XSLT – część 2

Rodzaje przetwarzania XSLT (1)

- Przetwarzanie sterowane strukturą dokumentu źródłowego (ang. *push*):
  - przechodzimy po strukturze dokumentu źródłowego,
  - generujemy fragmenty struktury dokumentu wyjściowego.

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


Rodzaje przetwarzania XSLT (2)

- Przetwarzanie sterowane strukturą dokumentu wyjściowego (ang. *pull*):
  - jedna duża reguła dla węzła *root*,
  - generujemy strukturę dokumentu docelowego,
  - wyciągamy odpowiednie wartości z dokumentu źródłowego.

```
<xsl:stylesheet version="1.0"
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     xmlns="http://www.w3.org/TR/xhtml1/strict">
<xsl:template match="/">
  <html><head><title>Expense Report Summary</title></head>
    <body>
      <h1>Company: <xsl:value-of select="company/name"/></h1>
      <p>Total Amount:
      <xsl:value-of select="expense-report/total"/></p>
    </body>
  </html>
</xsl:template>
</xsl:stylesheet>
```


Uproszczone przekształcenia

- Tylko jeden wzorzec dla węzła *root*.
- Pomijamy element `stylesheet`.

```
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xmlns="http://www.w3.org/TR/xhtml1/strict"
      xsl:version="1.0">
  <head><title>Expense Report Summary</title></head>
  <body>
    <h1>Company:
    <xsl:value-of select="company/name"/></h1>
    <p>Total Amount:
    <xsl:value-of select="expense-report/total"/></p>
  </body>
</html>
```


Zaawansowane możliwości XSLT

- Sortowanie węzłów.
- Tryby przetwarzania (*modes*):
  - przełączanie między trybami,
  - niezależnie definiowane wzorce dla każdego trybu.
- Zmienne.
- Wzorce nazwane:
  - wywoływane jak podprogramy (procedury),
  - przekazywanie parametrów,
  - rekursja.

Sortowanie

- Można stosować w:
  - `apply-templates`,
  - `for-each`.

```
<xsl:template match="miasta">
  <h1>Miasta wg liczby mieszkańców</h1>
  <xsl:apply-templates select="miasto">
    <xsl:sort select="liczba-mieszk"
              order="descending"/>
  </xsl:apply-templates>
</xsl:template>
```

Tryby przetwarzania (*modes*)

```
<xsl:template match="/">
  <h1><xsl:value-of select="book/title"/></h1>
  <h2>Spis treści</h2>
  <xsl:apply-templates mode="toc"/>
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="chapter" mode="toc">
  <p><a href="#{generate-id()}">
  <xsl:value-of select="title"/></a></p>
</xsl:template>

<xsl:template match="chapter">
  <h2><a name="{generate-id()}">
  <xsl:value-of select="title"/></a></h2>
  <xsl:apply-templates/>
</xsl:template>
```


Zmienne

- Jak w funkcyjnych językach programowania:
  - brak instrukcji przypisania,
  - brak efektów ubocznych.
- Deklaracja:
  - `<xsl:variable name="..."/>`
  - wartość:
    - atrybut `select` → wyrażenie odpowiedniego typu,
    - zawartość elementu → fragment drzewa wynikowego,
- Użycie:
  - w wyrażeniach: *$name*,
  - `<xsl:copy-of select="`*expression*`"/>`

Wykorzystanie rekursji w XSLT (1)

- Sposób na brak „prawdziwych” zmiennych i pętli iteracyjnych.
- Przykład – suma wartości książek:

```
<books>
  <book>
    <title>Pan Tadeusz</title>
    <qty>12</qty><price>10.99</price>
  </book>
  <book>
    <title>Mistrz i Małgorzata</title>
    <qty>1</qty><price>15.99</price>
  </book>
  <book>
    <title>Imię Róży</title>
    <qty>2</qty><price>6.99</price>
  </book>
</books>
```


Wykorzystanie rekursji w XSLT (2)

- Przykład – XSLT:

```
<xsl:template name="total-val">
  <xsl:param name="list"/>
  <xsl:choose>
    <xsl:when test="$list">
      <xsl:variable name="first" select="$list[1] "/>
      <xsl:variable name="rest">
        <xsl:call-template name="total-val">
          <xsl:with-param name="list"
            select="$list[position()!=1] "/>
        </xsl:call-template>
      </xsl:variable>
      <xsl:value-of select="$first/qty * $first/price
                            + $rest"/>
    </xsl:when>
    <xsl:otherwise>0</xsl:otherwise>
  </xsl:choose>
</xsl:template>
```


