# Optymalizacja czasowa transformacji dokumentów XML dla potrzeb zastosowań WWW

Michał Jaśtak

**Streszczenie**

Artykuł prezentuje technikę transformacji dokumentów XML z użyciem XSLT wspomaganą pamięciami podręcznymi procesorów XSLT i dokumentów wynikowych. Wspomaganie to ma na celu przyśpieszenie odpowiedzi na żądanie HTTP. Prezentowana technika wykorzystuje język programowania Java i serwlety.

## 1. Wstęp

Rola serwisów WWW we współczesnej sieci Internet rośnie z dnia na dzień. Na nich opiera się w coraz większym stopniu działalność wielu firm, gdyż dzięki nim mają między innymi możliwość prezentacji coraz większemu gronu odbiorców profilu swojej działalności czy też swoich produktów. Coraz większą popularność zdobywają też internetowe sklepy, banki, itp. Twórcy wielu z tych serwisów WWW odchodzą od statycznego kodu HTML na rzecz tworzonego dynamicznie, bardzo często stosując techniki oparte na języku programowania Java. Niezwykle istotne jest zapewnienie efektywności tego typu silnie obciążonych serwisów WWW. W tym artykule poruszamy problematykę polepszenia efektywności generowania dynamicznych stron HTML z wykorzystaniem serwletów Javy.

Eric M. Burke w artykule [7] omawia kilka problemów związanych z tworzeniem stron WWW z zastosowaniem inkluzji JSP w kodzie HTML, czy też tworzenia dokumentów HTML poprzez serwlety. Ukazuje również metodę dość ciekawą i w porównaniu do pozostałych najmłodszą, związaną z użyciem XML i XSL (patrz [1] i [2]). Metoda ta polega na zastosowaniu języka XML do opisu prezentowanej informacji, zaś XSL do opisu sposobu jej prezentacji. Technika ta ma więc tę przewagę nad zastosowaniami z użyciem wplatania kodu HTML w serwlety Javy, czy też JSP, że umożliwia ścisły rozdział treści dokumentu, formy jego prezentacji i samego kodu serwletu. Daje to możliwość separacji pracy programistów serwletów od osób tworzących warstwę prezentacyjną danych w HTML, co upraszcza utrzymanie i rozwój serwisów WWW.

Przetwarzanie tą metodą statycznych dokumentów XML daje się łatwo rozbudować na dokumenty tworzone dynamicznie z użyciem takich interfejsów programowania aplikacji (API) jak np. JDOM, na podstawie takich źródeł danych jak np. bazy danych (z użyciem JDBC), czy też serwery LDAP (z użyciem JNDI).

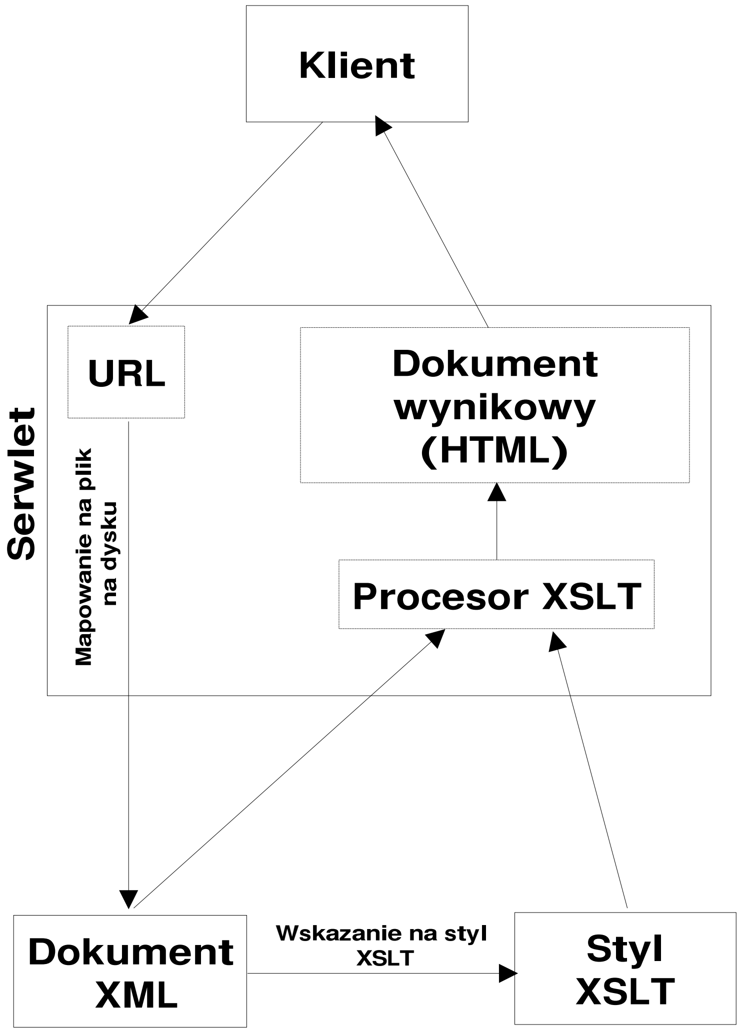
Artykuł ten koncentruje się na możliwościach optymalizacji tej metody prezentacji informacji na stronach WWW pod kątem minimalizacji czasu potrzebnego na uzyskanie odpowiedzi na żądanie HTTP.

