkuszyk stylistyczny w XSL – po kolei

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>

<xsl:template match="/">
  <HTML><HEAD><TITLE>Konspekty</TITLE></HEAD>
  <BODY><H1>Konspekty przedmiotów</H1>
    <xsl:apply-templates select="eres_konspekty/przedmiot">
      <xsl:sort select="@id" />
    </xsl:apply-templates>
  </BODY>
</HTML>
</xsl:template>

<xsl:template match="przedmiot">
  <H2><xsl:value-of select="@id" /><xsl:value-of select="@wersja" /></H2>
  <xsl:apply-templates />
  <HR />
</xsl:template>

<xsl:template match="konspekt">
  <H3>Konspekt</H3>
  <xsl:apply-templates />
</xsl:template>
```

Nagłówek

- Standardowy prolog XML
- Odwołanie do przestrzeni nazw xsl (XSLT)
- Określenie typu wprowadzanego dokumentu



Arkuszyk stylistyczny w XSL – po kolei

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
```

```
<xsl:template match="/">
  <HTML><HEAD><TITLE>Konspekty</TITLE></HEAD>
  <BO
  </B
  </HTML
</xsl:template>
```

Szablony xsl:template

- Przetwarzają elementy dopasowane do wzorca match
- Zawartość szablonu steruje tworzeniem drzewa wynikowego

```
<xsl:template match="przedmiot">
  <H2><xsl:value-of select="@id" />{<xsl:value-of select="@wersja" />}</H2>
  <xsl:apply-templates />
  <HR />
</xsl:template>

<xsl:template match="konspekt">
  <H3>Konspekt</H3>
  <xsl:apply-templates />
</xsl:template>
```



Arkuszyk stylistyczny w XSL – po kolei

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
```

```
<xsl:template match="/">
  <HTML><HEAD><TITLE>Konspekty</TITLE></HEAD>
  <BODY><H1>Konspekty przedmiotów</H1>
  <xsl:apply-templates select="eres_konspekty/przedmiot">
    <xsl:sort select="@id" />
  </xsl:apply-templates>
</BODY>
</HTML>
</xsl:tem
```

Przetwarzanie rekurencyjne

- Dopasowywanie do wzorców odbywa się rekurencyjnie
- Elementy xsl:apply-templates wywołują to przetwarzanie
- Dzięki rekurencji można przetworzyć całe drzewo wejściowe

```
<xsl:temp
  <H2><x
  <xsl:a
  <HR />
</xsl:template>

<xsl:template match="konspekt">
  <H3>Konspekt</H3>
  <xsl:apply-templates />
</xsl:template>
```



Arkuszyk stylistyczny w XSL – po kolei

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>

<xsl:template match="/">
  <HTML><HEAD><TITLE>Konspekty</TITLE></HEAD>
  <BODY><H1>Konspekty przedmiotów</H1>
    <xsl:apply-templates select="eres_konspekty/przedmiot">
      <xsl:sort select="@id" />
    </xsl:apply-templates>
  </BODY>
</HTML>
</xsl:template>

<xsl:template match="przedmiot">
  <H2><xsl:value-of select="@id" /><xsl:value-of select="@wersja" /></H2>
  <xsl:apply-templates />
  <HR />
</xsl:template>

<xsl:template match="Konspekt">
  <H3>Konspekt</H3>
  <xsl:apply-templates />
</xsl:template>
```

Tworzenie wyniku

- Napisy nie będące elementami XSL są wyprowadzane na wyjście
 - jeśli zawierają znaczniki, to muszą być *well-formed*
- Element `xsl:value-of` pozwala wyprowadzać wartości elementów i atrybutów dokumentu wejściowego



Arkuszyk stylistyczny – wynik

```
<?xml version="1.0" encoding="ISO-8859-2">
<!DOCTYPE eres_konspekty SYSTEM "konspekt" [
<?xml-stylesheet type="text/xsl" href="k"
</eres_konspekty>
  <przedmiot id="KBD2" wersja="1">
    <konspekt>
      <czesc_konspektu id="Streszczenie">
        <P> Monograficzny przedmiot poświęcony
        narzędziom Oracle. </P>
      </czesc_konspektu>
      <czesc_konspektu id="Treść przedmiotu">
        <P> Omawiane są podstawowe zagadnienia
        RDBMS Oracle8 i Oracle8<I>i</I></P>
        <P> Przedstawiane są także narzędzia
        <UL>
          <LI> Oracle Forms, </LI>
          <LI> Oracle Reports </LI>
        </UL>
        <P>oraz narzędzia do tworzenia dynami-
        </czesc_konspektu>
      </konspekt>
    </przedmiot>
  </eres_konspekty>
```



DOM i SAX

DOM (*Document Object Model*)

- Obiektowy model przetwarzania XML
 - dokument jest reprezentowany jako drzewo obiektów
 - tworzona jest reprezentacja całego dokumentu
- Standard W3C, podobny do DOM dla HTML
 - obiektowy model dokumentu
 - zbiór klas i metod do manipulowania dokumentem
- Cechy
 - duże zużycie zasobów i ograniczona wydajność
 - możliwe modyfikowanie dokumentów
- API DOM
 - wbudowane w przeglądarki WWW, np. MSIE 5
 - dostępne dla wielu języków programowania
 - zawarte w wielu dostępnych parserach, np. we wszystkich parserach Oracle

SAX (*Simple API for XML*)

