Dlaczego Python jest lepszy od XSLT?

Jarek Miszczak

10 lutego 2005 roku

Streszczenie

Celem artykułu jest przyjrzenie się językowi XSLT jako uniwersalnemu językowi programowania i porównananie go z językiem Python. Głównym, ale nie jedynym obszarem będzie zastosowanie XSLT oraz Pythona do przetwarzania dokumentów XML. Uwagi dotyczące Pythona mają zastosowanie do większości języków skryptowych. Postaram się przedstawić możliwości i ograniczenia XSL. Okazuje się jednak, że pomimo jego wad jest on niezastąpiony w wielu przypadkach.

1 Wstęp

XSLT (*ang.* XSL Transformations)[2] jest *jedynym ustandaryzowanym przez W3C językiem programowania przeznaczonym do przetwarzania dokumentów XML*[1]. Jest on pierwszą częścią specyfikacji rozszerzalnego języka stylów XSL (*ang.* eXtensible Stylesheet Language) – drugą częścią jest XSL-FO (*ang.* XSL Formating Objects).

XSLT jest językiem zapisanym w składni XML-owej – tekst programu XSLT musi być poprawnym domumentem XML. W obrębie tego dokumentu mogą znajdować się:

– elementy z przestrzeni nazw `http://www.w3.org/1999/XSL/Transform` odpowiedzialne za sterowanie parserem (następujące bezpośrednio po `xsl:transform`) oraz instrukcje (potomków elementu `xsl:template`),

– elementy z innych przestrzeni nazw pozwalające wstawiać dosłownie zawartość do dokumentu wynikowego np. elementy HTML z przestrzeni nazw `http://www.w3.org/TR/REC-html40`,

– elementy rozszerzeń definiowanych przez procesory XSLT (np. instrukcje do obsługi baz danych specyficzne dla procesora Saxon[2] czy udostępniane przez zestaw rozszerzeń EXSLT).

Natomiast jeżeli chodzi o Pythona i inne języki skryptowe to ich głównymi cechami są:

- wysokopoziomowe typy i struktury danych,
- dynamiczna kontrola typów,
- otwarta implementacja.

Przykłady to PHP, Perl, Python, Pike... Są to języki ogólnego przeznaczenia, najczęściej z elementami obiektowości, modułami/pakietami, obsługą wyjątków.

[1] Inne języki przeznaczone do transformacji XML/SGML to FOSI czy DSSSL. XSLT ma w sobie dużo z przypominającego LISP DSSSL

[2] http://saxon.sourceforge.net/

2 Programowanie

XSLT jest prostym językiem należącym do rodziny języków funkcyjnych. Został on zaprojektowany na potrzeby XML-a i ma dość ograniczone możliwości operowania na danych które nie są zapisane w postaci XML-a. Pyton jest natomiast językiem imperatywnym, bardzo elastycznym, dającym się zastosować niemal wszędzie, także do tworzenia/obrabiania dokumentów XML. Przykładem takiego zastosowania jest parser `xmlproc`.

2.1 Implementacja

Pisząc transformację w XSLT nie musimy się martwić o szczegóły związane z otwieraniem pliku, chodzeniem po drzewie DOM czy tworzeniem nowych elementów tego drzewa. Takimi szczegółami zajmuje się procesor języka – pełni on rolę kompilatora i interpretera języka. Oczywiście uzależnia to programistę od możliwości procesora, szczególnie gdy sięgamy po rozszerzenia specyficzne dla konkretnej implementacji. Co więcej procesor wykonuje za nas część pracy stosując domyślne wzorce służące np. do przetwarzania (a w zasadzie usuwania) komentarzy i instrukcji przetwarzania:

```
<xsl:template match="processing-instruction()|comment()"/>
```

czy zajmując się wyszukiwaniem elementów pasujących do wzorców.