## 2. Rozwiązanie standardowe

Rysunek 1 prezentuje schemat działania serwletu naszkicowany w pracy [6]. Serwlet taki po otrzymaniu żądania HTTP mapuje URL podany w żądaniu na dokument XML odpowiadający mu na dysku. Dokument ten dzięki *Processing instruction* `xml-stylesheet` (patrz [3]) związany jest z konkretnym stylem XSL. Powiązanie to jest odczytywane i na jego podstawie jest tworzony procesor XSLT zawierający tenże styl XSL. Procesor ten wykonuje transformację dokumentu XML do dokumentu wynikowego, który jest odbierany przez serwlet i odsyłany do klienta. Proces ten jest dość czasochłonny, o czym można się przekonać w dalszej części tego artykułu. Tego typu rozwiązanie może być dużym obciążeniem dla serwera WWW, który miałby go używać w praktyce.

## 3. Optymalizacja pracy serwletu z użyciem pamięci podręcznej

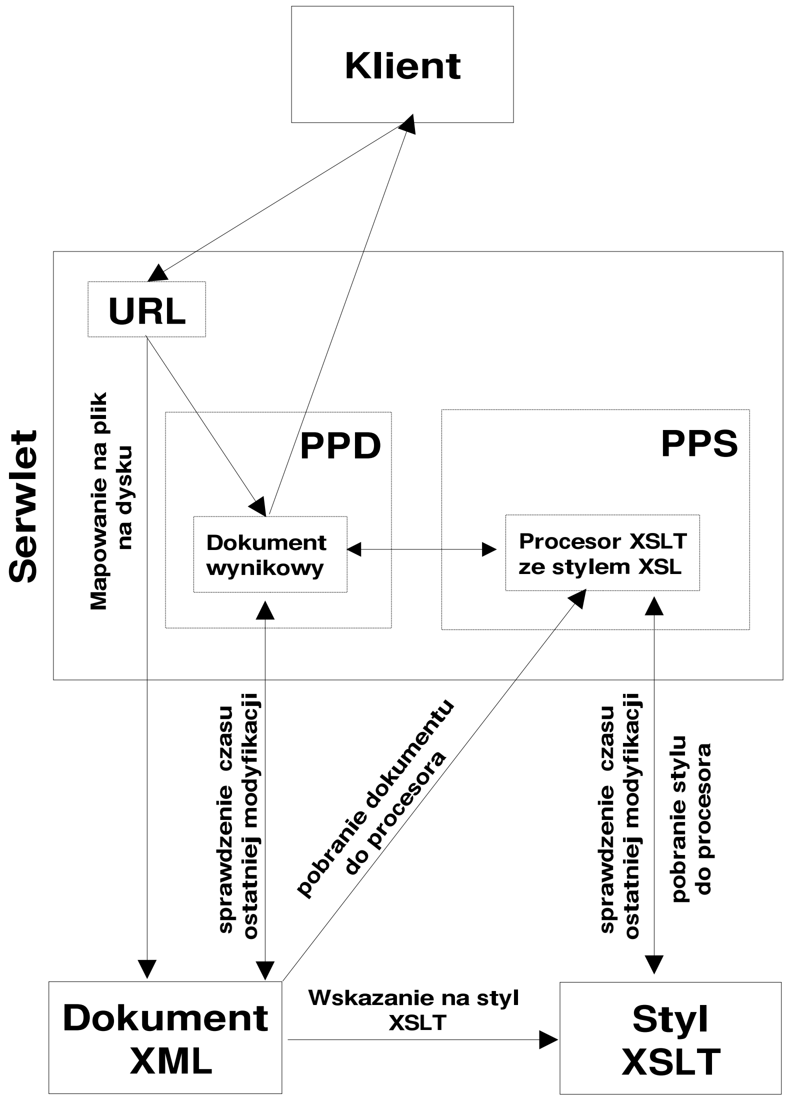
Patrząc na szczegóły pracy serwletu w rozwiązaniu standardowym, wydaje się, że najbardziej czasochłonne fragmenty jego pracy stanowią ładowanie stylu XSLT do procesora i sama transformacja dokumentu XML z użyciem tego stylu. W pracy [7] wskazany jest przykład możliwej optymalizacji takiej sytuacji dla konkretnego dokumentu za pomocą zapamiętania procesora XSLT z prekompilowanym stylem XSL. Przykład ten rozszerzymy, zaimplementujemy i ocenimy. Jego rozszerzenie polegać będzie na wykorzystaniu pamięci podręcznej zawierającej przygotowane do użycia procesory XSLT z prekompilowanymi stylami XSL (w ogólniejszym przypadku niż wskazany w [7]). Ten