- Model przetwarzania zorientowany na zdarzenia:
 - początek/koniec elementu lub innej części dokumentu powoduje zdarzenie
 - metody obsługi zdarzeń definiuje użytkownik
- Standard *de facto*
- Cechy
 - małe zużycie zasobów i dobra wydajność
 - brak prostych możliwości modyfikowania dokumentów
- API SAX
 - dostępne dla wielu języków programowania
 - zawarte w wielu dostępnych parserach, np. we większości parserów Oracle



DOM – przykład

```
import java.io.*; import java.net.*;
import org.w3c.dom.*; import org.w3c.dom.Node; import oracle.xml.parser.v2.*;

public class DOMSample {
    static public void main(String[] argv) {
        try {
            DOMParser parser = new DOMParser();
            parser.parse("file:///T:/xml/konspekty.xml");
            XMLDocument doc = parser.getDocument();

            NodeList nl = doc.getElementsByTagName("przedmiot");
            for (int j=0; j < nl.getLength(); j++) {
                Element e = (Element)nl.item(j);
                NamedNodeMap nnm = e.getAttributes();
                if (nnm != null) {
                    for (int i=0; i < nnm.getLength(); i++) {
                        Node n = nnm.item(i);
                        if (n.getNodeName()=="id") {
                            System.out.print(n.getNodeValue());
                        }
                    }
                }
            }
        } catch (Exception e) { /* tu obsługa wyjątków */ }
    }
}
```



DOM – po kolei

```
import java.io.*; import java.net.*;
import org.w3c.dom.*; import org.w3c.dom.Node; import oracle.xml.parser.v2.*;

public class
static pub
try {
    DOMP
    pars
    XMLD
    Node
    for
        Element e = (Element)nl.item(j);
        NamedNodeMap nnm = e.getAttributes();
        if (nnm != null) {
            for (int i=0; i<nnm.getLength(); i++) {
                Node n = nnm.item(i);
                if (n.getNodeName()=="id") {
                    System.out.print(n.getNodeValue());
                }
            }
        }
    } catch (Exception e) { /* tu obsługa wyjątków */ }
} }
```

Cel programu

- Wypisanie identyfikatorów przedmiotów

Importy

- Typowe składniki Środowiska Java
- DOM API
- *Oracle Parser for Java* wersja 2



DOM – po kolei

```
import java.io.*; import java.net.*;
import org.w3c.dom.*; import org.w3c.dom.Node; import oracle.xml.parser.v2.*;

public class DOMSample {
    static public void main(String[] argv) {
        try {
            DOMParser parser = new DOMParser();
            parser.parse("file:///T:/xml/konspekty.xml");
            XMLDocument doc = parser.getDocument();

            Node
            for
                Element e = (Element)nl.item(j);
                NamedNodeMap nnm = e.getAttributes();
                if (nnm != null) {
                    for (int i=0; i<nnm.getLength(); i++) {
                        Node n = nnm.item(i);
                        if (n.getNodeName()=="id") {
                            System.out.print(n.getNodeValue());
                        }
                    }
                }
            } catch (Exception e) { /* tu obsługa wyjątków */ }
        }
    }
}
```

Analiza leksykalna

- Parser analizuje plik XML z podanego URL
- Wynikiem jest drzewo węzłów, odzwierciedlające strukturę całego dokumentu



DOM – po kolei

```
import java.io.*; import java.net.*;
import org.w3c.dom.*; import org.w3c.dom.Node; import oracle.xml.parser.v2.*;

public class ...
    static public ...
        try {
            DOMParser parser = new DOMParser();
            XMLDocument doc = parser.parse("file:///T:/xml/konspekty.xml");

            NodeList nl = doc.getElementsByTagName("przedmiot");
            for (int j=0; j < nl.getLength(); j++) {
                Element e = (Element)nl.item(j);
                NamedNodeMap nnm = e.getAttributes();
                if (nnm != null) {
                    for (int i=0; i < nnm.getLength(); i++) {
                        Node n = nnm.item(i);
                        if (n.getNodeName()=="id") {
                            System.out.print(n.getNodeValue());
                        }
                    }
                }
            }
        } catch (Exception e) { /* tu obsługa wyjątków */ }
    }
}
```

„Wymowanie” potrzebnych elementów

- Znalezienie elementów o znaczniku przedmiot
- Przejrzenie listy atrybutów
- Znalezienie wartości atrybutu o nazwie id



SAX – przykład

```
import org.xml.sax.*; import java.io.*; import java.net.*;
import oracle.xml.parser.v2.*;

public class SAXSample extends HandlerBase {

    static public void main(String[] argv) {
        SAXSample sample = new SAXSample();
        Parser parser = new SAXParser();
        parser.setDocumentHandler(sample);
        try {
            parser.parse("file:///T:/xml/konspekty.xml");
        }
        catch (Exception e) { /* tu obsługa wyjątków */ }
    }

    public void startElement(String name, AttributeList atts) throws SAXException {
        if (name=="przedmiot") {
            for (int i=0;i<atts.getLength();i++) {
                String aname = atts.getName(i);
                if (aname=="id") {
                    String value = atts.getValue(i);
                    System.out.println(value);
                }
            }
        }
    }
}
```



SAX – po kolei

```
import org.xml.sax.*; import java.io.*; import java.net.*;
import oracle.xml.parser.v2.*;
```

```
public class
```