Natomiast w przypadku Pythona to na programiście leży zadanie tworzenia wynikowego dokumentu, wykonania transformacji i zapisania wyniku. Trzeba przyznać, że zadanie to jest bardzo ułatwiane przez moduły standardowe. XML może być przetwarzany zdarzeniowo z wykorzystaniem interfejsu SAX lub jeżeli potrzebujemy możliwości zmiany/tworzenia elementów drzewa przy pomocy interfejsu DOM.

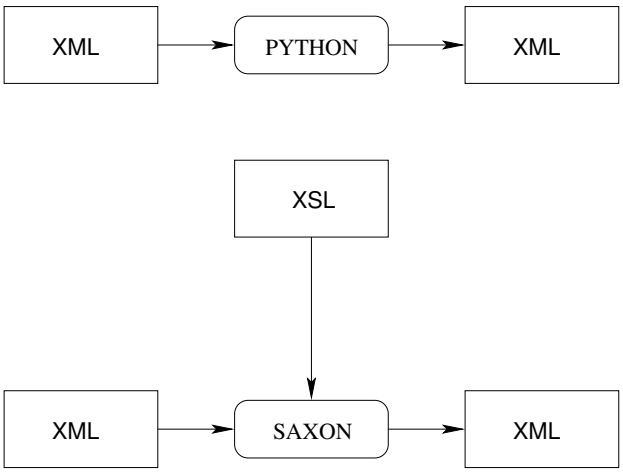
```
import sys
from xml.dom.ext import *
from xml.dom.ext.reader import Sax2

# zbuduj dokument na podstawie zdarzeń
# czytając ze standardowego wejścia
reader = Sax2.Reader()
doc = reader.fromStream(sys.stdin)

# dodaj nowy element
new = doc.createElement('Kot')
new.setAttribute('wiek', '3')
doc.documentElement.appendChild(new)

# wypisz na standardowe wyjście
PrettyPrint(doc,sys.stdout)
```

Oczywiście można przy tym wykorzystać ogromne możliwości tego języka i przeskoczyć tak elementarne ograniczenia XSLT jak brak możliwości zmiany wartości zmiennej. Także w przypadku pracy z czystym tekstem lub TEX-em Python jest wygodniejszy, gdyż daje większą kontrolę na postacią wynikową pliku. Python może zostać łatwo zastosowany do implementacji języków programowania. Dodatkowo nie musimy się obawiać uzależnienia od jakiejś szczególnej funkcji udostępnianej przez konkretny interpreter – mamy jedną, spójną, otwartą implementację.



Rysunek 1: Przetwarzanie XML-a w Pythonie i przy wykorzystaniu XSLT

Zaletą, która jest wspólna dla obu tych języków, jest przejrzystość ich składni. Reguły XML wymuszają na programiście pisanie czytelnego kodu XSLT. Jeszcze bardziej jest to widoczne w przypadku Pythona, gdzie język wymusza jeden styl robienia wcięć i nie dopuszcza swobody charakterystycznej np. dla Perla.

Największą zaletą XSLT jest jego prostota i możliwość łatwego zaimplementowania jako części składowej większego systemu. Wynika to z powszechnej obecności XML-a. Procesor XSLT może być częścią przeglądarki internetowej lub zostać łatwo użyty przy generowaniu stron HTML z wykorzystaniem PHP. Jednym słowem jest on możliwy do wykorzystania niemal wszędzie, szczególnie tam gdzie nie jest potrzebna cała gama możliwości dostarczanych przez Pythona.

2.2 Biblioteki

Jedną z typowych cech programistów jest zamiłowanie do wykorzystywania pracy innych. Wraz z dystrybujcą Pythona dostajemy bogatą bibliotekę modułów, a dodatkowo istnieje wiele projektów rozszerzających możliwości języka o możliwość budowania GUI. Do pracy z XML-em możemy wykorzystać w zależnosci od potrzeby moduły implementujące interfejsy DOM oraz SAX i dzięki temu korzystać ze wszystkich ich możliwości. Jeżeli to nam nie wystarcza to dzięki modułowi `xml.xpath` możemy do poruszania się po drzewie XML-woym wykorzystać język XPath.