Rysunek 1: *Przepływ danych podczas pracy zwykłego serwletu XML/XSL.*

rodzaj pamięci podręcznej oznaczać będziemy `PPS`. W naszych rozważaniach weźmiemy dodatkowo pod uwagę niezależną od poprzedniej pamięć podręczną dokumentów wynikowych (`PPD`). Rysunek 2 prezentuje zmodyfikowany pod tym kątem schemat działania serwletu

Przy zastosowaniu pamięci podręcznych `PPS` i `PPD` droga, jaką musimy przebyć od przyjęcia przez serwlet żądania HTTP do odesłania do klienta dokumentu w HTML, zależy od dwóch czynników: czasu ostatniej modyfikacji




Rysunek 2: *Diagram ilustrujący przepływ danych podczas pracy serwletu z użyciem pamięci podręcznej*

dokumentu źródłowego XML i czasu ostatniej modyfikacji stylu XSL.

W przypadku zmiany samego dokumentu XML uzyskujemy przyśpieszenie jego ponownej transformacji dzięki zastosowaniu `PPS`. Zmiana stylu XSL pociąga za sobą niestety konieczność ponownej transformacji dokumentu XML i odświeżenia obu pamięci podręcznych. Warto jednak pamiętać, że w codziennej pracy tandemu XML/XSL zmiana stylu zachodzi dużo rzadziej niż zmiana samego dokumentu.

## 3.1. Implementacja

Główne interfejsy programistyczne API jakich używać będziemy do realizacji naszego zadania to JAXP w wersji 1.1 (patrz [4]) oraz Servlet API w wersji 2.2 (patrz [5]).

Implementacja składa się z trzech klas: *XMLCache* – realizujacej `PPD`, *XSLCache* – realizującej `PPS` i *xslTransform*, która realizuje właściwy serwlet. Omówimy pokrótce najważniejsze fragmenty tych klas; pełen kod źródłowy znaleźć można w *http://www.ncac.torun.pl/~warlock/kkio2001/*.

Dokumenty wynikowe przechowywane są w postaci tablicy bajtów wewnątrz obiektów klasy *XMLCache*, które reprezentują elementy `PPD`.

Każdy dokument XML jest podczas prezentacji przez WWW przekształcany z użyciem pewnego stylu XSL. Powiązanie między dokumentem a stylem, podobnie jak w rozwiązaniu standardowym, jest realizowane za pomocą *Processing instruction* `xml-stylesheet` (patrz [3]). Podczas tworzenia obiektu klasy *XMLCache* następuje sprawdzenie tego powiązania i zapamiętanie dwóch obiektów klasy *File* wskazujących odpowiednio na plik zawierający dokument XML i plik zawierający skojarzony z nim styl XSL. Realizuje to poniższy fragment kodu:

```
TransformerFactory tFactory =
                   TransformerFactory.newInstance();
Source xslStyle = tFactory.getAssociatedStylesheet(
                  new StreamSource(xmlFile), null, null, null);
URL xslURL = new URL(xslStyle.getSystemId());
   xslFile = new File(xslURL.getFile());
```