Cel programu

- Wypisanie identyfikatorów przedmiotów

Imports

- Typowe składniki Środowiska Java
- API SAX
- Oracle XML Parser for Java wersja 2

```
static public void main(String[] argv) {
    SAXSample sample = new SAXSample();
    Parser parser = new SAXParser();
    parser.setDocumentHandler(sample);
    try {
        parser.parse("file:///T:/xml/konspekty.xml");
    } catch (Exception e) { /* tu obsługa wyjątków */ }
}

public void startElement(String name, AttributeList atts) throws SAXException {
    if (name=="przedmiot") {
        for (int i=0;i<atts.getLength();i++) {
            String aname = atts.getName(i);
            if (aname=="id") {
                String value = atts.getValue(i);
                System.out.println(value);
            }
        }
    }
}
```



SAX – po kolei

```
import org.xml.sax.*; import java.io.*; import java.net.*;
import oracle.xml.parser.v2.*;
```

```
public class SAXSample extends HandlerBase {
```

```
static public void main(String[] argv) {
    SAXSample sample = new SAXSample();
    Parser parser = new SAXParser();
    parser.setDocumentHandler(sample);
    try {
        parser.parse("file:///T:/xml/konspekty.xml");
    }
    catch (Exception e) { /* tu obsługa wyjątków */ }
}
```

Analiza leksykalna

- Podłączenie klasy zawierającej metody obsługi zdarzeń
- Wywołanie parsera dla podanego URL

```
public void startElement(String name, AttributeList atts) throws SAXException {
    if (name=="przedmiot") {
        for (int i=0;i<atts.getLength();i++) {
            String value = atts.getValue(i);
            System.out.println(value);
        }
    }
}
```



SAX – po kolei

```
import org.xml.sax.*; import java.io.*; import java.net.*;
import oracle.xml.parser.v2.*;
```

```
public class SAXSample extends HandlerBase {
```

```
    static pub
    SAXSamp
    Parser
    parser.
    try {
        pars
    }
    catch (
```

Obsługa zdarzeń

- Metoda obsługuje zdarzenie przeczytania początkowego znacznika elementu
- Jako parametr otrzymuje listę atrybutów
- Jeśli element jest szukanym elementem o znaczniku `przedmiot`, to znajduje się wartość atrybutu `id`

```
    public void startElement(String name, AttributeList atts) throws SAXException {
        if (name=="przedmiot") {
            for (int i=0;i<atts.getLength();i++) {
                String aname = atts.getName(i);
                if (aname=="id") {
                    String value = atts.getValue(i);
                    System.out.println(value);
                }
            }
        }
    }
}
```



- Przypomnienie: podstawy XML
- Przetwarzanie dokumentów w XML
- **Narzędzia dla XML**
- Zastosowania XML
- Podsumowanie



Oracle XDK

Oracle XML Developer's Kit

- Podstawowe składniki
 - parsery XML dla różnych języków programowania
 - ♦ Java
 - ♦ C
 - ♦ C++
 - ♦ PL/SQL
 - procesor XSLT
 - procesor schematów XML Schema
 - generatory klas dla języków obiektowych
- Gotowe rozwiązania
 - *Oracle XML SQL Utility (XSU)*
 - *Oracle XSQL Servlet*
 - *Oracle XML Transviewer Beans*



Oracle XDK c.d.

Oracle XDK for JAVA

- *XML Parser/XSLT Processor for Java v. 2*
 - DOM Level 1, SAX 1.0
 - XSLT, *XML Namespaces*
 - przetwarzanie z walidacją i bez
 - liczne strony kodowe, w tym ISO-8859-2
 - możliwość wykorzystania we wszystkich warstwach s.i.
 - procesor XSL dostępny z linii komendy:
`java oracle.xml.parser.v2.oraxsl`
- *XML Parser for Java v. 2.0.2.9 beta*
 - zawiera także DOM2 i SAX2 (częściowo)
- *Oracle XML Schema Processor for Java*
 - uzupełnienie parsera o analizę typów danych
- *XML Class Generator for Java*
 - generuje klasy języka Java na podstawie DTD lub schematu

Oracle XDK for PL/SQL

- *XML Parser for PL/SQL*
 - DOM Level 1
 - XSLT
 - brak SAX i akceptowania przestrzeni nazw
- Sposób realizacji
 - osłona w PL/SQL do narzędzi w Javie
 - działa w oparciu o *Oracle8i Java VM*
- Więcej szczegółów i języków → w artykule



DOM w PL/SQL – przykład

```
create or replace function domsample return varchar2 is
  p xmlparser.parser;
  doc xmldom.DOMDocument;
  nl xmldom.DOMNodeList;
  n xmldom.DOMNode;
begin
  p := xmlparser.newParser;
  xmlparser.parse(p, 'file:///T:/xml/konspekty.xml');
  doc := xmlparser.getDocument(p);

  nl := xmldom.getElementsByTagName(doc, 'przedmiot');

  for j in 0..xmldom.getLength(nl)-1 loop
    n := xmldom.item(nl, j);
    return xmldom.getAttribute(xmldom.makeElement(n), 'id');
  end loop;

exception when others then
  raise_application_error(-20000, 'Błąd');
end domsample;
```



DOM w PL/SQL – po kolei

```
create or replace function domsample return varchar2 is
  p xmlparser.parser;
  doc xmldom.DOMDocument;
  nl xmldom.DOMNodeList;
  n xmldom.DOMNode;
begin
  p := xml
  xmlparser.
  doc := xml

  nl := xml

  for j in 0
    n := xm
    return
  end loop;

exception when others then
  raise_application_error(-20000, 'Błąd');
end domsample;
```