Wykorzystanie rekursji w XSLT (3)

- Przykład – XSLT, c.d.:

```
<xsl:template match="/">
  <xsl:variable name="total">
    <xsl:call-template name="total-val">
      <xsl:with-param name="list" select="//book"/>
    </xsl:call-template>
  </xsl:variable>

  <p>Total sales: <xsl:value-of select="$total"/>
  </p>
</xsl:template>
```


Ograniczona użyteczność zmiennych

- Specjalny typ danych: fragment drzewa wynikowego (*result tree fragment*):
  - wartość można jedynie:
    - skopiować do drzewa wynikowego lub innej zmiennej,
    - przekształcić do napisu;
  - nie można (w czystym XSLT) przekształcić na *node set*,
  - praktycznie niemożliwe obliczenia na zmiennych w kilku przebiegach.
- Zmienna typu *node set*:

```
<xsl:variable name="b" select="/books"/>
<xsl:for-each select="$b/book">...</xsl:for-each>
```

- Zmienna typu *result tree fragment*:

```
<xsl:variable name="subtotals">
  <xsl:for-each select="/books/book">
    <subtl><xsl:value-of select="qty * price"/></subtl>
  </xsl:for-each>
</xsl:variable>
```

Funkcja `node-set()`

- Przekształca fragment drzewa wynikowego (*result tree fragment*) w zbiór węzłów (*node set*).
- Niedostępna w „czystym” XSLT:
- Dostępna jako rozszerzenie m. in. w:
  - procesorach: XT, Saxon, MSXML,
  - bibliotece rozszerzeń EXSLT.

```
<xsl:variable name="subtotals">
  <xsl:for-each select="/books/book">
    <subtl><xsl:value-of select="qty * price"/></subtl>
  </xsl:for-each>
</xsl:variable>

<xsl:value-of
  select="sum(exsl:node-set($subtotals)/subtl)"/>
```


Generowanie przekształceń XSL (1)

- Problem:
  - źródło przekształcenia nie zawiera metainformacji o strukturze dokumentu,
  - metainformacje pojawiają się na wyjściu.

```
<wniosek-urlopowy>
 <wniosek>
  <pracownik>Szymon Zioło</pracownik>
  <rodzaj>wypoczynkowy</rodzaj>
  <od>2003-06-20</od>
  <do>2003-06-27</do>
  <dni-roboczych>6</dni-roboczych>
 </wniosek>
 <decyzja>
  <zgoda>1</zgoda>
  <zastępca>Jan Kowalski</zastępca>
 </decyzja>
</wniosek-urlopowy>
```

Wniosek urlopowy

Wniosek

Pracownik: **Szymon Zioło**
Rodzaj urlopu: **wypoczynkowy**
Od dnia: **2003-06-20**
Do dnia: **2003-06-27**
Ilość dni roboczych: **6**

Decyzja przełożonego

Zgoda przełożonego: **tak**
Zastępca: **Jan Kowalski**

Źródło: Zioło, Sz., *XSLT do kwadratu*, Software 2.0, nr 6/2003

Generowanie przekształceń XSL (2)

- Rozwiązanie:
  - zapisanie metainformacji w szablonie,
  - generowanie przekształcenia z szablonu.

```
<dokument nazwa="wniosek-urlopowy"
          etykieta="Wniosek urlopowy">
  <sekcja nazwa="wniosek" etykieta="Wniosek">
    <pole nazwa="pracownik" etykieta="Pracownik:"/>
    <pole nazwa="rodzaj" etykieta="Rodzaj urlopu:"/>
    <pole nazwa="od" etykieta="Od dnia:"/>
    <pole nazwa="do" etykieta="Do dnia:"/>
    <pole nazwa="dni-roboczych"
          etykieta="Ilość dni roboczych:"/>
  </sekcja>
  <sekcja nazwa="decyzja" etykieta="Decyzja przełożonego">
    <pole nazwa="zgoda" etykieta="Zgoda przełożonego:"
          typ="boolean"/>
    <pole nazwa="zastępca" etykieta="Zastępca:"/>
  </sekcja>
</dokument>
```