Każdy obiekt klasy *XSLCache* reprezentujący elementy `PPS`, zawiera jako jedno z pól obiekt klasy *Templates* przechowujący wzorce procesorów XSLT z prekompilowanym stylem XSL. Zastosowanie klasy *Templates* związane jest z koniecznością zapewnienia równoległego wykorzystania tego samego elementu `PPS` przez kilka wątków serwletu. Przygotowany do działania wzorzec procesora XSLT jest tworzony w konstruktorze klasy *XSLCache*, w nastepującym fragmencie kodu:

```
TransformerFactory tFactory =
                   TransformerFactory.newInstance();
xslTemplates = tFactory.newTemplates(
               new StreamSource(xslFile));
```

Dla konkretnego wątku serwletu odpowiedni procesor XSLT uzyskujemy poprzez metodę:

```
public Transformer newTransformer()
       throws TransformerConfigurationException
{ return xslTemplates.newTransformer(); }
```

Obiekt klasy *xslTransform* realizuje pamięci podręczne PPS i PPD poprzez dwa obiekty klasy *Hashtable*, po jednym dla każdej z nich. W przypadku PPD kluczem indeksującym odpowiedni element jest wartość zwrócona przez metodę *getRequestURI* klasy *HttpServletRequest*, zaś dla PPS – bezwzględna ścieżka do pliku zawierającego styl XSL.

## 3.2. Porównanie wydajności

Rozwiązanie przez nas rozważane zostało zaimplementowane z użyciem języka Java w wersji Java 2 SDK 1.3 firmy IBM. Jako parsera XML użyto Xerces'a w wersji 1.3.1 (patrz [9]), jako procesora XSLT użyto Xalan'a (patrz [8]) w wersji 2.0.1, zaś jako serwer serwletów użyto Resin'a (patrz [10]) w wersji 1.2.5.

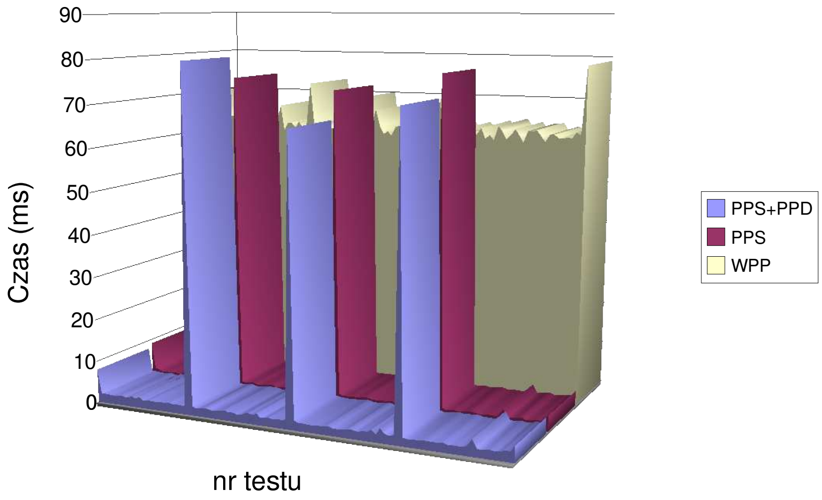
Implementacja była testowana na komputerze klasy Pentium III Coppermine z 256KB Cache L2, 256 MB RAM, z interfejsem dyskowym Ultra ATA 100, pod kontrolą systemu operacyjnego Linux (dystrybucja RedHat 7.1) z jądrem 2.4.2
Wszystkie przeprowadzone testy polegały na wykonaniu 100 serii, każda po 100 żądań HTTP, a następnie wyciągnieciu średniej z odpowiadających sobie w kolejnych seriach żądań.

Podczas testów dała się wyraźnie zaobserwować bardzo długą odpowiedź na pierwsze po starcie serwletu żądanie HTTP, rzędu 1000 - 2300 ms. Związana jest ona z inicjacją serwletu i ładowaniem klas potrzebnych do dalszej pracy. Rysunki i tablice prezentujące wyniki testów pomijają tę pierwszą odpowiedź w celu zwiększenia czytelności całości.

W opisach wszystkich rysunków użyto skrótów, z których część już występowała w tym artykule, a mianowicie: PPS – pamięć podręczna procesorów XSLT, PPD – pamięć podręczna dokumentów wynikowych, WPP – wyłączona pamięć podręczna obu typów .