W przypadku XSLT standardowy język jest dość ubogi, natomiast poszczególne implementacje często dodają własne rozszerzenia. Oczywiście wykorzystanie ich powoduje uzależnienie się od konkretnej implementacji i stworzony w ten sposób kod jest mniej przenośny. Istnieje także kilka projektów stworzenia zestawu funkcji XSLT które są często wykorzystywane w pracy z tym językiem.

Najpopularniejszy zestaw rozszerzeń dla XSLT 1.0 to EXSLT[3]. Przykłady rozszerzeń zdefiniowanych w EXSLT i udostępnianych między innymi przez parser Saxon to funkcje operacji na zbiorach `difference()` czy `intersection()`, funkcje do manipulacji datą `day-in-year()`, `day-in-month()` czy przydatne podstawowe funkcje matematyczne, takie jak `cos()`, `sin()` czy `sqrt()`.

Podobnym projektem jest XSLT Standard Library[4]. Aby z niej skorzystać wystarczy w swoim pliku stylu wpisać linijkę

[3] http://www.exslt.org/

[4] http://xsltsl.sourceforge.net/.

```
<xsl:import href="http://xsltsl.sourceforge.net/modules/stdlib.xsl"/>
```

W szkicu rekomendacji W3C dla XSLT 2.0 wiele z rozszerzeń EXSLT znalazło swoje miejsce i mogą być wykorzystywane w parserach wspierających XSLT 2.0 oraz XPath 2.0, na przykład w Saxonie 8.

2.3 Funkcje i rekurencja

XSLT jak każdy język funkcyjny jest nastawiony na wykorzystanie rekurencji. Odpowiednikami funkcji z języków imperatywnych są w XSLT wzorce (*ang.* templates). Element `xsl:template` może mieć atrybut `name`, który pozwala na odwołanie się do niego. Wzorce mogą być wywołane z parametrami. Przykładowo

```
<template match="/">
<xsl:call-template name="header">
  <xsl:with-param name="name">Autor Tekstu</xsl:with-param>
  <xsl:with-param name="e-mail">ja_sam@moja.domena</xsl:with-param>
</xsl:call-template>

<xsl:template name="header">
  <xsl:element name="h1">
    <xsl:param name="name"/>
  </xsl:element>
  <xsl:element name="h2"
    <xsl:param name="e-mail"/>
  </xsl:element>
</xsl:template>
```

Rekurencja daje też możliwość poradzenia sobie z prostym zagadnieniem poszatkowania wartości elementu w postaci listy $x_1, x_2, \ldots, x_k$.

```
<xsl:template name="split">
  <xsl:param name="napis"/>
  <xsl:choose>

    <xsl:when test="contains($napis,',')">
      <xsl:variable name="tmp" select="substring-before($napis,',')"/>
      <xsl:value-of select="$tmp"/>
      <xsl:call-template name="split">
      <xsl:with-param name="napis" select="substring-after($napis,',')"/>
      </xsl:call-template>
    </xsl:when>

    <xsl:otherwise>
      <xsl:value-of select="$napis"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

Przetwarzanie napisów w Pythonie jest realizowane przez moduł standardowy `string` i sprawę w tym wypadku załatwia funkcja `split`.

Funkcja o tej nazwie jest także dostępna jako rozszerzenie EXSLT, ale zgodnie z informacją na stronie EXSLT, jest ona implementowana tylko przez parser 4Suite[5].

[5] http://4suite.org/

```
  <xsl:template match="/">
    <xsl:variable name="napis" select="/element"/>
    <xsl:for-each select="str:split($napis, ',')">
      <xsl:value-of select="."/>
    </xsl:for-each>
  </xsl:template>
```

2.4 Iteracja

Jedyną konstrukcją iteracyjną XSLT 1.0 jest pętla `xsl:for-each`. Jednak język ten nie pozwala na zmianę wartości zmiennej podczas wykonania, zatem nie jest możliwe zadeklarowanie licznika pętli i konieczne jest użycie rekurencji. Chyba że zastosuje się sztuczkę, w której bazujemy na zbiorze elementów w dokumencie XML.

