Uniwersytet w Białymstoku

Wydział Matematyczno-Fizyczny

Instytut Matematyki

Maja Joanna Podobińska

Zastosowanie XML w opracowywaniu dokumentacji technicznej

Praca dyplomowa napisana
pod kierunkiem
dr. Jana Kowalskiego

Białystok 2005

Składam serdeczne podziękowania
dr. Janowi Kowalskiemu
za...

Maja Joanna Podobińska

Spis treści

Wstęp

Tekst wstępu do pracy. Tekst wstępu do pracy.

Rozdział 1

Wprowadzenie do XML

1.1 Co to jest XML

Pod koniec lat 90. pojawił się rozszerzalny język znaczników XML (ang.*eXtensible Markup Language*), który jest językiem do obróbki i przechowywania dokumentów elektronicznych zawierających takie elementy jak: tekst, zdjęcia, animacje, odnośniki, itp. Zadaniem tego języka jest łatwiejsze składowanie informacji, tworzenie funkcjonalnych rozszerzeń sieci WWW w nowych obszarach, do których inne języki nie nadają się. Język ten także ma pokonać barierę niekompatybilności różnych systemów komputerowych umożliwiając użytkownikom szybsze i łatwiejsze wyszukiwanie oraz wymianę danych naukowych, produktów komercyjnych i wielojęzycznych dokumentów. Jeśli mamy potrzebę zapisania określonych danych o określonej strukturze, XML okaże się najlepszym narzędziem, bez względu na to jakie te dane by nie były. W przeciwieństwie do np. HTML, XML nie ma określonej liczby znaczników, bo pozwala przechowywać dowolne dane, to najbardziej wygodny sposób, bo sami go określamy.

Sami określamy strukturę danych, która może być tabelaryczna, ale może także tworzyć drzewo. W ten sposób jesteśmy nieograniczeni. Na tym przede wszystkim polega wyższość XML nad innymi formami zapisu danych. Oddzielenie treści od formy pozwala skupić się na samych danych. Zwykle programy dzięki wspólnemu formatowi XML mogą łatwiej wymieniać dane, a informacje publikowane mogą być łatwiej przetwarzane. Weźmy na tapetę notowania spółek giełdowych. Załóżmy, że chcemy napisać program analizujący wahania kursów akcji. Potrzebne więc jest na bieżąco aktualizowanie źródła takich danych. Cóż z tego, że portali finansowych jest kilkanaście, skoro wszystkie one serwują kursy akcji w postaci tabelek HTML. Nawet gdybyśmy kosztem karkołomnej pracy przygotowali narzędzie do odczytywania kursów akcji z komórek tabelek pliku HTML, to drobna zmiana na stronach danego portalu np. niewielkie przegrupowanie danych w tabeli zrujnowałoby nasze narzędzie. Stałoby się tak dlatego, że dane w postaci HTML są przyjazne człowiekowi,

ale nie maszynie. Gdyby natomiast te dane zapisać w postaci pliku XML, zawierającego tylko właściwą treść (a nie formę), stałoby się ono łatwiejsze do indeksowania i przetwarzania. Jeśli chcielibyśmy takie dane wyświetlić jako stronę sieci Web, to przetworzenie ich do postaci czytelnej dla człowieka byłoby możliwe dzięki arkuszom stylów.

1.2 Znaczniki i podstawowe pojęcia

Ścisłe reguły XML chronią programy przed nieprzewidywalnymi danymi wejściowymi, które mogą powodować, że odmówią działania lub zaczną dawać dziwne wyniki. Stąd należy zapewnić, by dane były „czyste" i poprawne syntetycznie. Stąd też należy najpierw zrozumieć potrzebę używania parserów (analizatorów składni). Każdy program, który pracuje na danych XML najpierw musi je przeanalizować składniowo. Analiza składniowa (ang. parsing) to proces, w którym tekst XML jest zbierany i rozdzielany na oddzielne części. Znaki specjalne <,> i & mówią parserowi o tym, kiedy odczytuje znacznik, dane znakowe lub inny symbol. Jeden z rodzajów symboli noszący nazwę odwołania do encji jest symbolem zastępczym dla treści pochodzącej z innego źródła (pojedynczy znak lub ogromny plik) Analizator szuka źródła i wstawia je do dokumentu w celu przeanalizowania. Kiedy zostanie znaleziony znacznik końcowy elementu, jego nazwa jest porównywana z nazwą znacznika początkowego, umieszczonego na stosie. Jeżeli sobie odpowiadają, element zostaje zdjęty ze stosu i analiza zostaje wznowiona. W przeciwnym razie musiał pojawić się błąd i parser raportuje o nim. Narzędzie zwykle wskazuje wszystkie miejsca, w których najprawdopodobniej znajdują się błędy kodu oraz krótkie komunikaty określające, na czym polega błąd

1.2.1 Znaczniki

Znaczniki wyznaczają granicę elementów, pozwalają na wstawianie komentarzy i instrukcji specjalnych oraz deklarowanie ustawień dla środowiska analizy składniowej. Parser działa opierając się na znacznikach, które umożliwiają mu rozbicie dokumentu na oddzielne obiekty XML. Istnieje wiele różnych typów obiektów w XML.

