XSL — czyli jak przeczytać XMLowego „Pana Tadeusza”

Mariusz Olko
empolis Polska Sp. z o. o.
Płocka 5a, 01–231 Warszawa
Mariusz.Olko@empolis.pl

Streszczenie

XSL jest szeregiem specyfikacji opisujących w jaki sposób przeprowadzić dokument XML z jego „czystej” strukturalnej formy do niekoniecznie prostej postaci graficznej pozwalającej na jego efektywne użytkowanie.

W artykule opisane są koncepcje leżące u podstaw trasformacji dokumentu od postaci abstrakcyjnej (znakowanego czysto semantycznie i strukturalnie) do formy dogodnej do prezentacji końcowemu użytkownikowi. Omawiamy również podstawy transformacji XSLT na przykładzie przekształcenia „Pana Tadeusza” z XMLa do HTMLa.

1. Wprowadzenie

Dla wielu osób fascynacja technologią dokumentów strukturalnych zaczyna się w momencie, kiedy przygotują swój pierwszy dokument, w którym znajdą się tylko znaczniki semantyczne i strukturalne. Można by rzec, jest to stan, w którym osiągnięta została idealna SGMLowo/XMLowa czysta forma. Pytaniem przed którym się staje w tym momencie jest jak teraz ten dokument można skonsumować lub mówiąc inaczej, jak go przetworzyć.

W początkowym okresie istnienia SGMLa, przetwarzanie dokumentu oznaczało albo skorzystanie z gotowych, bardzo drogich rozwiązań któregoś z producentów oprogramowania SGMLowego, albo samodzielne napisanie programu przetwarzającego. To drugie rozwiązanie pociągało za sobą dużo niższe koszty, jednak nie było wcale łatwe technicznie. Jednym z podstawowych zadań, było napisanie parsera, który byłby w stanie rozpoznawać konstrukcje SGMLowe i przekazywać informacje o nich do właściwego programu. Z czasem pojawił się bardzo dobry parser napisany przez Jamesa Clarka [1] oraz środowiska do przetwarzania dokumentów, których dobrym reprezentantem jest pakiet w języku Perl SGMLPL [2]. W każdym jednak wypadku formatowanie dokumentów wymagało oprócz umiejętności programowania dogłębnej znajomości docelowego medium, takiej jak np. komend sterujących naświetlarką lub, w późniejszym okresie, używanego dość często TEXa. Pojawiające się propozycje języków opisujących styl (np. FOSI) miały zbyt małe możliwości by można je było używać do ogólnych zastosowań.

W związku z tym powstała inicjatywa, aby stworzyć i ustandaryzować język formatowania, który pozwoliłby na opisanie wyniku formatowania w sposób niezależny od urządzenia wyjściowego przy zachowaniu możliwie wysokiej jego jakości. Dodatkowym wymaganiem było, aby język ten można było stosować zarówno do formatowania wsadowego, jak też interakcyjnego np. w edytorach czy też w przeglądarkach. Tak narodził się DSSSL (*Document Style Semantics and Specification Language*), który w roku 1996 doczekał się standaryzacji przez ISO.

DSSSL jest językiem opartym na podzbiorze języka Scheme, pozbawionym efektów ubocznych, to znaczy, że np. nie można używać zmiennych globalnych. Przetwarzanie dokumentu zapisuje się w DSSSL w sposób deklaratywny podając reguły przekształcania, które mają być zastosowane do poszczególnych elementów dokumentu. DSSSL definiuje też model generycznego docelowego urządzenia formatującego w postaci „obiektów formatujących”. W języku tych obiektów zapisuje się końcowe formatowanie dokumentu.