```
<xsl:template match="/">
  <xsl:variable name="val" select="/tabela/@ile"/>
    <table>
      <xsl:for-each select="(document('')//*)[position() &lt; $val]">
        <tr><td>Wiersz <xsl:value-of select="position()"/></td></tr>
      </xsl:for-each>
    </table>
</xsl:template>
```

Wyrażenie w artybucie `select` instrukcji `for-each` nakazuje wykonanie pętli po wszystkich elementach arkusza `document('')//*` mających pozycję mniejszą od narzuconej wartości `[position() < $val]`. Oczywiście istnieje przy tym możliwość wyczerpania elementów i przerwania pętli.

W XSLT 2.0 zadanie to jest uproszczone poprzez wykorzystanie konstrukcji

```
<xsl:variable name="seq" as="xs:integer*">
  <xsl:for-each select="1 to 3">
    <xsl:sequence select=".*2"/>
  </xsl:for-each>
</xsl:variable>
```

która wykorzystuje możliwości XPath 2.0.

2.5 Funkcje wyższego rzędu

Jedną z ciekawych cech funkcyjnych jest możliwość operowania na funkcjach – funkcja może pobierać jako argument lub zwracać jako wynik inną funkcję. W przypadku XSLT odpowiada to przetwarzaniu/generowaniu dokumentu XML zawierającego elementy z przestrzeni nazw `http://www.w3.org/1999/XSL/Transform`. Możemy łatwo napisać program, który pisze program w oparciu o dane. Pozwala na to mechanizm aliasów dla przestrzeni nazw. Po umieszczeniu w dokumencie XSLT poniższego fragmentu

```
<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="mpn"
  result-prefix="xsl"/>
```

```
  <!--
     treść programu
  -->

</xsl:stylesheet>
```

wszystkie elementy w arkuszu zadeklarowane w przestrzeni nazw `mpn` będą w dokumencie wynikowym przeniesione do przestrzeni nazw `xsl`.

3 Przykłady

3.1 Własna mini-bibliografia

Najczęściej XSL jest wykorzystywany do generowania rezultatu w postaci kodu HTML. Powiedzmy, że chcemy stworzyć sobie spis książek w podręcznej bibliotece. Interesuje nas autor, tytuł i streszczenie. Oczywiście zawsze można stworzyć do takiego celu bazę danych, ale XML daje nam *de facto* możliwość stworzenia własnego języka do tego celu. Weźmy wpis w postaci

```
<Ksiazka>
  <Autor>Kubuś Puchatek</Autor>
  <Tytul>Linux dla pluszaków</Tytul>
  <Wydane>Stumilowy Las</Wydane>
  <Rok>2005</Rok>
</Ksiazka>
```

W połączeniu z poniższym arkuszem

```
<xsl:template match="Ksiazka">
  <!-- dodajemy element po elemencie -->
  <xsl:element name="div">
    <xsl:attribute name="class">author</xsl:attribute>
    <xsl:for-each select="./Autor">
      <xsl:value-of select="."/>
      <xsl:text>, </xsl:text>
    </xsl:for-each>
  </xsl:element>

  <!-- trochę krócej -->
  <div class="title">
    <xsl:value-of select="./Tytul"/>
    <xsl:text>, </xsl:text>
  </div>

  <xsl:element name="div">
    <xsl:attribute name="class">wydanie</xsl:attribute>
      <xsl:value-of select="./Wydanie"/>
      <xsl:text>, </xsl:text>
      <xsl:value-of select="./Rok"/>
    <xsl:text>.</xsl:text>
  </xsl:element>
</xsl:template>
```

otrzymujemy fragment kodu HTML. Do wyświetlenia wyniku możemy wykorzystać PHP i przeglądarkę WWW. Transformację wykonuje w tym wypadku wbudowany w PHP procesor.