Obiekt	**Cel**	**Przykład**
element pusty	W określonym miejscu dokumentu reprezentuje informacje	<xref linkend="abc"/ >
kontener	Grupuje elementy i dane znakowe	tekst
deklaracja	Dodaje nowy paramet, encję lub definicję gramatyczną do środowiska analizy składniowej	<!ENTITY autor "Jan Nowik">
instrukcja przetwarzania	Przekazuje specjalną instrukcję	<? cel dane?>
komentarz	wstawia uwagę, która jest ignorowana przez procesor XML	<!–komentarz–>
sekcja CDATE	Tworzona jest sekcja danych znakowych, która nie podlega analizie składniowej, zachowuje wszystkie w niej znaki specjalne	<![CDATA [znaki ampersand!&&&&&&] >
odwołanie do encji	Nakazuje parserowi wstawienie pewnego tekstu składniowego gdzie indziej	& nazwa

Tabela 1.1: Typy znaczników w XML

Elementy to najpowszechniejszy typ obiektów XML. Dzielą one dokument na coraz mniejsze komórki. Wewnątrz znaczników początkowych elementów czasem można zobaczyć pewne dodatkowe znaki obok nazwy elementu w postaci `nazwa= "wartość"` są to atrybuty, wiążą informację z elementem.

Deklaracja (ang. *declaration*) nigdy nie pojawia się wewnątrz elementu, ale mogą wystąpić na początku dokumentu lub w zewnętrznym pliku definicji typu dokument.

Kolejne trzy obiekty są używane do zmiany zachowania parsera w trakcie przetwarzania dokumentu.

Instrukcje przetwarzające (ang. *processing instructions*) jako dyrektywy programowe występują w kodzie dokumentu dla wygody (np. przechowują numer strony dla określonego narzędzia formatującego).

Komentarze (ang. *comments*) w czasie przetwarzania przez parser nie są brane pod uwagę, gdyż mają one znaczenie tylko dla autora dokumentu. *Sekcja CDATA* (ang. *CDATA sections*) to specjalne fragmenty, dla których parser pwinien tymczasowo zawiesić proces rozpoznawania znaczników.

Odwołania do encji (ang. *entity references*) nakazują parserowi, aby wstawił z góry zdefiniowaną porcję tekstu. Obiekty te nie mają postaci podobnej do innych znaczników. Zamiast nawiasów ostrych jako ograniczyków, wykorzystywane są znaki ampersand (&) oraz średnika.

1.2.2 Prolog dokumentu

Istotną rzeczą jest potraktowanie dokumentu jako jednostki logicznej, a nie fizycznej. Nie należy zakładać, że dokument będzie zawierał się w pojedynczym pliku na komputerze. Często dokument jest rozdzielony na wiele plików i niektóre mogą znajdować się w innych systemach. Jedynym wymaganiem jest aby parser XML miał możliwość powiązania poszczególnych części w jedną spójna całość.Oto przykładowy dokument XML

```
<?xml version="1.0" standalone="no"?> Deklaracja XML
<!DOCTYPE                             Początek deklaracji DOCTYPE
  przypomnienie                       Nazwa elementu korzenia
  SYSTEM "/home/reminder.dtd"         Identyfikator DTD
  [                                   Ogranicznik początkowy
                                      podzbioru wewn.
  <!ENTITY smile                      Deklaracja encji
   "<graphic file="smile.jpg"/>">
  ]>                                  Ogranicznik końcowy
                                      podzbioru wewn.

<przypomnienie>                       Początek elementu dokumentu
  &smile;                             Odwołanie do encji
                                      zadeklarowanej powyżej
  <msg>Uśmiechnij się!
  Zawsze może być gorzej.</msg>
</przypomneinie>                      Koniec elementu dokumentu
```

Deklaracja XML jest niewielkim zbiorem szczegółowych informacji, które przygotowują procesor XML do pracy z dokumentem. Jest opcjonalnym ele-

mentem dokumentu, ale w razie użycia zawsze musi znaleźć się w pierwszym wierszu.

```
<?xml parametr1 parametr2 ... ?>
```

parametr składa się z nazwy, znaku równości (=) oraz wartości ujętej w cudzysłów lup apostrof.