Język DSSSL doczekał się jednej poważnej implementacji napisanej znowu przez Jamesa Clarka. Choć wiele osób próbowało z powodzeniem używać DSSSL do przygotowywania dokumentów SGML do prezentacji, to prawdziwe zainteresowanie wzbudziła część języka związana z możliwościami przekształcania dokumentów SGML do innych dokumentów SGML. Część ta pojawiła się oryginalnie jako uzupełnienie języka formatowania, by umożliwić implementację bardziej zaawansowanych funkcji, jak na przykład przestawianie elementów przy budowaniu spisu treści.

Po zaistnieniu XMLa wśród osób związanych oryginalnie z SGMLem pojawiły się głosy o konieczności skonstruowania języka formatowania dla XML

Polska Grupa Użytkowników Systemu TEX, 2001 (http://www.GUST.org.pl)

z prawdziwego zdarzenia. Idea ta miała początkowo wielu przeciwników, którzy twierdzili, że wystarczają możliwości jakie dają style CSS połączone z możliwościami programowego przekształcania dokumentów językami skryptowymi. Mimo tego zwolennicy wysokopoziomowego deklaratywnego języka opisu stylu rozpoczęli prace nad specyfikacją XSL. XSL nie przypomina oryginalnego języka DSSSL mając składnię XMLową i rozbudowane możliwości adresowania informacji wewnątrz dokumentu XML, niemniej jednak czerpie wiele z doświadczeń zebranych przy opracowywaniu tamtego standardu. Czymś więcej niż zbiegiem okoliczności może być fakt, że redaktorem części XSLa odpowiedzialnej za przekształcanie dokumentu był wspomniany już wcześniej J. Clark.

2. Droga od xmla do wydruku

Idealną sytuacją z punktu widzenia dokumentów strukturalnych jest gdy oprócz samej treści dokument zawiera jedynie informacje (znakowanie) mające odbicie w aspektach biznesowych, których dotyczy. Niepożądanym jest, gdy w dokumencie znajdują się informacje (znakowanie), które wkraczają w obszary technologii późniejszego przetwarzania i wykorzystania dokumentu. Tak więc w ustrukturalizowanych dokumentach będziemy raczej mieć elementy takie jak *autor*, *tytuł* lub *rodział* niż *linia*, *blok tekstu* lub *żywa pagina*. Znakowanie tekstu w sposób ogólny ma wiele zalet, które dają o sobie znać zarówno na etapie przygotowywania tekstu przez autora lub redaktora, jak też w momencie jego bezpośredniego wykorzystania. Trzeba jednak wziąć pod uwagę fakt, że im bardziej ogólnie dokument jest przygotowany tym więcej i bardziej skomplikowanej pracy trzeba wykonać, aby przetworzyć do druku lub prezentacji na ekranie. W związku z tym potrzebujemy mocnych i elastycznych narzędzi, które sprostają tym potrzebom.

Do wizualizacji dokumentu potrzebujemy w zasadzie dwóch elementów:

1. sposobu opisu *wyglądu* dokumentu docelowego,
2. metody przekształcenia znakowania ogólnego dokumentu wyjściowego na znakowanie zorientowane na prezentację dokumentu docelowego.

Jak to było wspomniane wyżej, XSL adresuje właśnie te dwa aspekty formatowania. Na XSL składają się obecnie trzy rekomendacje konsorcjum W3C. Po pierwsze jedna ze specyfikacji definiuje język XMLowy do specyfikowania samego sposobu formatowania elementów w druku bądź w przeglądarce, druga opisuje język przekształcania dokumentów XML w inne dokumenty XML natomiast trzecia definiuje język adresowania elementów dokumentu XML.

Elementy definiujące semantykę samego formatowania są często nazywane „obiektami formatującymi” (ang. *formatting objects*, w skrócie FO). XSLowe obiekty formatujące są opisane w specyfikacji [4], która obecnie jest wciąż na etapie standaryzacji. Jednocześnie sam projekt obiektów formatujących posiada kilka implementacji. Żadna z nich nie jest jednak kompletna i żadna nie jest jeszcze produkcyjnej jakości. Jedną z implementacji jest zestaw makr TEXowych autorstwa Sebastiana Rahtza o nazwie `passivetex` dostępny w CTANie w katalogu `macros/passivetex`.