### 3.2.1. Test reakcji na zmianę dokumentów XML

Test ten będziemy dalej nazywać testem XML; jego wyniki obrazują rysunek 3 oraz tablica 3.2.1. W trakcie trwania tego testu, regularnie co 25 żądań HTTP, następowała zmiana dokumentu XML na dysku. Widać to wyraźnie na rysunku 3 w postaci 3 dużych skoków czasu uzyskania odpowiedzi.




Rysunek 3: *Test XML: Porównanie czasów odpowiedzi*

| | PPD + PPS | PPS | WPP |
|---|---|---|---|
| Średnia: | 4.31 ms | 4.37 ms | 63.58 ms |
| Min: | 1.78 ms | 1.75 ms | 60.21 ms |
| Max: | 80.32 ms | 77.78 ms | 79.01 ms |

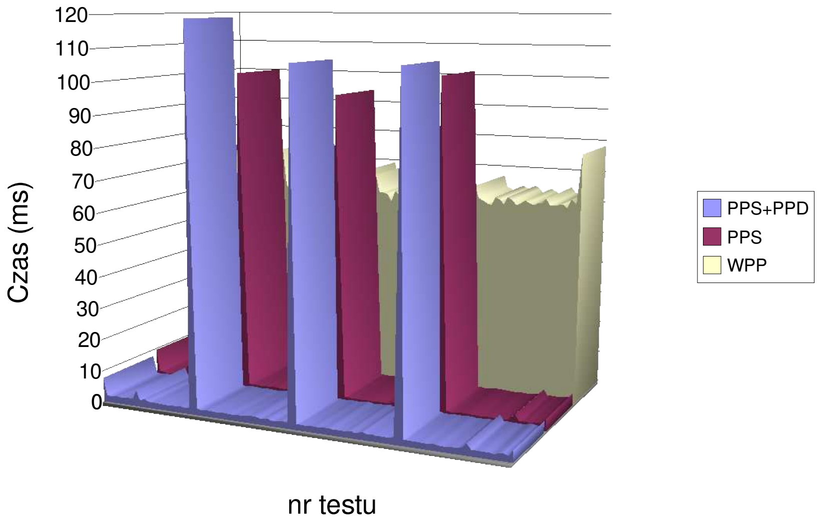
Tablica 3.2.1: *Statystyka czasów odpowiedzi na żądanie HTTP dla testu XML*

### 3.2.2. Test reakcji na zmianę stylu XSL

Test ten nazywać będziemy dalej testem XSL; jego wyniki obrazują rysunek 4 i tablica 3.2.2. W tym teście, również co 25 żądań HTTP, następowała zmiana stylu XSL na dysku. I w tym wypadku występują 3 wyraźne skoki czasu odpowiedzi.

| | PPD + PPS | PPS | WPP |
|---|---|---|---|
| Średnia: | 5.39 ms | 6.69 ms | 63.64 ms |
| Min: | 1.73 ms | 1.82 ms | 60.15 ms |
| Max: | 119.14 ms | 103.05 ms | 78.48 ms |

Tablica 3.2.2: *Statystyka czasów odpowiedzi na żądanie HTTP dla testu XSL*




Rysunek 4: *Test XSL: Porównanie czasów odpowiedzi*

### 3.2.3. Test czasu odpowiedzi w przypadku niezmienności dokumentu XML i stylu XSL

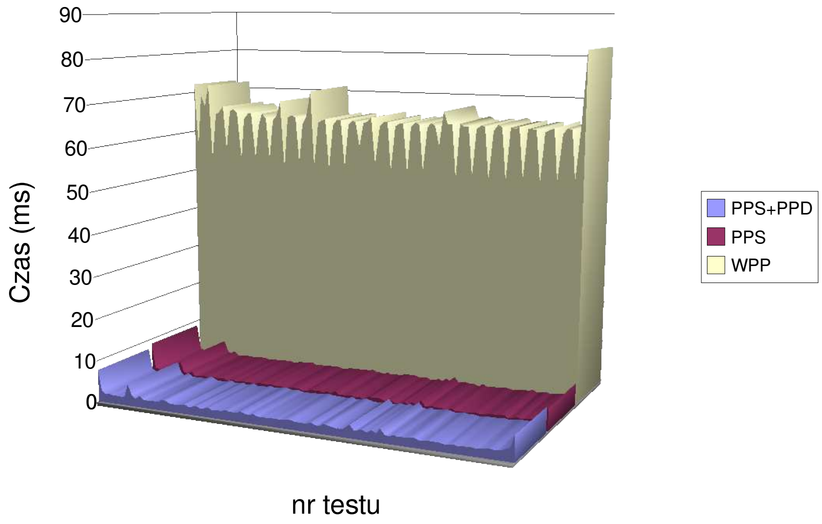
Test ten nazywać będziemy testem niezmienności. Obrazuje on czas potrzebny na uzyskanie odpowiedzi HTTP przy pracy serwletu na dokumentach XML i stylach XSL niezmiennych przez cały okres trwania testu. Wyniki tego testu obrazują rysunek 5 oraz tablica 3.2.3.