Cel programu

- Wypisanie identyfikatorów przedmiotów

Typy i zmienne

- Oracle XML Parser for PL/SQL jest w rzeczywistości jedynie osłoną wywołującą parser w Javie
- Te specjalne typy w środku zawierają po prostu numery „uchwyty”



DOM w PL/SQL – po kolei

```
create or replace function domsample return varchar2 is
  p xmlparser.parser;
  doc xmldom.DOMDocument;
  nl xmldom.DOMNodeList;
  n xmldom.DOMNode;
begin
  p := xmlparser.newParser;
  xmlparser.parse(p, 'file:///T:/xml/konpekty.xml');
  doc := xmlparser.getDocument(p);

  nl := xmldom.getDocumentNodeList(doc);
  for j in 0..nl.getLength-1 loop
    n := xmldom.item(nl, j);
    return n.getNodeName;
  end loop;
exception when others then
  raise_application_error(-20000, 'Błąd');
end domsample;
```

Analiza leksykalna

- Parser analizuje dokument XML z podanego URL
- Możliwe jest także wczytywanie dokumentów ze zmiennych VARCHAR2 lub CLOB
- Wynikiem analizy leksykalnej jest drzewo odzwierciedlające strukturę całego dokumentu – zmienna `doc` zawiera „uchwyt” do takiego drzewa



DOM w PL/SQL – po kolei

```
create or replace function domsample return varchar2 is
  p xmlparser.parser;
  doc xmldom.DOMDocument;
  nl xmldom.DOMNodeList;
  n xmldom.DOMNode;
begin
  p := xmlparser.newParser;
  xmlparser.parse(p, 'file:///T:/xml/konpekty.xml');
  doc := xmlparser.getDocument(p);

  nl := xmldom.getElementsByTagName(doc, 'przedmiot');

  for j in 0..nl.getLength-1 loop
    n := xmldom.item(nl, j);
    return xmldom.getAttribute(xmldom.makeElement(n), 'id');
  end loop;
exception when others then
  raise_application_error(-20000, 'Błąd');
end domsample;
```

„Wydobycie” szukanego elementu

- Znalezienie listy elementów o szukanym znaczniku `przedmiot`
- Przeszukanie tej listy
- Znalezienie wartości atrybutu o nazwie `id`



Oracle XML SQL Utility (XSU)

Podstawowe funkcje

- Generowanie XML na podstawie zapytania SQL
 - jako tekstu
 - jako struktury DOM
 - jako ciągu zdarzeń SAX2
- Generowanie DTD i schematów *XML Schema*
- Przekształcenia generowanych dokumentów
 - proste: zmiana nazw znaczników
 - złożone: użycie XSLT
- Wykonywanie DML na podstawie XML
 - wprowadzanie
 - modyfikacja
 - kasowanie
- Możliwość przetwarzania złożonych dokumentów XML
 - współdziałanie z XSLT
 - użycie struktur obiektowych
 - generowanie atrybutów

Współdziałanie

- Wykorzystuje
 - *Oracle XML Parser for Java v. 2*
 - JDBC
- Dostępność narzędzia
 - dostarczane z *Oracle8i 8.7.1* oraz *Oracle9i*
 - dostępne także osobno
- Możliwości użycia
 - w bazie danych ($\geq 8.1.6$; Java API lub PL/SQL API)
 - na serwerze aplikacyjnym (Java API)
 - na serwerze Web (servlet)
 - na kliencie (aplet albo aplikacja)



XSU – mapowanie

Standardowe mapowanie

- Wynik zapytania SQL jest mapowany w standardową strukturę XML
 - `<ROWSET>`
 - `<ROW num="numer_rekordu">`
 - znaczniki odpowiadające kolumnom

Przykład

```
select id_przedmiotu as "id", wersja, nazwa
from przedmioty p order by id_przedmiotu
```

```
<?xml version = '1.0'?>
<ROWSET>
  <ROW num="1">
    <id>KBD2</id>
    <WERSJA>1</WERSJA>
    <NAZWA>Komercyjne bazy danych 2</NAZWA>
  </ROW>
</ROWSET>
```

Proste zmiany mapowania

- Zmiana nazw
 - znaczników `ROWSET` i `ROW`
 - atrybutu `num`
 - użycie małych liter
- Usunięcie numeracji lub jej zastąpienie przez wartość wybranej kolumny
- Określenie formatu dla dat
- Użycie atrybutów zamiast elementów
 - kolumnie w zapytaniu należy nadać alias rozpoczynający się od `@`

Transformacje XSLT

- Po stronie klienta – dołączanie instrukcji przetwarzania `<?xml-stylesheet...?>`
- Po stronie XSU – rejestracja arkusza XSLT (nie polecane do dużych dokumentów – słaba wydajność)



XSU c.d.