Sam język przetwarzania nazywany XSLT jest opisany w specyfikacji [3], która jest już rekomendacją konsorcjum W3C i w niedługim czasie powinna doczekać się drugiej, rozszerzonej wersji. Odłączenie języka przetwarzania XSLT od samego języka specyfikującego formatowanie pozwoliło na użycie również innych sposobów wizualizacji dokumentów niż tylko poprzez obiekty formatujące XSL. Rolę tych obiektów może spełnić na przykład HTML, który w obecnych wersjach różnych producentów zapewnia bardzo rozbudowane możliwości typograficzne. Inną możliwością jest skorzystanie z własnych „obiektów formatujących”. Takimi obiektami mogą być np. standardowe klasy LATEXa, „przetłumaczone” na składnię XMLową. Dokument XMLowy przekształcony do takich XMLowo/LATEXowych obiektów formatujących można skompilować TEXem przy użyciu pakietu `xmltex`, który interpretuje bezpośrednio składnię XMLa rozwijając dla każdego z elementów odpowiadające makro TEXowe.

3. Dokument xml

3.1. Pan Tadeusz

W przykładach w dalszej części artykułu będziemy używać tekstu „Pana Tadeusza” oznakowanego w XMLu według prostego DTD zamieszczonego poniżej. Dokument ten będziemy przekształcać do strony HTML.

```
<!ELEMENT poemat     (info, ksiega+)            >
<!ELEMENT info       (autor, tytul,
                       podtytul, podpodtytul) >

<!ELEMENT autor      (#PCDATA)                  >
<!ELEMENT tytul      (#PCDATA)                  >
<!ELEMENT podtytul   (#PCDATA)                  >
<!ELEMENT podpodtytul (#PCDATA)                 >

<!ELEMENT ksiega     (numer?, tytul,
                       spis?, strofa+)          >
<!ELEMENT numer      (#PCDATA)                  >

<!ELEMENT spis       (rzecz+)                   >
<!ELEMENT rzecz      (#PCDATA)                  >
```


Polska Grupa Użytkowników Systemu TEX, 2001 (http://www.GUST.org.pl)

```
<!ELEMENT strofa    (w+)                          >
<!ELEMENT w         (#PCDATA|i|emph|wstrofa)*>
<!ELEMENT i         (#PCDATA)                     >
<!ELEMENT emph      (#PCDATA)                     >
<!ELEMENT wstrofa   EMPTY                         >

<!ENTITY eacute      "&#x00E9;"                   >
<!ENTITY agrave      "&#x00E0;"                   >
<!ENTITY lguillemet "&#x00AB;"                    >
<!ENTITY rguillemet "&#x00BB;"                    >
```

Poemat składa się z informacji redakcyjnych oraz z trzynastu ksiąg. W informacjach redakcyjnych znajduje się autor oraz trzy poziomy tytułów. Każda księga zaczyna się słownie zapisanym numerem księgi, po którym występuje tytuł. Po tytule umieszczony jest spis rzeczy pojedynczej księgi a następnie sekwencja strof składających się z wersetów.

W wersecie może znaleźć się jedno z dwóch rodzajów wyróżnień, jakie występują w poemacie. Są to skład kursywą oraz skład drukiem rozstrzelonym. Dodatkowo niektóre z wersetów są podzielone pomiędzy dwie różne strofy. W naszym dokumencie jest to zaznaczone pustym elementem `<wstrofa/>`.

3.2. Drzewo dokumentu

Dokument XML może być zamodelowany jako struktura drzewiasta. Drzewo dokumentu jest podstawową strukturą, którą przetwarza język XSLT. Szczegółowy model dokumentu XML jako drzewa jest opisany w rekomendacji [3].