version -deklaruje wersję używanego standardu XML -obecnie "1.0"

encoding -definiuje kodowanie znaków użytych w dokumencie (domyślnie UTF-8)

standalone –informuje parser o tym czy poza dokumentem występują jakieś deklaracje

Wielkość liter jest istotna w użytych nazwach i wartościach parametrów. Nazwy parametrów zaczynamy małymi literami. Istotna też jest kolejność. Parametr *version* musi być określony jeśli podaje się inne parametry.

```
<?xml version='1.0' encoding='iso-8859-2' standalone='no' ?>
```

Deklaracja typu dokumentu

```
<!DOCTYPE element identyfikator DTD[ deklaracja1 ...] >
```

Identyfikator DTD obsługuje dwie metody identyfikacji: systemową i publiczną. **Identyfikator systemowy** (ang. *system identifier*) przyjmuje formę: słowo kluczowe SYSTEM, po którym występuje adres fizyczny ujęty w cudzysłów (SYSTEM "identyfikator systemowy").

```
<!DOCTYPE doc SYSTEM "/user/local/xml/dtds/try.dtd">
```

Powyższa deklaracja wskazuje plik o nazwie (try.dtd), znajdujący się w systemie lokalnym

Alternatywą identyfikatora systemowego jest **identyfikator publiczny** (ang. *public identifier*). W przeciwieństwie do ścieżki systemowej, która może się zmieniać identyfikator publiczny nigdy się nie zmienia (osoba przenosi się z miasta do miasta, ale pesel ma ten sam). Jednym z problemów jest to, że

niewiele parserów jest w stanie obsługiwać identyfikatory publiczne i nie istnieje żaden oryginalny rejestr odwzorowujący je na lokalizacje fizyczne. Nie są godne zaufania jako takie i muszą zawierać zastępczy identyfikator systemowy na wypadek wystąpienia problemów.

```
PUBLIC "identyfikator publiczny" "identyfikator systemowy"
```

Deklaracje są porcjami informacji wymaganymi do połączenia i sprawdzenia poprawności dokumentu. Parser XML w pierwszej kolejności odczytuje deklaracje z podzbioru zewnętrznego, następnie odczytuje z podzbioru wewnętrznego w kolejności, w jakiej się pojawiają. Istnieje kilka rodzaji deklaracji. Między innymi *deklaracje encji*, które tworzą nazwaną porcję kodu XML. Taka porcja może być wstawiona w dowolnym miejscu w dokumencie

```
<!ENTITY nazwa identyfikator lub wartość >
```

Fragment *identyfikator lub wartość* powoduje skojarzenie nazwy z porcją kodu XML. Ten segment XML staje się encją, która stanowi komponent dokumentu wstawiany przez parser przed rozpoczęciem analizy składniowej. Encję tę można wstawić do dokumentu za pomocą odwołania do encji; np:

```
 & nazwa;
```

1.2.3 Elementy

Kontenery to podstawowy materiał budulcowy XML, dzielący dokument na hierarchiczną strukturę. Składnia kontenera jest natępująca:

```
< nazwa atrybut1 atrybut2 ..> treść </nazwa>
```

Kontener pusty jest bardzo podobny,ponieważ nie zawiera żadnej treści, nie ma potrzeby stosowania znacznika zamykającego:

```
< nazwa atrybut1 atrybut2 .. />
```

Atrybut definiuje właściwość elementu. Kojarzy on nazwę z wartością, która jest ciągiem znaków. Atrybutów używa sie w następujący sposób:

```
nazwa="wartość"  może być nazwa='wartość'
```

nazwa może zawierać dowolny znak alfanumeryczny (a-z, A-Z oraz 0-9), znaki akcentowane, znaki z alfabetów niełacińskich (grecki, arabski). Jedyne znaki interpunkcyjne dopuszczalne w nazwie to łącznik (-), podkreślenie (_) i kropka (.). Nazwy rozpoczynamy wyłącznie od liter, ideogramu lub (_). Nazwy są zależne od wielkości liter. Nie ma ograniczeń do długości nazwy. Poprawnie sformułowany element to np.:

```
<imie> Jan </imie>
```

natomiast źle sformuowane elementy to np.:

```
<-liczba+nazwa> 54862 </-liczba+nazwa>
<1-wszyROK> W tym roku ..... </1-wszyROK>
```