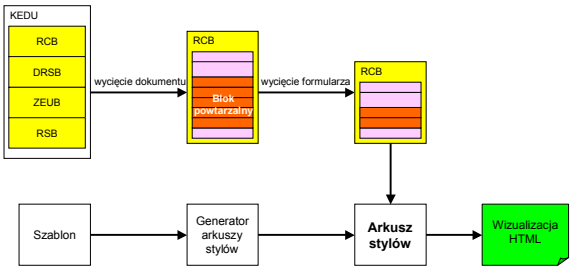

Generator – przykład (1)

```
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:res="http://www.w3.org/1999/XSL/TransformAlias">
  <xsl:namespace-alias stylesheet-prefix="res"
                       result-prefix="xsl"/>

  <xsl:template match="/">
    <res:stylesheet version="1.0">
      <res:output method="html"/>
      <xsl:apply-templates/>
    </res:stylesheet>
  </xsl:template>

  <xsl:template match="sekcja">
    <res:template match="{@nazwa}">
      <p><b><xsl:value-of select="@etykieta"/></b></p>
      <table><res:apply-templates/></table>
    </res:template>
    <xsl:apply-templates/>
  </xsl:template>
```

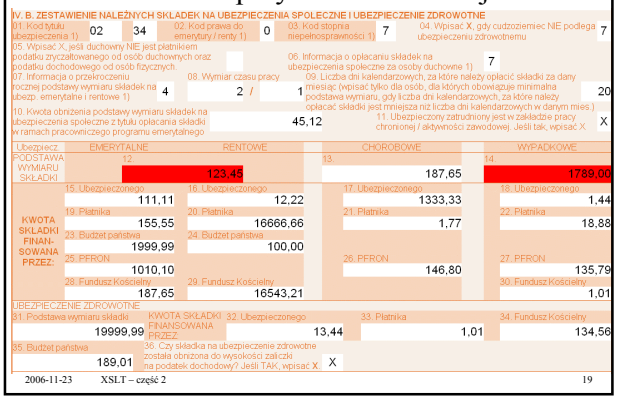

Generator – przykład (2)

```
  <xsl:template match="pole">
    <res:template match="{@nazwa}">
      <tr><td><xsl:value-of select="@etykieta"/></td>
      <td><b>
        <xsl:choose>
          <xsl:when test="@typ='boolean'">
            <res:choose>
              <res:when test="text()='1'">tak</res:when>
              <res:otherwise>nie</res:otherwise>
            </res:choose>
          </xsl:when>
          <xsl:otherwise>
            <res:value-of select="text()"/>
          </xsl:otherwise>
        </xsl:choose>
      </b></td></tr>
    </res:template>
    <xsl:apply-templates/>
  </xsl:template>

</xsl:stylesheet>
```


Zastosowanie w projekcie: KEDU ZUS



KEDU ZUS – przykład wizualizacji



IV. B. ZESTAWIENIE NALEŻNYCH SKŁADEK NA UBEZPIECZENIA SPOŁECZNE I UBEZPIECZENIE ZDROWOTNE

01. Kod tytułu ubezpieczenia 1)	02	34	02. Kod prawa do emerytury / renty 1)	0	03. Kod stopnia niepełnosprawności 1)	7	04. Wpisać X, gdy cudzoziemiec NIE podlega ubezpieczeniu zdrowotnemu	7

05. Wpisać X, jeśli duchowny NIE jest płatnikiem podatku zryczałtowanego od osób duchownych oraz podatku dochodowego od osób fizycznych.

06. Informacja o opłacaniu składek na ubezpieczenia społeczne za osoby duchowne 1) 7

07. Informacja o przekroczeniu rocznej podstawy wymiaru składek na ubezp. emerytalne i rentowe 1) 4

08. Wymiar czasu pracy 2 / 1

09. Liczba dni kalendarzowych, za które należy opłacić składki za dany miesiąc (wpisać tylko dla osób, dla których obowiązuje minimalna podstawa wymiaru, gdy liczba dni kalendarzowych, za które należy opłacać składki jest mniejsza niż liczba dni kalendarzowych w danym mies.) 20