W drzewie dokumentu znajdują się węzły kilku typów, z których najważniejsze to:

korzeń ten węzeł jest jeden dla każdego dokumentu, jego dzieckiem jest węzeł elementu głównego,

element węzeł odpowiada elementowi, jego dziećmi mogą być węzły elementowe, tekstowe, instrukcje przetwarzania i komentarze,

tekst węzeł odpowiada fragmentowi tekstu w dokumencie, nie może mieć węzłów potomnych,

komentarz węzeł zawiera tekst komentarza, nie może mieć węzłów potomnych,

instrukcja sterująca węzeł zawiera tekst instrukcji sterującej.

Korzystając z tego modelu możemy reprezentować „Pana Tadeusza” jako drzewo podobne do tego na rysunku 1. Na rysunku prostokąty symbolizują węzły elementowe natomiast owale węzły tekstowe. Na samym szczycie drzewa znajduje się węzeł-korzeń.

4. Transformacja dokumentu

4.1. Arkusz stylu

Opis sposobu przetwarzania dokumentu jest umieszczany w pliku nazywanym arkuszem stylu. Plik musi być dobrze uformowanym plikiem XMLowym — tzn. musi zaczynać się deklaracją XMLową, a wszystkie znaczniki początku elementu muszą posiadać sparowane znaczniki końca. W arkuszu stylu mogą znaleźć się dwa rodzaje elementów. Jedne elementy opisują logikę przetwarzania dokumentu, drugie natomiast są elementami, które zostaną wstawione do drzewa dokumentu wyjściowego. Aby procesor mógł poprawnie rozróżnić jedne elementy od drugich, elementy opisujące przetwarzanie należą do przestrzeni nazw związanych ze standardem XSLT. W praktyce oznacza to, że zwyczajowo poprzedza się te elementy przedrostkiem `xsl:`, który jest przyporządkowany odpowiedniej przestrzeni nazw XML na jednym z głównych elementów.

Jednym z pierwszych elementów w arkuszu powinien być element `<xsl:output>`, którego atrybut `method` będzie zawierał informację o formacie pliku wyjściowego. Ma to wpływ na sposób zapisywania i formatowania pliku wyjściowego. Zazwyczaj atrybut ten przyjmuje wartość *html*, jeśli wynikiem transformacji ma być dokument HTML, lub *xml*, jeśli tworzymy nową wersję dokumentu XML.

Początek więc naszego arkusza stylów powinien wyglądać następująco:

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

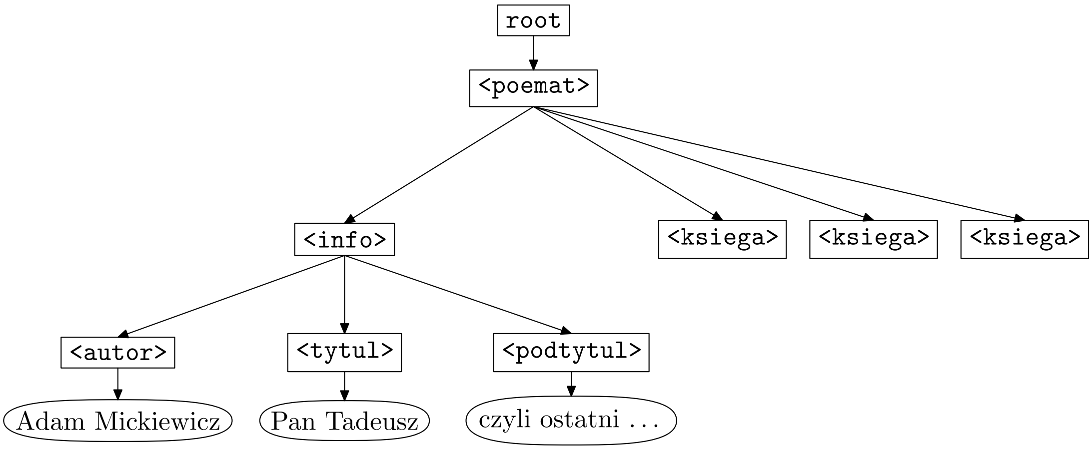
Arkusz stylów składa się z wielu szablonów przypisanych różnym wzorcom węzłów dokumentu XML. Na różnych etapach przetwarzania procesor XSLT usiłuje dobrać do konkretnego węzła dokumentu najbardziej szczegółowo opisujący go wzorzec zapisany w arkuszu stylów, a następnie wykonać szablon skojarzony z przyporządkowanym wzorcem.