Wstawianie znaków białych spacji (tabulator, nowy wiersz i spacja) w znaczniku jest dozwolone, o ile nie znajduje się między nawiasem ostrym otwierającym a nazwą elementu. Istnieje jeszcze kilka reguł dotyczących znaczników kontenerów. Nazwy podane w znaczniku początkowym i końcowym muszą być identyczne. Znacznik końcowy musi występować zawsze po (nigdy przed) znacznikiem początkowym. Oba znaczniki muszą znajdować się w obrębie tego samego elementu nadrzędnego. Nakładanie się jest naruszeniem tej reguły. Poniżej mamy przykład nakładania:

```
<a> Tak <b> wygląda </a> nakładanie się </b>
```

Poprawnie jest:

```
<a> Tak  powinno </a> <b> to wyglądać </b>
```

Atrybutów (ang. *attribute*) można używać w celu nadania elementowi unikatowych etykietek, przydzielanie ich do określonych kategorii, dodanie znaczenia logicznego lub innego krótkiego ciągu danych znakowych (chęć odróżnienia elementów o tej samej nazwie). Atrybut (class) może być używany w arkuszu stylów w celu określenia specjalnej czcionki lub koloru. Innym sposobem jest nadawanie unikatowych identyfikatorów (ciąg znaków), jednocześnie przypisanych do tego samego elementu w dokumencie. Można wskazać ten element i wykonać działania np. przytoczenie itd. Nie ma ograniczeń co do atrybutów pod warunkiem, że żadne dwa atrybuty nie posiadają tej samej nazwy. Poniżej przedstawiono przykład znacznika początkowego elementu z trzema atrybutami:

```
<płyta tytuł="Story" artysta="Tracy Chapman" wytwórnia="Elektra">
```

Poniższy przykład jest nieprawidłowy:

```
<sad owoc="jabłko"  owoc="gruszka"  owoc="śliwka">
```

W celu obejścia tego ograniczenia można użyć jednego atrybutu, który będzie przechowywał wszystkie wartości:

```
<sad owoc="jabłko gruszka  śliwka">
```

Może też być użyty atrybut o różnych nazwach:

```
<sad owoc1="jabłko"  owoc2="gruszka"  owoc3="śliwka">
```

Można również użyć dodatkowego elementu:

```
<sad>
     <owoc> jabłko  </owoc>
     <owoc> gruszka </owoc>
     <owoc> śliwka </owoc>
</sad>
```

Niektóre nazwy atrybutów są zarezerwowane w XML. Zwykle rozpoczynają się one od przedrostka „`xml`" np. `xmlns`.

1.2.4 Encje

Encje to symbole w XML do których można wielokrotnie się odwoływać. Deklarowane są w prologu dokumentu lub w definicji DTD. Różne typy encji mają różne zastosowania. Można podstawiać znaki, które sprawiają trudność lub których się nie da wprowadzić za pomocą encji znakowych. Dzięki encji zewnętrznej można wstawiać do dokumentu treść spoza niego. Ponadto zamiast wielokrotnie wpisywać ten sam składnik, na przykład pewien tekst, można definiować własne ogólne encje. Pojedyncza encja składa się z nazwy i wartości. Odwołanie się do encji składa się ze znaku ampersand (&), nazwy encji oraz znaku średnika (;). Parser XML rozpoczyna przetwarzanie dokumentu od odczytania serii *deklaracji*, w której następuje powiązanie nazw encji z ich wartościami. Wartością może być wszystko od pojedynczego znaku po plik XML. Każde napotkane przez parser odwołanie do encji powoduje, że parse sprawdza tabelę w pamięci w poszukiwaniu czegoś, czym ma zastąpić oznacznik. Zastępuje go odpowiednim tekstem lub znacznikiem a następnie wznawia analizę składniową tuż przed miejscem, w którym skończył, tak aby przeanalizować również nowo wstawiony tekst. Każde odwołanie do encji znajdujące się wewnątrz wstawionego tekstu także zostaje zastąpione. Poniżej przedstawiony został przykład dokumentu, który definiuje encje ogólne i odwołuje się do nich w tekście.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE      wiadomość SYSTEM "/wiadoność.dtd"
  [
    <!ENTITY osoba "Jan Adam Kowalewski">
    <!ENTITY agent "Anna Kowalska">
    <!ENTITY telefon "<numer> 022-444-1234</numer>">
  ]>
<wiadomość>
<wstęp> Szanowny Pan &osoba; </wstęp>
  <treść>
    Mam zaszczyt poinformować, że wygrał Pan samochód.
    Proszę zadzwonić do naszego agenta &agent; pod numer &telefon;.
```