Źródło danych

- Zapytanie SQL
- `ref cursor`
- `ResultSet` (JDBC)

Generowanie złożonych struktur

- Wykorzystanie danych relacyjno-obiektowych
- Użycie perspektyw obiektowych
- Użycie podzapytań typu `CURSOR`

Zapis złożonych danych

- Wykorzystanie perspektyw obiektowych
- Jeśli potrzeba, wykorzystanie wyzwalaczy `instead of`

Interfejsy XSU

- Java
- PL/SQL
- Linia komendy
`java OracleXML getXML|putXML parametry`
- XSQL Servlet



XSU w PL/SQL – przykład

```
create or replace function XSUSample return CLOB is
  ctxquery DBMS_XMLQuery.ctxType;
  result CLOB;
  query varchar2(255) := 'select id_przedmiotu as "id", wersja, nazwa
                        from przedmioty p order by id_przedmiotu';
  xsl varchar2(80) := 'file:///T:/xml/xsu/przedm.xsl';
begin
  ctxquery := DBMS_XMLQuery.newContext(query);
  DBMS_XMLQuery.setXSLT(ctxquery, xsl);
  result := DBMS_XMLQuery.getXML(ctxquery);
  DBMS_XMLQuery.closeContext(ctxquery);
  return result;
end;
```



XSU w PL/SQL – po kolei

```
create or replace function XSUSample return CLOB is
  ctxquery DBMS_XMLQuery.ctxType;
  result CLOB;
  query varchar2(255) := 'select id_przedmiotu as "id", wersja, nazwa
                        from przedmioty p order by id_przedmiotu';
  xsl varchar2(80) := 'file:///T:/xml/xsu/przedm.xsl';
begin
  ctxquery := DBMS_XMLQuery.newContext(query);
  DBMS_XMLQuery.setXSLT(ctxquery, xsl);
  result := DBMS_XMLQuery.getXML(ctxquery);
  DBMS_XMLQuery.closeContext(ctxquery);
  return result;
end;
```

Cel programu

- Wypisanie wyniku zapytania w postaci XML
- Transformacja XML za pomocą XSL

Inicjalizacja XSU

- Utworzenie kontekstu
- Rejestracja arkusza XSLT (przetwarzanie po stronie XSU)



XSU w PL/SQL – po kolei

```
create or replace function XSUSample return CLOB is
  ctxquery DBMS_XMLQuery.ctxType;
  result CLOB;
  query varchar2(255) := 'select id_przedmiotu as "id", wersja, nazwa
                        from przedmioty p order by id_przedmiotu';
  xsl varchar2(80) := 'file:///T:/xml/xsu/przedm.xsl';
begin
  ctxquery := DBMS_XMLQuery.newContext(query);
  DBMS_XMLQuery.setXSLT(ctxquery, xsl);
  result := DBMS_XMLQuery.getXML(ctxquery);
  DBMS_XMLQuery.closeContext(ctxquery);
  return result;
end;
```

Generowanie XML

- Generowanie dokumentu i transformacja XSLT
- Zwolnienie zasobów



XSQL Servlet

Co to jest?

- Servlet wykorzystujący XSU
- Umożliwia
 - zadawanie zapytań do bazy danych ze stron WWW
 - otrzymywanie wyniku w postaci XML
 - transformacje wyniku
 - zapis danych dostarczonych w postaci XML do bazy

XSQL Pages

- Strony WWW napisane w XML
- Wykorzystują
 - specjalnie zdefiniowaną składnię (w XML, przestrzeń nazw `xsql`)
 - zapytania SQL
 - parametry przekazane w URL

Współdzielenie

- Wykorzystuje
 - Oracle XML Parser for Java wersja 2
 - XML SQL Utility
 - JDBC (Oracle8i lub Oracle Lite)
 - serwer http z Servlet API 2.1 i JSP
- Dostępność narzędzia
 - dostarczane z Oracle8i 8.7.1 oraz Oracle9i
 - dostępne także osobno
- Możliwości użycia
 - wywoływanie stron XSQL przez serwer http
 - dostęp z linii komendy `xslt.bat`
 - wywoływanie z programów w Javie: `XSQLRequest.process()`
 - wynik w postaci tekstu lub obiektu DOM
 - włączanie do stron JSP



XSQL Servlet c.d.

Zapytania

- Instrukcja `<xsql:query>`
 - zawiera zapytanie SQL
 - zwraca dokument XML – jak w XSU
 - może wykorzystywać parametry z URL: `{@parametr}`
- Parametry połączenia z bazą są zapisane w pliku `XSQLConfig.xml`

Zapis dokumentów XML do bazy

- Instrukcja `<xsql:insert-request>`
 - powoduje zapis danych z XML do tabeli
 - dokument XML z danymi przesyła się do strony XSQL za pomocą metody POST
- Możliwe jest przetwarzanie przesłanych przez POST formularzy HTML – są one wstępnie przetwarzane na ustaloną postać XML

Transformacje XSLT

- Instrukcja `<?xml-stylesheet ... ?>` powoduje dokonanie transformacji
 - przez servlet (domyślnie)
 - przez klienta (`client="yes"`)
- Sterowanie transformacją przez parametry URL
 - `xml-stylesheet` – adres arkusza XSLT
 - `transform=client`
 - transformacja na kliencie
 - `xml-stylesheet=none`
 - zakaz transformacji

Dodatkowe możliwości

- Wykonywanie DML
- Włączanie XML i/lub XSQL do strony
- Wywołanie OWA
- Wywołanie programu w Javie
- Wykorzystanie `ref cursors`
- Ustawianie i odczyt `cookies`
- Ustalanie wartości parametrów na podstawie zapytania