Pojedynczy szablon jest zapisany we wnętrzu elementu `<template>`. W atrybucie `match` podaje się wzorzec adresujący węzeł, do którego dany szablon ma być stosowany. W naszym przykładzie będziemy chcieli, aby w wynikowym dokumencie HTML każdy początek strofy rozpoczynał nowy akapit, oraz aby każdy werset kończył się znakiem nowej linii. Zapiszemy to w postaci dwóch szablonów przyporządkowanych odpowiednio węzłom związanym z elementem `<strofa>` oraz `<w>`.

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

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


Polska Grupa Użytkowników Systemu TEX, 2001 (http://www.GUST.org.pl)



Rysunek 1: Drzewiasta reprezentacja dokumentu XML

Znaczenie tych szablonów powinno być oczywiste. Wstawiają one do drzewa wynikowego transformacji elementy `<P>` oraz pusty element `<BR>`. We wnętrzu obu szablonów znajduje się element `<xsl:apply-templates>`, który jest poleceniem dla procesora XSLT by w to miejsce wstawić wynik przetwarzania węzłów, będących dziećmi węzła bieżącego. Ta rekurencyjna metoda pozwala na zapisanie przetwarzania w postaci szablonów zależnych jedynie od kontekstu danego węzła, a nie wiedzy o pełnej strukturze dokumentu.

Jeśli procesor nie potrafi dopasować konkretnego wzorca z arkusza stylu, wtedy do transformacji stosuje wzorzec domyślny. Dla węzłów tekstowych domyślny szablon wpisuje w drzewo wynikowe tekst będący wartością węzła. Dla węzłów elementowych natomiast szablon domyślny zawiera jedynie element `<xsl:apply-templates>`, czyli polecenie „przetwórz wszystkie węzły potomne”.

4.2. Wzorce

Najprostsze wzorce przyporządkowane do szablonów (takie jak te z poprzedniego przykładu) mają postać nazwy elementu, którego węzły chcemy wybrać. W ogólnym przypadku wzorce mogą mieć dużo bardziej skomplikowaną konstrukcję pozwalającą na bardzo szczegółowe wyspecyfikowanie warunków, które powinny spełniać interesujące nas węzły. Pełna składnia wzorców jest zdefiniowana w rekomendacji konsorcjum W3C XPATH [5].

Wyrażenie XPATH jest zapisywane w postaci ścieżki dostępu do adresowanych węzłów zapisanej w postaci poszczególnych *kroków*. Jeżeli ścieżka dostępu zaczyna się od znaku `/` to oznacza to, że rozpoczynamy nawigację od korzenia dokumentu. Jeżeli zaczyna się od znaków `//` to oznacza, że możemy rozpocząć nawigację gdziekolwiek w dokumencie. Jeśli zaczynamy ścieżkę *krokiem*, to nawigacja zaczyna się w bieżącym węźle drzewa.

Najprostszy zapis kroku, to podanie nazwy elementu, do którego chcemy się przemieścić. Tak więc zapis `info/tytul` oznacza, że chcemy wybrać tylko te elementy `<tytul>`, które znajdują się wewnątrz elementu `<info>`. W naszym przykładzie wzorzec `/poemat/ksiega` pasuje do wszystkich węzłów z księgami poematu.