```
function xml2xml($xml, $xsl, $encoding){

  $xsltproc = xslt_create();
  xslt_set_encoding($xsltproc, $encoding);
  $output = xslt_process($xsltproc, $xml, $xsl);

  if (empty($output)) {
    die('XSLT processing error: '. xslt_error($xsltproc));
  }

  xslt_free($xsltproc);
  return $output;
}
```

Przy odrobinie wysiłku możemy napisać styl, który skonwertuje nasz zbiór do formatu BibTEX-a albo przeniesie do SQL, ale wówczas musimy liczyć się z pojawieniem się trudności związanych z kontrolowaniem wyniku, zwłaszcza jeżeli zależy nam na jego czytelności. Do takiego zadania Python czy Perl nadają się znacznie lepiej.

3.2 Operacje na zbiorach

Abstrahując od konkretnego zadania zobaczmy jak XSLT radzi sobie z matematyką, a dokładniej ze zbiorami. W standardowym XSLT 1.0 dostępna jest operacja na zbiorach elementów (*ang.* nodes) – jest nią sumowanie i jest ona reprezentowana operatorem `|` (pionowa kreska). EXSLT udostępnia funkcje takie jak `set:intersection()` pozwalające na bardziej zaawansowane operowanie na zbiorach.

Najlepiej zobaczyć to na przykładzie. Jako przykład posłuży nam dokument

```
<?xml version="1.0" encoding="UTF-8"?>
<set>
  <item>1</item><item>2</item>
  <item>3</item><item>4</item>
  <item>5</item><item>6</item>
</set>
```

zawierający zbiór elementów `<item>`.

Poniższy arkusz stosuje elementy EXSLT do wyznaczenia przekroju i różnicy wybranych podzbiorów.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:set="http://exslt.org/sets"
  extension-element-prefixes="set">
  <xsl:output method="xml" indent="yes"/>
  <xsl:template match="/">
    <!-- zdefinuj dwa podzbiory -->
    <xsl:variable name="subSet1" select="/set/item[position() &gt; 1]"/>
    <xsl:variable name="subSet2" select="/set/item[position() &lt; 5]"/>
```

```
    <!-- ich przekrój -->
    <xsl:text>Przekrój: </xsl:text>
    <xsl:for-each select="set:intersection($subSet1,$subSet2)">
      <xsl:value-of select="."/>
    </xsl:for-each>

    <!-- roznica subSet1 - subSet2-->
    <xsl:text>Różnica: </xsl:text>
    <xsl:for-each select="set:difference($subSet1,$subSet2)">
      <xsl:value-of select="."/>
    </xsl:for-each>

  </xsl:template>
</xsl:stylesheet>
```

Jednak, podobnie jak w wielu innych wypadkach, można się obejść bez dodatkowych funkcji. Standardowa funkcja `count()` zwraca ilość elementów w zbiorze.

```
    <!-- elementy ktorych dodanie nie zmienia zbioru -->
    <xsl:for-each select="$subSet1[count(.|$subSet2) = count($subSet2)]">
      <xsl:text>Intersection: </xsl:text>
      <xsl:value-of select="."/>
      <xsl:if test="position() != last()">, </xsl:if>
    </xsl:for-each>

    <!-- elementy zbioru  subSet1 ktore powiększają zbior subSet2 -->
    <xsl:for-each select="( $subSet1[count(.|$subSet2) != count($subSet2)]">
      <xsl:text>Roznica: </xsl:text>
      <xsl:value-of select="."/>
      <xsl:if test="position() != last()">, </xsl:if>
    </xsl:for-each>
  </xsl:template>
```

3.3 Wizualizacja języka programowania

Ciekawym zastosowaniem XML-a jest opisywanie układów bramek kwantowych w języku QML. Język ten powstał wraz z utworzeniem równoległego symulatora układów kwantowych udostępnianego publicznie przez WWW przez fundację Fraunhofera [6]. Opis prostego obwodu jest przedstawiony poniżej

```
<QML>
  <Circuit Name="test1" Id="test1.qml" Size="3"
    Description="Prosty QML">
    <Operation Step="0">
    </Operation>
    <Operation Step="1">
      <Application Name="G" Id="0" Bits="0,1">
        <Gate Type="CNOT"/>
      </Application>
      <Application Name="G" Id="2" Bits="2">
        <Gate Type="HADAMARD"/>
```

[6] http://www.qc.fraunhofer.de/

```
      </Application>
  </Operation>
<QML>
```

Język może być łatwo przekształcony na postać graficzną w SVG.