```
  </treść>
</wiadomość>
```

Powyższy przykład w przypadku rozstrzygnięcia encji ma następującą postać:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE      wiadomość SYSTEM "/wiadoność.dtd" ]>
<wiadomość>
<wstęp> Szanowny Pan Jan Adam Kowalewski </wstęp>
  <treść>
    Mam zaszczyt poinformować, że wygrał Pan samochód.
    Proszę zadzwonić do naszego agenta Anna Kowalska pod numer
    <numer> 022-444-1234</numer> .
  </treść>
</wiadomość>
```

Encje znakowe zawierają pojedynczy znak. Dzielą się one na dwie grupy. *Predefiniowane encje znakowe* (ang. *predefined character entities*) umożliwiają bezpieczne używanie znaków, które nie mogą być używane w tekście dokumentu XML :

Encja	Wartość
amp	&
apos	’
gt	>
lt	<
quot	”

Referencje numeryczne odwołują się do znaków poprzez ich numery z zestawie znaków Unicode. Numer w nazwie encji można wyrazić w postaci liczby dziesiętnej

```
&# liczba dziesiętna ;
```

lub liczby szesnastkowej

```
 &#x liczba szesnastkowa ;
```

Encje o treści mieszanej (ang. *mixed-content entity*) posiadają wartość o nieograniczonej długości i mogą zawierać znaki i tekst. Encje te dzielą się na dwie kategorie: wewnętrzne i zewnętrzne. W przypadku *encji wewnętrznych* (ang. *internal entity*) tekst zastępujący jest definiowany w deklaracji encji.

```
    <!ENTITY osoba "Jan Adam Kowalewski">
```

Są one najczęściej używane w celu zastępowania powtarzanych fraz i nazw. Zwiększa to dodatkowo dokładność i możliwości konserwacji kodu, gdyż w razie konieczności zmiany, należy jej dokonać tylko w jednym miejscu. W przypadku *encji zewnętrznych* (ang. *external entity*) tekst zastępujący znajduje się w innym pliku.

```
<!ENTITY osoba SYSTEM "plik.xml">
```

Są one przydatne w przypadku importowania treści, która jest dzielona przez wiele dokumentów lub która zmienia się zbyt często. Umożliwiają również dużych, monolitycznych dokumentów na mniejsze części, które można edytować zespołowo i które są szybciej przesyłane siecią.
Encje niepoddawane analizie składniowej (ang. *unparsed entity*) zawierają treść , która nie powinna być analizowana. Są one używane w celu importowania grafiki, dźwięku oraz innych danych nieznakowych. Deklaracja encji nieanalizowanej wygląda podobnie do encji zewnętrznej z pewnymi dodatkowymi informacjami podawanymi na końcu.

```
<!DOCTYPE doc [
    <!ENTITY logo SYSTEM "logo.gif" NDATA GIF>
]>
<dokument>
   <grafic src="&logo;"/>
   Oto oferta naszej firmy ...
</dokument>
```

Po informacji o ścieżce systemowej występuje słowo kluczowe `NDATA`. Informuje ono parser, że treść encji ma specjalną format lub notację. Po słowie kluczowym występuje *identyfikator notacji*, który określa format danych.

1.2.5 Inne znaczniki

Dodatkowymi elementami zbioru znaczników są komentarze, instrukcje przetwarzania oraz sekcje CDATA. Wszystkie posiadają wspólną cechę: w pewien sposób chronia swoją swoją zawartość przed parserem.
Komentarze to notatki umieszczone w dokumencie, które nie są przetwarzane przez procesor XML. Jeżeli kilka osób pracuje na tych samych danych plikach, wiadomości te mogą mieć nieocenioną wartość. Mogą być używane w celu identyfikowania tworzenia plików i sekcji, w celu wspomożenia nawigacji po skomplikowanym dokumencie lub po prostu w celu komunikowania się z innymi użytkownikami. Przykładowy komentarz może wyglądać tak:

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

Komentarze mogą występować w dowolnym miejscu oprócz sekcji przed deklaracją XML oraz wewnątrz znaczników. Mogą za to zawierać znaczniki, więc

mogą być używane do „wyłączania" fragmentów dokumentu. Używając tej techniki należy uważać aby nie komentować innych komentarzy. Ponieważ parser będzie informował o błędzie w momencie dotarcia do komentarza wewnętrznego.