W naszym tekście „Pana Tadeusza” użyliśmy elementu `<tytul>` do określenia zarówno tytułu samego poematu jak też tytułów jego poszczególnych ksiąg. W dokumencie wynikowym, tytuł poematu powinien być sformatowany jako HTMLowy element `<H1>`, natomiast tytuły ksiąg jako element `<H2>`. Możemy to osiągnąć zapisując dwa szablony jeden ogólny dla wszystkich elementów `<tytul>`, drugi tylko dla takich które są bezpośrednimi potomkami elementu `<info>`.

```
<xsl:template match="tytul">
 <H2><xsl:value-of select="."/></H2>
</xsl:template>

<xsl:template match="info/tytul">
  <H1><xsl:value-of select="."/></H1>
</xsl:template>
```

W przypadku gdy do jednego węzła pasuje więcej niż jeden wzorzec, transformator rozstrzyga, który szablon użyć stosując reguły priorytetowe. Mówią one między innymi, że szablon przyporządkowany do wzorca bardziej szczegółowego (czyli w naszym wypadku taki, który mówi o typie rodzica), ma pierwszeństwo nad wzorcem bardziej ogólnym.

Podane dotychczas przykłady dotyczyły adresowania elementów według ich nazw. XPATH pozwala

Polska Grupa Użytkowników Systemu TEX, 2001 (http://www.GUST.org.pl)

na nawigację również po innych tak zwanych *osiach.* W ogólnym przypadku *krok* ma postać *oś::test węzła[predykat].* Najważniejsze osie po których możemy się poruszać to:

- oś przodków *ang.* ancestor (tzn. węzłów położonych w drzewie „nad” węzłem bieżącym),
- oś potomków *ang.* descendant (tzn. węzły leżące w drzewie poniżej),
- oś dzieci *ang.* child (tzn. węzły bezpośrednio doczepione w drzewie do węzła bieżącego),
- oś rodzica *ang.* parent (tzn. węzła do którego węzeł bieżący jest bezpośrednio doczepiony),
- oś poprzedników *ang.* preceding (tzn. wszystkich węzłów w dokumencie znajdujących się „przed”),
- oś następników *ang.* following (tzn. wszystkich węzłów w dokumencie znajdujących się „za”),
- oś sąsiadów z przodu *ang.* proceeding-sibling (tzn. wszystkich węzłów mających tego samego rodzica, które znajdują się „przed”),
- oś sąsiadów z tyłu *ang.* following-sibling (tzn. wszystkich węzłów mających tego samego rodzica, które znajdują się „za”),
- oś siebie samego *ang.* self.

Lista ta nie jest wyczerpująca, ale pokrywa najczęściej napotykane przypadki.

Tak więc zapisany wcześniej w formie skrótowej wzorzec `poemat/ksiega` może zostać zapisany w formie rozwiniętej jako `self::poemat/child::ksiega`. Oznacza to, że na osi *węzła bieżącego* chcemy znaleźć element o nazwie `poemat`, po czym znaleźć element-dziecko o nazwie `ksiega`. Jeśli chcemy wybrać wszystkie węzły (również te tekstowe, z komentarzami itp.) należy podać jako test węzła funkcję `node()`. Tak więc, aby zaadresować wszystkie węzły potomne od elementu `<spis>` należy zapisać wyrażenie `self::spis/descendant::node()`. W teście możemy umieścić też znak `*` wybierający wszystkie węzły elementowe, lub jedną z funkcji wybierającą węzły określonego typu: `comment()`, `text()` lub `processing-instruction()`.

Ostatnim elementem kroku może być predykat. Jest on zapisywany w nawiasach kwadratowych i podaje warunek, który powinien być spełniony przez wybierany węzeł. Jeżeli wyrażenie podane w predykacie wylicza się do liczby to jest to interpretowane jako test położenia węzła w stosunku do jego sąsiadów. Na przykład wers 125 trzeciej księgi możemy zaadresować jako `/poemat/ksiega[3]//w[125]`. W każdym innym przypadku wyrażenie jest konwertowane do wartości boolowskiej. Na przykład wszystkie wersy zawierające tekst zapisany kursywą możemy wyszukać wzorcem `w[./i]`.