Za zastosowaniem w takim wypadku Pythona przemawia kilka argumentów. Portal przez który udostępniany jest symulator oparty jest na Plone, zatem zintegrowanie z nim kodu w Pythonie jest naturalne. Sam język jest na tyle rozbudowany, że łatwiej jest sparsować go i przerobić z wykorzystaniem Pythona, niż stosując wyszukane lub niestandardowe elementy XSLT.

3.4 Maszyna Turinga

O tym że XSLT jest rzeczywiście pełnoprawnym językiem programowania można się przekonać uruchamiając Maszynę Turinga napisaną w tym języku [1].

Jeden przykładów kodu maszyny Turinga znajduje się w kolekcji niekonwencjonalnych zastosowań XSLT na stronie [5]. Można tam znaleźć przykłady wykorzystania XSLT, między innymi do generowania trójkąta Sierpińskiego. Maszyna ta potrafi dodawać oraz kodować napisy algorytmem ROT13. Oczywiście jako ciąg wejściowy przyjmuje odpowiednio skonstruowany dokument XML.

4 Narzędzia

XSLT jest językiem prostym, ale przy pracy z dużymi dokumentami przydatne jest oprogramowanie wspierające programistę. W przypadku XSLT najprostszym rozwiązaniem jest instrukcja `xsl:message`, pozwalająca dodać do programu komunikaty diagnostyczne [8]. Pełni ona rolę standardowego modułu Pythona `warnings`. Jeżeli takie podejście nie wystarcza w Pythonie można wykorzystać moduł `pdb` służący do odpluskwiania kodu. Istnieje także wiele środowisk programistycznych dla tego języka – przykłady to Eclipse+PyDev – integrujących się z `pdb`.

W przypadku XSLT dostępnych jest kilka narzędzi realizujących podobne funkcje, ale są to w przeważającej części produkty komercyjne (rozwijane m.in. przez firmy Altova, StylusStudio oraz ActiveState). Pozwalają one na dokładne śledzenie przebiegu transforamcji. Można to wypróbować wykorzystując przeznaczoną dla środowiska Eclipse wtyczkę Oxygen XML Editor. Daje ona możliwość nie tylko edycji kodu XSLT, ale także wykonania go krok po kroku. Można definiować punkty stopu (*ang.* break points), obserwować wartość zmiennych oraz kontekst wykonania instrukcji.

Wśrod narzędzi darmowych najlepsze rozwiązanie daje xslt-process dla XEmacsa. Jest on częścią standardowej dystrybucji XEmacas. Pozwala on na debuggowanie kodu xslt przy współpracy z PSGML lub XSLide – trybami Emacsa do edycji XML/XSL. Współpracuje on z procesorami Saxon i Xalan.

Innym darmowym narzędziem jest XSL Debugger [7]. Jest to narzędzie analogiczne do gdb przeznaczone dla XSLT. Dostępna jest nakładka graficzna KXsltDbg. Możliwości tego narzędzia nie są tak duże jak wtyczki edytora Oxygen czy xslt-process. Niestety do przetwarzania dokumentów wykorzystywana może być jedynie biblioteka libxslt z projektu Gnome.

Natomiast jeżeli chcemy dokładnie prześledzić czas wykonania poszczególnych części naszego programu możemy wykorzystać profiler chatchXSL[8]. Współpracuje on z Saxonem oraz Xalanem. Pozwala na wygenerowania dokładnego raportu wyszczególniającego przebieg wykonanej transformacji.