10. Kwota obniżenia podstawy wymiaru składek na ubezpieczenia społeczne z tytułu opłacania składki w ramach pracowniczego programu emerytalnego 45,12

11. Ubezpieczony zatrudniony jest w zakładzie pracy chronionej / aktywności zawodowej. Jeśli tak, wpisać X X

Ubezpiecz.	EMERYTALNE	RENTOWE	CHOROBOWE	WYPADKOWE
PODSTAWA WYMIARU SKŁADKI	12. 123,45		13. 187,65	14. 1789,00
KWOTA SKŁADKI FINANSOWANA PRZEZ:	15. Ubezpieczonego 111,11	16. Ubezpieczonego 12,22	17. Ubezpieczonego 1333,33	18. Ubezpieczonego 1,44
	19. Płatnika 155,55	20. Płatnika 16666,66	21. Płatnika 1,77	22. Płatnika 18,88
	23. Budżet państwa 1999,99	24. Budżet państwa 100,00		
	25. PFRON 1010,10		26. PFRON 146,80	27. PFRON 135,79
	28. Fundusz Kościelny 187,65	29. Fundusz Kościelny 16543,21		30. Fundusz Kościelny 1,01

UBEZPIECZENIE ZDROWOTNE

31. Podstawa wymiaru składki	KWOTA SKŁADKI FINANSOWANA PRZEZ:	32. Ubezpieczonego	33. Płatnika	34. Fundusz Kościelny
19999,99		13,44	1,01	134,56

35. Budżet państwa 189,01

36. Czy składka na ubezpieczenie zdrowotne została obniżona do wysokości zaliczki na podatek dochodowy? Jeśli TAK, wpisać X X

Formatting Objects – przykład drzewa wynikowego

```
<?xml version="1.0" encoding="utf-8"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
...
  <fo:page-sequence>
    <fo:flow>
      <fo:block font-size="18pt" font-weight="bold"
       text-align="centered">Preface</fo:block>
      <fo:block font-size="12pt" space-before="1pc"
       text-align="justified"> This is a simple
       test document. It shows a
       <fo:inline font-style="italic">partial</fo:inline>
       fo-result tree (page layout missing).</fo:block>
    </fo:flow>
  </fo:page-sequence>
...
</fo:root>
```


Formatting Objects – przykłady reguł

```
<xsl:template match="chapter">
  <fo:flow><xsl:apply-templates/></fo:flow>
</xsl:template>

<xsl:template match="chapter/title">
  <fo:block font-size="18pt" font-weight="bold"
            text-align="centered">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:template match="para">
  <fo:block font-size="12pt" space-before="1pc"
            text-align="justified">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>

<xsl:template match="emphasis">
  <fo:inline font-style="italic"><xsl:apply-templates/>
  </fo:inline>
</xsl:template>
```


Narzędzia

- Procesory XSLT:
  - XT, James Clark (Java),
  - Oracle XML Parser for Java / C / PL-SQL,
  - Xalan, Apache (Java, C++),
  - SAXON, Michael H. Kay (Java; implementuje XSLT 2.0 i XQuery),
  - Sablotron (C++, open source),
  - Microsoft XML Core Services (MSXML 4.0),
  - XSLTC (XSLT compiler, Java), Apache.
- Procesor XSL:FO:
  - FOP, Apache (Java; generuje dokumenty w formacie PDF).
- Edytory XSLT:
  - XMLSPY, Altova
  - Xselerator XSL Editor/Debugger, MarrowSoft,
  - xslide – Emacs Major Mode for XSL Stylesheets.

Najważniejsze ograniczenia XSLT 1.0

- Brak konwersji fragmentów drzewa wynikowego na pełnoprawne zbiory węzłów.
- Brak możliwości generowania wielu dokumentów wyjściowych.
- Brak wsparcia dla grupowania węzłów.
- Brak możliwości definiowania własnych funkcji.

Gdzie szukać dalej

- EXSLT
  - www.exslt.org/
- Kosek, J., *Understanding the node-set() Function*
  - www.xml.com/pub/a/2003/07/16/nodeset.html
- Tyszko, S., *Rekurencyjne szablony w XSLT*
  - Software 2.0, nr 6/2002, Wydawnictwo Software
- Zioło, Sz., *XSLT do kwadratu*
  - Software 2.0, nr 6/2003, Wydawnictwo Software