Jako przykład zastosowania tych rozbudowanych możliwości adresowania w praktyce, przygotujmy szablon formatujący wersy poematu, które są podzielone pomiędzy dwie różne strofy, tak jak na przykład wers 125 w księdze trzeciej.

Mamy do rozwiązania dwa problemy. Po pierwsze, chcielibyśmy rozróżnić taki wers od innych, a po drugie zaprezentować go tak, aby początek nowej strofy był wcięty na długość ostatniej linii strofy poprzedniej.

Popatrzmy na propozycję

```
<xsl:template match="w[wstrofa]">
 <xsl:apply-templates
   select="wstrofa/preceding-sibling::node()"/>
 <P/>
 <SPAN style="visibility: hidden">
 <xsl:apply-templates
   select="wstrofa/preceding-sibling::node()"/>
 </SPAN>
 <xsl:apply-templates
   select="wstrofa/following-sibling::node()"/>
 <BR/>
</xsl:template>
```

Szablon posiada wzorzec wybierający tylko te wiersze, które zawierają w sobie element `<wstrofa>`. Do wybrania wierszy posłużyliśmy się predykatem, który jest fałszywy, jeśli wiersz nie zawiera elementu `<wstrofa>`. W treści szablonu poddajemy najpierw domyślnemu przetwarzaniu wszystkie węzły, które znajdują się przed elementem `<wstrofa>`, po czym na końcu wstawiamy znak nowego akapitu. Teraz powtarzamy to samo przetwarzanie, ale wynik otaczamy elementem HTMLowym `<SPAN>` i ukrywamy. W obu przypadkach skorzystaliśmy z adresowania węzłów wzdłuż osi *sąsiedzi z przodu.* Dalej pozostaje przetworzyć normalnie wszystkie węzły następujące po elementcie `<wstrofa>`. W ten sposób wykorzystując mocne mechanizmy wybierania węzłów i ich specyfikacji przy pomocy XPATH mogliśmy zwięźle zapisać tę dość skomplikowaną operację.

4.3. Iteracje

W poprzednich przykładach zapisaliśmy wzorce, dzięki którym otrzymamy sformatowaną zasadniczą część naszego przykładowego poematu. Aby wygenerowana strona WWW była bardziej użyteczna możemy pokusić się, aby na jej początku wygenerować spis treści z hiperlinkami do każdej z ksiąg.

Generację spisu treści umieścimy w szablonie dla elementu `<poemat>`. Ten szablon będzie przede wszystkim generował standardowe elementy zewnętrzne poprawnej strony HTML. Zaczniemy zatem od elementu `<HTML>` i nagłówka, w którym ustawimy tytuł strony.

```
<xsl:template match="poemat">
```


Polska Grupa Użytkowników Systemu TEX, 2001 (http://www.GUST.org.pl)

```
<HTML>
  <HEAD>
    <TITLE>
      <xsl:value-of
           select="//poemat/metryka/tytul"/>
    </TITLE>
  </HEAD>
  <BODY>
    <xsl:apply-templates select="info"/>
```

Wewnątrz tytułu skorzystaliśmy z elementu `<xsl:value-of>`, który umożliwia wstawienie tekstu będącego wartością węzła, którego adres został podany w atrybucie `select`. Dla węzła tekstowego jego wartość jest zdefiniowana jako tekst znajdujący się w węźle, w przypadku węzła elementowego jego wartością jest konkatenacja wartości wszystkich jego węzłów dzieci.

Zasadnicza część dokumentu HTML rozpoczyna się elementem `<BODY>`. Wewnątrz zostają umieszczone przetworzone informacje redakcyjne. Korzystając z odpowiednich szablonów zostaną tu umieszczone: autor, tytuł, podtytuł itp.