XSQL Servlet – przykład

```
<?xml version="1.0"?>

<xsql:query connection="tt" xmlns:xsql="urn:oracle-xsql"
  rowset-element="eres_konspekty" row-element="przedmiot"
  id-attribute="id" id-attribute-column="id" tag-case="lower"
>
  select id_przedmiotu as "id", wersja as "wersja",
     cursor (select id_czesci as "id" from konspekty k
              where p.id_przedmiotu=k.id_przedmiotu and p.wersja=k.wersja
              order by id_czesci) as konspekt
  from przedmioty p
  where id_przedmiotu='{@parametr}'
  order by id_przedmiotu
</xsql:query>
```

Cel przetwarzania

- Generowanie wielopoziomowego dokumentu XML na podstawie zapytania SQL



XSQL Servlet – po kolei

```
<?xml version="1.0"?>

<xsql:query connection="tt" xmlns:xsql="urn:oracle-xsql"
  rowset-element="eres_konspekty" row-element="przedmiot"
  id-attribute="id" id-attribute-column="id" tag-case="lower"
>
  select id_przedmiotu as "id", wersja as "wersja",
     cursor (select id_czesci as "id" from konspekty k
              where p.id_przedmiotu=k.id_przedmiotu and p.wersja=k.wersja
              order by id_czesci) as konspekt
  from przedmioty p
  where id_przedmiotu='{@parametr}'
  order by id_przedmiotu
```

Zapytanie SQL

- Do struktury danych przedstawionej na ERD
- Podzapytanie typu CURSOR użyte dla uzyskania struktury wielopoziomowej
- Aliasy kolumn określają nazwy elementów XML
- Wykorzystano parametr z URL



XSQL Servlet – po kolei

```
<?xml version="1.0"?>

<xsql:query connection="tt" xmlns:xsql="urn:oracle-xsql"
  rowset-element="eres_konspekty" row-element="przedmiot"
  id-attribute="id" id-attribute-column="id" tag-case="lower"
>
  select id_przedmiot
  cursor (select id
          where p.id
          order by
  from przedmioty p
  where id_przedmiotu
  order by id_przedmi
</xsql:query>
```

Instrukcja <xsql:query>

- Powoduje generowanie XML na podstawie zapytania SQL zawartego w elemencie
- Atrybut `connection` określa połączenie z bazą danych
- Pozostałe atrybuty modyfikują standardowe znaczniki



XSQL Servlet – po kolei

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="przedm.xsl"?>
<xsql:query connection="tt" xmlns:xsql="urn:oracle-xsql"
  rowset-element="eres_konspekty" row-element="przedmiot"
  id-attribute="id" id-attribute-column="id" tag-case="lower"
>
  select id_przedmiot
  cursor (select id
          where p.id
          order by id_czesci) as konspekt
  from przedmioty p
  where id_przedmiotu='{@parametr}'
  order by id_przedmiotu
</xsql:query>
```

Transformacja XSLT

- Dodanie instrukcji <?xml-stylesheet> powoduje transformację po stronie serwetu
- Umieszczenie w tej instrukcji atrybutu `client="yes"` przenosi transformację na stronę klienta



XSQL Servlet – wynik

```
<?xml version="1.0"?>
```

```
<xsql:query connection="tt"
rowset-element="eres_konsp"
id-attribute="id" id-attr:
>
  select id_przedmiotu as
  cursor (select id_cze:
  where p.id_pr:
  order by id_c:
  from przedmioty p
  where id_przedmiotu='{@}
  order by id_przedmiotu
</xsql:query>
```



Czy już warto używać XML?

57

Oracle XML Transviewer Beans

Co to jest

- Zestaw komponentów typu *Java Beans* do tworzenia interfejsu graficznego do aplikacji XML

```
<?xml version="1.0"?>
<root>
  <row num="1">
    <id>KBD2
    </id>
    <wersja>1
    </wersja>
    <czesci_konspektow>
      <czesci_konspektow_row num="1">
        <id>Streszczenie
        </id>
      </czesci_konspektow_row>
      <czesci_konspektow_row num="2">
        <id>Treść
        </id>
      </czesci_konspektow_row>
    </czesci_konspektow>
  </row>
</root>
```

Składniki

- *DOMBuilder Bean* – interfejs do parsera DOM
- *SourceViewer Bean* – wyświetlanie źródłowych plików w XML, z wyróżnianiem elementów składni za pomocą kolorów i możliwością edycji
- *Tree Viewer Bean* – wyświetlanie struktury XML w formie drzewa
- *Transformer Bean* – transformacje XML na podstawie podanego skryptu XSL



Czy już warto używać XML?

58

XML w Oracle Reports 6i

Możliwości

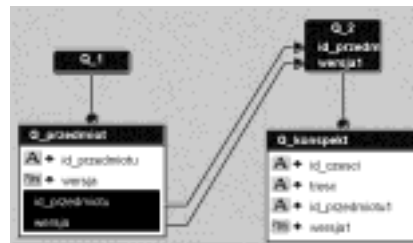
- Wyprowadzanie wyników w XML
- Duże możliwości wpływania na budowę XML

Ustawienia dla XML

- Parametry raportu
 - DESTYPE=FILE
 - DESNAME=nazwa.xml
 - DESFORMAT=XML
- Właściwości raportu
 - XML Tag – znacznik elementu głównego
 - XML Prolog Type, XML Prolog Value – ustawienia prologu (może być z pliku)
- Właściwości grup
 - Outer XML Tag – znacznik grupy
 - Outer XML Attributes – atrybuty grupy (lista par: nazwa="&obiekt")
- Właściwości grup/kolumn
 - XML Tag – znacznik wystąpienia grupy / kolumny
 - XML Tag Attributes – atrybuty wystąpienia grupy / kolumny
 - Exclude from XML Output – nie tworzyć elementu
 - Contains XML Tags – nie zamieniać znaków specjalnych na encje

Przykład