| | PPD + PPS | PPS | WPP |
|---|---|---|---|
| Średnia: | 2.24 ms | 2.24 ms | 62.03 ms |
| Min: | 1.75 ms | 1.7 ms | 51.51 ms |
| Max: | 7.65 ms | 8.65 ms | 82.38 ms |

Tablica 3.2.3: *Statystyka czasów odpowiedzi na żądanie HTTP dla testu niezmienności*

## 4. Wnioski

Wyniki testów wskazują, że wykorzystanie pamięci podręcznej przyśpiesza czas uzyskania odpowiedzi na żądanie HTTP od 8 do ponad 25 razy w stosunku do czasów uzyskiwanych przy zastosowaniu standardowej transformacji dokumentów XML. Optymalizacja tego typu daje serwerowi WWW, na którym




Rysunek 5: *Test niezmienności: Porównanie czasów odpowiedzi*

wykorzystujemy techniki związane z XML drugą młodość, zwielokrotniając ilość możliwych do obsłużenia w jednostce czasu żądań HTTP.

Porównanie ze sobą czasów uzyskania odpowiedzi na żądanie HTTP przy użyciu różnych rodzajów pamięci podręcznej wykazuje różnicę rzędu 1 ms. Fakt ten wskazuje na to, że głównym czynnikiem wpływającym na szybkość uzyskania odpowiedzi jest przygotowanie do pracy procesora XSLT, nie zaś sam proces transformacji dokumentu XML. Można więc pokusić się o rezygnację z `PPD`, uzyskując uproszczenie kodu źródłowego w Javie i pozostać tylko przy wykorzystaniu `PPS`.

Zwróćmy uwagę na to, że w kodzie źródłowym implementującym pamięć `PPS` zakładamy powiązanie dokumentu XML z dokładnie jednym stylem XSL. Ogranicza to do pewnego stopnia zastosowania tego rozwiązania. Modyfikacja tego ograniczenia mogłaby iść w kierunku możliwości wyboru stosowanego stylu XSL w zależności od np. rodzaju przeglądarki WWW, od której nadeszło żądanie HTTP, czy też preferowanego przez użytkownika tejże przeglądarki języka.

## 5. Podziękowania

Chciałbym w tym miejscu podziękować panu Tomkowi Pikowi za uwagi dotyczące implementacji w Javie serwletu używającego pamięci podręcznej oraz panu prof. dr hab. inż. Krzysztofowi Goczyle za uwagi wniesione podczas powstawania tego artykułu.

# Bibliografia


[1] World Wide Web Consortium, *Extensible Markup Language (XML) 1.0 (Second Edition)*, W3C Recommendation, 2000

[2] World Wide Web Consortium, *Extensible Stylesheet Language Transformations (XSLT) Version 1.0*, W3C Recommendation, 1999

[3] World Wide Web Consortium, *Associating Style Sheets with XML documents Version 1.0*, W3C Recommendation, 1999

[4] Sun Microsystems, *Java API for XML Processing Version 1.1*, Sun Microsystems, 2001

[5] Sun Microsystems, *Java Servlet API Specification 2.2*, Sun Microsystems, 2001

[6] B. Marchal, *Servlet Programming for Teams*, iPlanet, 1999,

[7] Eric M. Burke, *Developing, Applying and Optimizing XSLT with Java*, O'Reilly Network, 2000,

[8] Procesor XSLT Xalan, *http://xml.apache.org/xalan-j/index.html*

[9] Parser XML Xerces, *http://xml.apache.org/xerces-j/index.html*

[10] Serwer serwletów Resin, *http://www.caucho.com/*