Spis treści „Pana Tadeusza" umieścimy wewnątrz nienumerowanej listy `<UL>`.

```
<UL>
  <xsl:for-each select="/poemat/ksiega">
    <LI><A HREF="#{position()}">
      <xsl:value-of select="tytul"/>
    </A></LI>
  </xsl:for-each>
</UL>
<HR/>
```

Do iteracji po kolejnych księgach poematu użyliśmy elementu `<xsl:for-each>`, który jest instrukcją dla procesora XSLT, aby dla wszystkich węzłów dokumentu, które zostaną wybrane przez wyrażenie podane w atrybucie `select` zastosować szablon umieszczony we wnętrzu elementu. W naszym przykładzie szablon ten wstawi dla wszystkich kolejnych ksiąg element listy HTMLowej `<LI>`. Wewnątrz elementu wstawiona zostanie kotwica linku z tytułem księgi jako częścią aktywną linku. W atrybucie `HREF` zostanie zapisany identyfikator elementu do którego link prowadzi. Ten identyfikator jest zapisany w postaci funkcji *position()* ujętej w nawiasy klamrowe. Nawiasy oznaczają dla procesora XSLT wyrażenie, które powinno zostać obliczone w trakcie przetwarzania, a jego wynik wstawiony do docelowego drzewa. Funkcja *position()* podaje miejsce danego węzła w kolekcji, w tym wypadku jest to kolekcja węzłów po których odbywa się iteracja elementem `<xsl:for-each>`.

Po zbudowaniu spisu treści wznawiamy przetwarzanie wszystkich węzłów dzieci z wyjątkiem elementu `<info>`, który już został umieszczony na stronie wynikowej.

```
    <xsl:apply-templates
         select="*[not(self::metryka)]"/>
  </BODY>
 </HTML>
</xsl:template>
```

Ostatnim szablonem, który musimy dopisać jest szablon dla elementu kięga. To tutaj umieścimy cele do których będą odwoływać się linki hipertekstowe w spisie treści.

```
<xsl:template match="ksiega">
  <SPAN ID="{count(preceding::ksiega)+1}" />
  <xsl:apply-templates/>
</xsl:template>
```

Szablon umieszcza na początku każdej księgi pusty element `<SPAN>` zawierający w atrubucie `ID` identyfikator, który jest obliczony przez procesor podczas generacji docelowego drzewa. Wartością tego identyfikatora jest, obliczona przez funkcję *count()*, liczba węzłów odpowiadających elementowi `<ksiega>` znajdujących się przed bieżącym węzłem.

5. Zakończenie

W tym krótkim artykule próbowaliśmy pokazać podstawowe sposoby używania języka XSLT do przekształcania dokumentów XML. Język ten oferuje dużo więcej możliwości niż to udało się nam tutaj zmieścić. Szersze omówienie szczegółów języka oraz technik jego używania można znaleźć w książce [6].

„Pana Tadeusza" w formacie XML oraz style XSLT omawiane w tym artykule można znaleźć na stronie `http://www.molko.com.pl/pan-tadeusz/`.

Bibliografia

[1] strona domowa *Jamesa Clarka* patrz `http://www.jclark.com/`

[2] *Pakiet do przetwarzania dokumentów* SGML *w języku Perl* patrz `http://www.oasis-open.org/cover/publicSW.html#sgmlspm`

[3] World Wide Web Consortium, XSL *Transformations (*XSLT*)*, patrz `http://www.w3.org/TR/xslt.html`

[4] World Wide Web Consortium, *Extensible Stylesheet Language (XSL)*, patrz `http://www.w3.org/TR/xsl/`

[5] World Wide Web Consortium, XML *Path Language (XPath)*, patrz `http://www.w3.org/TR/xpath.html`

[6] Michael Kay, XSLT *Programmer's Reference*. Wrox Press Ltd., 2000

◇ Mariusz Olko
`Mariusz.Olko@empolis.pl`

Polska Grupa Użytkowników Systemu TeX, 2001 (http://www.GUST.org.pl)