```
<?xml version="1.0" encoding="windows-1250"?>
<!-- Generated by Oracle Reports version
6.0.8.11.3 -->
<eres_konspekty>
  <przedmiot id="KBD2" wersja="1">
    <konspekt>
      <czesc_konspektu id="Streszczenie">
        <tresc><P>Monograficzny przedmiot
poświęcony bazie danych i narzędziom
Oracle.</P></tresc>
      </czesc_konspektu>
    </konspekt>
  </przedmiot>
  .....
</eres_konspekty>
```



Czy już warto używać XML?

59

XML w innych produktach Oracle

Oracle JDeveloper 3.1

- Możliwość wykorzystania
 - Oracle XDK for Java
 - Oracle XML Transviewer Beans
 - XSU i XSQL Servlet.
- Model komponentowy BC4J
 - używa wewnętrznie języka XML do przechowywania metadanych, takich jak reguły przetwarzania (business rules)
- Komponent Web Bean
 - który może być używany w servletach i aplikacjach typu JSP.
 - czyta dane z komponentów typu BC4J i przekształca je na XML
 - może używać XSLT do transformacji wyjściowego dokumentu

Oracle Internet File System (iFS)

- Używa XML do celów konfiguracyjnych
 - tworzenie folderów, użytkowników itp.
 - rozszerzanie listy atrybutów plików (content management)
- Może automatycznie analizować zawartość ładowanych dokumentów XML
 - można wykorzystać wbudowany parser
 - struktura dokumentu musi być zarejestrowana w iFS (też za pomocą XML)
 - dane są przechowywane w
 - tabelach standardowych iFS
 - specjalnie utworzonych tabelach
 - do odtworzenia dokumentu trzeba napisać odpowiedni renderer

```
<?xml version = '1.0' encoding = 'UTF8' standalone = 'yes'?>
<FOLDER>
  <Name>guest</Name>
  <Description>guest's home folder</Description>
  <Owner RefType="Name" Classname="DIRECTORYUSER">guest<!--ID # 1060--></Owner>
  <ACL RefType="Name" Classname="SYSTEMACCESSCONTROLLIST">Public<!--ID # 423--></ACL>
  <CreateDate format="MMM dd HH:mm">mar 17 13:50</CreateDate>
  <Creator RefType="Name" Classname="DIRECTORYUSER">system<!--ID # 96--></Creator>
  <LastModifyDate format="MMM dd HH:mm">mar 17 13:50</LastModifyDate>
  <LastModifier RefType="Name" Classname="DIRECTORYUSER">system<!--ID # 96--></LastModifier>
  <LockState>0</LockState>
</FOLDER>
```



Czy już warto używać XML?

60

- Przypomnienie: podstawy XML
- Przetwarzanie dokumentów w XML
- Narzędzia dla XML
- **Zastosowania XML**
- Podsumowanie



Zastosowania XML

Specjalistyczne struktury danych

- Motywacja
 - tworzenie specyficznych formatów dziedzinowych
 - umożliwienie łatwej wymiany danych
- Stan obecny
 - bardzo wiele propozycji
 - w większości dziedzin jeszcze brak powszechnie uznanych standardów
- Przykłady
 - zastosowania naukowe, np.
 - ♦ MathML (*Mathematical Markup Language*)
 - ♦ CML (*Chemical Markup Language*)
 - modelowanie systemów, np.
 - ♦ PIF-XML (*Process Interchange Format*)
 - ♦ UXF (*UML eXchange Format*)
 - ♦ XML (*XML Metadata Interchange*)

Komunikacja i multimedia

- WML (*Wireless Markup Language*)
 - język opisu stron dla usług internetowych świadczonych z użyciem telefonii komórkowej
 - przeznaczony do pracy z protokołem WAP
- Multimedia — przykłady
 - SVG (*Scalable Vector Graphics*)
 - ♦ standard proponowany przez W3C
 - ♦ scala kilka propozycji specyfikacji grafiki wektorowej
 - SMIL (*Synchronized Multimedia Integration Language*)



EDI i e-commerce

EDI (*Electronic Data Interchange*)

- XML wydaje się idealnym narzędziem dla elektronicznej wymiany danych
 - czytelny
 - prosty
 - łatwy w przetwarzaniu
 - powszechnie używany
- Koncepcja XML/EDI:
 - semantyka istniejących standardów EDI
 - składnia XML
- Wiele propozycji standaryzacyjnych

XML a handel elektroniczny

- Bardzo dobre narzędzie do wymiany danych w handlu elektronicznym
 - proste i łatwe do wykorzystania (darmowe oprogramowanie)
 - wspierane przez wielkich producentów systemów ERP
 - odpowiednie do użycia także przez małe firmy
- Zastosowania B2C i B2B



EDI – przykład

Przykład

- Projekt komunikatów XML do gromadzenia danych na temat ochrony zdrowia
 - dane o lekach refundowanych
 - szpitalne statystyczne karty choroby
 - RUM i jego rozszerzenia
- Pierwszy komunikat już wprowadzony rozporządzeniem Ministra Zdrowia