[7] http://xsldbg.sourceforge.net/

[8] http://www.xslprofiler.org/

5 Programowanie w XML-u

XSLT nie jest jedynym językiem programowania opartym na składni XML. Tutaj przedstawię dwa alternatywne rozwiązania.

Pierwszy z nich to powłoka xsh powstała z połączenia XML-a z Perlem. Jest to powłoka pozwalająca na poruszanie się po drzewie XML-owym, dodawanie i modyfikację elementów. Całość jest zorganizowana jako moduł Perla, i pozwala na korzystanie z możliwości tego języka. Komendy wykorzystywane w standardowej powłoce do poruszania się po drzewie katalogów, w xsh służą do poruszania się po dokumencie XML – odpowiednikiem ścieżek są wyrażenia języka XPath. Powłoka ma także możliwość wykorzystania arkuszy XSL do przetwarzania dokumentów. Skrypty xsh są świetnym narzędziem do obrabiania dokumentów XML ze względu na możliwość stosowania Perla do operacji na zmiennych zawierających elementay XML.

Drugi projekt to o:XML [4], ktory jest obiektowym językiem programowania opartym na składni XML. Przypomina on nieco XSLT – w przestrzeni nazw `http://www.o-xml.org/lang/` zdefiniowane są instrukcje warunkowe `if` oraz `choose` a także pętla `for-each`. Jest on jednak bogatszy od standardowego XSLT. Poniższy prosty program

```
<?xml version="1.0"?>
<!-- wyliczanka.oml -->
<o:program comments="preserve" space="ignore"
  xmlns:o="http://www.o-xml.org/lang/">
  <o:param name="title" select="'Wyliczanka'"/>
  <o:element name="{$title}">verbatim
    <o:for-each select="split('Ala,Ola,Zosia,Tux', ' ')">
      <o:element name="{.}"/>
    </o:for-each>
  </o:element>
</o:program>
```

po wykonaniu polecenia

```
$ java -cp objectbox.jar org.oXML.engine.ObjectBox wyliczanka.oml
```

da wynik w postaci

```
<?xml version="1.0"?>
<Wyliczanka><Ala/><Ola/><Zosia/><Tux/></Wyliczanka>
```

Dodatkwo o:XML pozwala m.in. na definiowanie własnych typów i posiada obsługę wyjątków, czyli elementy jakie są spotykane w językach obiektowych klasy C++ czy Java.

6 Podsumowanie

W porównaniu z XSLT 1.0 nowa wersja rekomendacji W3C przyniesie wiele zmian. Lista ograniczeń XSLT 1.0 jest dość długa, natomiat o ilości nowinek w XSLT 2.0 oraz XPath 2.0 może świadczyć wzrost objętości szkiców rekomendacji. Mam nadzieję, że ten artykuł zachęci czytelnika do ich wypróbowania i zastosowania w swojej pracy.

Literatura

[1] A. Korlyukov, *Turing Machine in XSLT*, http://www.refal.net/ korlukov/tm/ oraz Unidex, Inc., *Universal Turing Machine in XSLT*, http://www.unidex.com/turing/utm.htm.

[2] Raporty techniczne i publikacjke konsorcjum W3C można znaleźć pod adresem http://www.w3.org/TR/.

[3] P. Prescod, *XSLT and Scripting Languages*, http://www.idealliance.org/papers/xml2001/papers/html/05-03-06.html.

[4] M. Klang, *Programing in o:XML*, http://www.o-xml.org/docs/.

[5] Incremental Development, Inc., *Gallery of Stupid XSL and XSLT Tricks*, http://www.incrementaldevelopment.com/xsltrick/.

[6] D. Pawson, *XSL Frequently Asked Questions*, http://www.dpawson.co.uk/xsl/index.html

[7] Sz. Zioło, *XSLT do kwadratu*, Software 2.0, Nr 6/2003.
Wersja elektroniczna: http://www.empolis.com/img/common/XSLTdokwadratu.pdf

[8] U. Ogbuji, *Debug XSLT on the fly*, http://www-106.ibm.com/developerworks/xml/library/x-debugxs.html