Sekcja CDATA jest miejscem, które może zawierać znaki zabronione. Parser jest informowany, że dana sekcja dokumentu nie zawiera kodu znaczników i powinna być traktowane jako zwykły tekst. Jeżeli często oznacza się znaki w tekście, ciągłe używanie predefiniowanych encji `<` `>` i `&` może okazać się uciążliwym zadaniem oraz trudno jest je czytaćć

```
<![CDATA[ nieanalizowane składniowo dane znakowe  ]] >
```

Jedyny element który nie może zawierać w sekcji CDATA jest ogranicznik zamykający `]]>`. Sekcje CDATA nie mogą być zagnieżdżane oraz nie ma możliwości użycia elementów lub atrybutów wewnątrz oznaczonego tekstu. Można to ominąć stosując odwołanie do encji zewnętrznej lub samych encji. Przykładowe użycie sekcji CDATA:

```
   <dokument> wówczas stwierdzamy "<![CDATA[
  jeżeli (  &x < &y )]] >", co kończy sprawę.</dokument>
```

Ten sam efekt można uzyskać w następujący sposób:

```
   <dokument> wówczas stwierdzamy "jeżeli
( &amp;x  &lt; &amp;y )", co kończy sprawę.</dokument>
```

Instrukcja przetwarzania zawiera dane, które są przeznaczone dla określonego procesora XML (np. gdy trzeba przechowywać numer strony). Instrukcja zawiera dwie informacje: słowo kluczowe celu oraz pewne dane. Jeżeli program obsługujący instrukcję przetwarzania rozpozna słowo kluczowe, może użyć danych w przeciwnym razie dane są ignorowane. Składnia instrukcji przetwarzania jest następująca:

```
<?  cel dana  ?>
```

Cel to słowo kluczowe, którego procesor XML używa w celu określenia tego, czy dane są przeznaczone dla niego, czy nie. Słowo kluczowe nie musi nic znaczy, na przykład nie musi to być nazwa programu , który będzie korzystał z danych. Instrukcja przetwarzania może używać więcej niż jeden program, a jeden program może akceptować wiele instrukcji. Dane zawarte w instrukcji mogą być dowolne z wyjątkiem kombinacji znaków `?>`, które są ogranicznikiem zamykającym.

```
<? coś ?>
<? print-formatter forcelinebreak?>
```

Jeżeli nie określono ciągu danych, samo słowo kluczowe celu może pełnić funkcję danych. Przykładem może być instrukcja łamania wiersza `<?lb?>`

Rozdział 2

XML i XSL

Transformacja to jedno z najważniejszych i najbardziej przydatnych technik pracy z kodem XML. **Transformacja** (ang. *transformation*) kodu XML polega na zmianie jego struktury, jego znaczników a czasem również jego treści do innej postaci. Przekształcony dokument może zostać zmieniony tylko w niewielkim lub w bardzo dużym stopniu. Cały proces bazuje na starannie przygotowanym dokumencie, noszącym nazwę arkusza stylów lub skryptu transformacji. Istnieje wiele powodów dokonywania transformacji XML. Najczęściej ma to na celu umożliwienie wykorzystania dokumentu w nowych obszarach, poprzez przekonwertowanie go do formatu prezentacyjnego. Można również wykorzystać transformację do zmiany treści. Niektóre zastosowania transformacji:

- zmiana aplikacji nieprezentacyjnej, takiej jak DocBook, na postać kodu HTML w celu wyświetlenia dokumentu jako strony WWW;
- sformatowanie dokumentu w celu utworzeniu formatu prezentacyjnego wysokiej jakości, na przykład PDF, poprzez ścieżkę XSL-FO;
- zamiana jednego słownictwa XML na inne, na przykład przekształcenie faktury z formatu stosowanego w danym przedsiębiorstwie do formatu powszechnie stosowanego w strefach biznesowych;
- wydobycie określonej porcji informacji i sformatowanie ich w inny sposób, na przykład skonstruowanie tabeli stałych na podstawie tytułów sekcji;
- zmiana realizacji XML na postać tekstową, na przykład przekształcenie pliku danych XML na plik z polami oddzielonymi przecinkami, który można zaimportować do programu Excel jako arkusz kalkulacyjny;
- przeformatowanie lub wygenerowanie treści. Przykładowo, wartości liczbowe mogą zostać zamienione z postaci całkowitoliczbowej na zmiennoprzecinkową lub na liczby rzymskie w celu utworzenia list numerowanych lub nagłówków sekcji;

- korekta dokumentu w celu poprawienia często występujących błędów lub usunięcia niepotrzebnych znaczników, co stanowi działanie przygotowawcze do dalszych prac.

Transformacje oferują wielkie możliwości, stanowią potężny, mimo to niezbyt skomplikowany mechanizm jeszcze wydajniejszego użycia kodu XML.