```
<?xml version="1.0" encoding="windows-1250"?>
<!DOCTYPE mz:komunikat SYSTEM "LEK.dtd" >
<mz:komunikat xmlns:mz="http://www.mz.gov.pl/csioz/START/XML" typ="LEK" wersja="1.00" >
  <mz:dokument id="xxxIDAXPRS" nr="xxxIDAIRRS" tryb="C" data="xxxIDAWURS" >
    <mz:nadawca >
      <mz:podmiot typ="0" symbol="xxxIDA2CVS" />
    </mz:nadawca>
    <mz:odbiorca >
      <mz:podmiot typ="0" symbol="xxxIDA2CVS" />
    </mz:odbiorca>
    <mz: sprawozdanie symbol="xxxIDAV2RS" data="xxxIDA53RS" >
      <mz:okres typ="xxxIDA0ASS" rok="xxxIDAICSS" nr="xxxIDASDSS"
        data-od="xxxIDA2ESS" data-do="xxxIDAKGSS" />
      <mz:komorka-org regon="xxxIDA1EVS" nr="000" />
    </mz: sprawozdanie>
    <mz:pozycja id="xxxIDAGNSS" tryb="D" >
      <mz:swiadczenie typ="0" >
        <mz:zlecenie data="xxxIDADYSS" >
          .....
        </mz:zlecenie>
      </mz:swiadczenie>
    </mz:pozycja>
  </mz:dokument>
</mz:komunikat>
```



XML w systemach z bazami danych

Zastosowania

- Reprezentacja danych poza bazą danych
 - łatwy zapis złożonych danych
 - zastosowanie do
 - ♦ przesyłania danych
 - ♦ prezentacji w WWW
- Reprezentacja danych w bazie danych
 - przydatna dla danych o strukturze zmiennej i/lub złożonej
 - zapis w formie CLOB zamiast BLOB
 - możliwości
 - ♦ przetwarzanie w bazie danych
 - ♦ wyszukiwanie kontekstowe
 - przykładowe zastosowania
 - ♦ złożone dokumenty tekstowe
 - ♦ arkusze kalkulacyjne

Zapis dokumentu XML w bazie danych

- Możliwości reprezentacji w bazie relacyjnej
 - reprezentacja generyczna
 - ♦ elastyczna
 - ♦ nieczytelna
 - ♦ nieefektywna
 - częściowa strukturalizacja
 - ♦ duże możliwości przetwarzania
 - ♦ ograniczona elastyczność
 - zapis czysto tekstowy
 - ♦ bardzo elastyczny
 - ♦ trudny do przetwarzania
 - wykorzystanie cech relacyjno-objektowych
- Przykład
 - Konspekty przedmiotów w systemie wspomagania dziekanatu ERES2
 - ♦ częściowa strukturalizacja
 - ♦ import z XML do bazy danych
 - ♦ prezentacja z bazy danych w HTML



XML w analizie systemów informacyjnych

Motywacja

- Dotyczy wczesnych etapów analizy
- Brak narzędzi, np. typu CASE
- Potrzebne modele
 - na ogół bazują na opisie słownym
 - ale powinny być możliwie sformalizowane
- Potrzebne różne formy prezentacji wyników
 - tabele
 - zestawienia
 - różne przekroje tych samych informacji
- Potrzebne eleganckie formatowanie
 - raporty drukowane
 - strony WWW

Typowe modele

- Planowanie analizy
 - analiza wymagań
 - plan działań i podział zadań
 - projekt raportu
- Słowniki pojęć, skrótów, instytucji i osób
- Wykazy spotkań i notatki z nich (*minutes*)

Budowanie modeli

- Tworzenie specjalizowanych struktur w XML
- Wielokrotne użycie
 - struktur z poprzednich projektów
 - struktur „uniwersalnych”
 - typowych elementów raportów



XML w analizie s.i. – przykłady

- Planowanie zadań analizy systemów informacyjnych
- Analiza procesów biznesowych do celów eksploracji danych (*data mining*)
 - sformalizowany spis procesów
 - spis i ocena zasobów danych
 - zależności między danymi a procesami
 - ocena „podatności” procesów na eksplorację danych
 - zautomatyzowane tworzenie rankingów
- Analizy wymagań funkcjonalnych i techniczno-organizacyjnych dla internetowych systemów informacyjnych
 - sformalizowany wykaz wymagań
 - warianty i klasyfikacje
 - projekty
 - systemu promocji eksportu
 - sieci innowacyjnej w dziedzinie nowych technologii
 - portalu samorządu gospodarczego



- **Przypomnienie: podstawy XML**
- **Przetwarzanie dokumentów w XML**
- **Narzędzia dla XML**
- **Zastosowania XML**
- **Podsumowanie**



Podsumowanie

- * XML zdobył duże i powszechne uznanie
- * Zestaw podstawowych specyfikacji jest niemal ukończony i już nadaje się do zastosowania
- * Język ma silne poparcie najbardziej znaczących producentów oprogramowania, w tym Oracle
- * XML już jest ważnym standardem zapisu i przekazywania informacji, a jego przydatność potwierdziła się
- * Dostępne są liczne narzędzia umożliwiające stosowanie XML, także narzędzia Oracle
- * Stosowanie XML w najbliższych latach stanie się zapewne powszechne
- * XML odegra szczególną rolę w rozwoju aplikacji Inter- i intranetowych, zwłaszcza typu *e-commerce*

Wnioski

- ⇒ Warto już używać XML
- ⇒ **Więcej: najwyższa pora zacząć używać XML**