2.1 Język XSL

W momencie gdy XML zyskał na popularności, jego pierwsi użytkownicy i programiści zaczęli opracowywać zasady formatowania wysokiej jakości. Po bliższym przyjrzeniu się językowi DSSSL (*Dokument Style Sementics and Specification Language*), okazało się, że charakteryzują go te same problemy, co w przypadku języka SGML (*Standard Generalized Markup Language*) : był zbyt obszerny, zbyt trudny w nauce i niełatwy w implementacji. James Clarke, jeden z pionierów technik przetwarzania tekstu, który współuczestniczył w opracowywaniu języka DSSSL, postanowił wykorzystać zdobyte doświadczenia i rozpoczął prace nad nieco odchudzoną wersją nowego języka. W ten sposób powstał rozszerzalny język arkuszy styli XSL. Język XSL w rzeczywistości stanowi połączenie trzech technik :

- XPath, który służy do znajdowania i obsługi komponentów dokumentu;
- XSL Transformations, służący do przekształcania dowolnego XML do postaci prezentacyjnego drzewa XSL Formatting Object;
- XSL Formatting Objects, język znaczników służący do wysokiej jakości formatowania tekstu.

2.1.1 XPath

Przy niewielkiej znajomości języka znaczników można lokalizować i dotrzeć do każdej porcji informacji. Jest to przydatne gdy jest konieczne zlokalizowanie określonych danych ze znanej lokalizacji (zwanej *ścieżką*) w danym dokumencie. Inna korzyść polega na tym, że można wykorzystać informacje o ścieżce w celu bardzo szczegółowego określenia charakteru przetwarzania całej klasy dokumentów. Zamiast po prostu podawać nazwę elementu lub wartości atrybutu w celu opracowania arkusza styli można użyć wszelkiego rodzaju dodatkowych szczegółów kontekstowych, w tym danych znajdujących się w dowolnym miejscu w dokumencie. W celu wyrażenia informacji o ścieżkach w unormowany sposób organizacja W3C zaleca użycie XML Path Language (zwanego również jako XPath). Każdy dokument XML posiada reprezentację graficzną w postaci struktury drzewiastej. Z uwagi na fakt, że istnieje tylko jedna możliwa konfiguracja drzewa dla dowolnego dokumentu, istnieje unikatowa ścieżka z korzenia (lub dowolnego wierzchołka wewnętrznego) do dowolnego innego

punktu. Język XPath opisuje po prostu, w jaki sposób dotrzeć do miejsca przeznaczenia. Każdy krok na ścieżce jest związany z rozgałęziającym się lub końcowym punktem w drzewie, noszącym nazwę `wierzchołka` (ang. *node*). W przypadku języka XPath istnieje siedem różnych rodzajów wierzchołków:

- **korzeń** to wierzchołek zawierający element dokumentu, zawiera również wszelkie komentarze i instrukcje przetwarzania;
- **element** może zawierać inne wierzchołki. W drzewie jest to punkt, w którym spotykają się dwie gałęzie;
- **atrybuty** są traktowane jako wierzchołki oddzielone od elementów. Pozwala to na wybranie elementu jako całości lub tylko atrybutu z tego elementu przy użyciu tej samej składni ścieżki;
- **tekst** jest obszarem nie przerwanego tekstu. Element może jednak posiadać więcej niż jeden wierzchołek tekstowy, jeśli jest rozdzielony elementami lub wierzchołkami innego rodzaju;
- **komentarz** jest przez większość procesorów XML odrzucane, są one traktowane jako poprawne wierzchołki;
- **instrukcja przetwarzania** mogą występować w dowolnym miejscu dokumentu pod wierzchołkiem korzenia;
- **przestrzeń nazw** stanowi fragment dokumentu, a nie tylko określa posiadania jednego elementu.

Ścieżka lokalizacji (ang. *location path*) to łańcuch *kroków lokalizacji* (ang. *location steps*), które pozwalają dotrzeć z jednego punktu dokumentu do drugiego. Krok lokalizacji posiada trzy części: *oś* (ang. *axis*), która opisuje kierunek przemieszczenia się, *test wierzchołka* (ang. *node test*), który określa, jakiego rodzaju wierzchołki są istotne oraz zestaw opcjonalnych *predykatów* (ang. *predicates*), które wykorzystują testy logiczne (zwracające wartość prawda-fałsz) w celu jeszcze dokładniejszego zbadania wierzchołków kandydujących. Osią jest słowo kluczowe, które określa kierunek przemieszczania się z dowolnego wierzchołka. Po osi występuje parametr testu wierzchołka, połączony z osią dwoma znakami dwukropka (::). W tabeli 2.1.1 wymieniono testy wierzchołków:

Warunek	**Dopasowanie**
/	wierzchołek korzenia zawierające wszelkie komentarze lub instrukcje poprzedzające go
node()	każdy wierzchołek. Przykładowo, krok `attribute::node()` wybrałby wszystkie atrybuty wierzchołka kontekstowego
*	w przypadku osi atrybutów-każdy atrybut, osi przestrzeni nazw-kążda przestrzeń nazw, innych osi- dowolny element
list	w przypadku osi atrybutów-atrybut o nazwie *lista* wierzchołka kontekstowego, osi przestrzeni nazw-przestrzeń nazw o nazwie *lista*, innych przypadkach- każdy element o nazwie *lista*
text()	dowolny wierzchołek testowy
processing-instruction()	dowolna instrukcja przetwarzania
processing-instruction('do-listy')	dowolna instrukcja przetwarzania, której celem jest *do-listy*
comment()	dowolny komentarz

Tabela 2.1: Testy wierzchołków

Kroki lokalizacji są łączone ze sobą za pomocą znaku ukośnika (/). Każdy krok przybliża do wierzchołka, który chce się zlokalizować. Przykładowo, w celu dostania się z wierzchołka korzeń do elementu *tabela*, znajdującego się wewnątrz elementu *podrozdział* wewnątrz elementu *rozdział* wewnątrz elementu *książka* można by zapisać następującą ścieżkę:

```
książka/rozdział/podrozdział/tabela
```

Taka składnia może być zbyt rozwlekła, jednak XPath oferuje pewne przydatne skróty, które wymieniono w tabeli 2.1.1.

Wzorzec	**Dopasowanie**
@książka	odpowiada atrybutowi o nazwie *książka*. Jest to równoważne zapisowi *attribute::książka*
.	wierzchołek kontekstowy. Jest to równoważne zapisowi *self::node()*
/*	odpowiada elementowi dokumentu
parent::/*/following-sibling::rok	odpowiada wszystkim elementom *rok*, które występują po rodzicu wierzchołka kontekstowego
..	odpowiada wierzchołkowi rodzica. Dwie kropki (..) to skrócona forma *parent::node()*
.//rola	odpowiada dowolnemu elementowi typu *rola*, który jest potomkiem wierzchołka bieżącego
//rola	odpowiada każdemu elementowi $<$ *rola* $>$, który jest potomkiem wierzchołka korzenia
../*	odpowiada wszystkim wierzchołkom siostrzanym

Tabela 2.2: Skróty lokalizacji ścieżek

Aby zobaczyć, w jaki sposób można używać osi oraz testów wierzchołków w celu docierania do wierzchołków, warto przeanalizować poniższy przykład.

```
<lista-cytatów>
  <cytat rodzaj="łacińskie" nr="c1">
    <tekst>Po ich owocach ich poznacie.</tekst>
    <źródło>Jezus Chrystus</źródło>
  </cytat>
  <cytat rodzaj="polityczne" nr="c2">
    <tekst> Człowiek jest dla człowieka pierwszą rzeczywistością.</tekst>
    <źródło>Martin Luter</źródło>
   </cytat>
  <cytat rodzaj="głupawe" nr="c3">
    <?śmiech?>
    <tekst>Mężczyzna ma tyle lat, na ile się czuje,
           kobieta tyle, na ile wygląda. </tekst>
  </cytat>
```

```
  <cytat rodzaj="przyjacielskie" nr="c4">
    <tekst>u przyjaciół wszystko jest wspólne.  </tekst>
    <źródło>Pitagoras</źródło>
  </cytat>
  <!-- jakiś komentarz? -->
  <cytat rodzaj="polityczne" nr="c5">
    <tekst>Nic nie jest niezmienne prócz przyrodzonych
           i niezbywalnych praw człowieka.</tekst>
    <źródło>Thomas Jefferson</źródło>
  </cytat>
</lista-cytatów>
```

W tabeli 2.1.1 zawarto pewne ścieżki lokalizacji oraz zwracane przez nie wartości.

Ścieżka	**Dopasowanie**
/lista-cytatów/child::node()	Wszystkie elementy *cytat* oraz komentarz XML
/lista-cytatów/cytat	wszystkie elementy *cytat*
/*/*	wszystkie elementy *cytat*
//comment()/following-sibling::*/@rodzaj	atrybut *rodzaj* ostatniego elementu
nr('c1')/parent::	pierwszy element *cytat*
nr('c4')/..	element dokumentu
nr('c1')/ancestor-or-self::*	element dokumentu oraz pierwszy element *cytat*
nr('c3')self::aforyzm	Nic. Pierwszy krok powoduje dopasowanie do trzeciego elementu *cytat* ale kolejny krok to zmienia, gdyz poszukuje elementy typu *aforyzm*
//processing-instrution()/../following::źródło	elementy *źródło* ostatnich dwóch elementów *cytat*

Tabela 2.3: Przyklady ścieżek lokalizacji

Jeżeli oś i typ wierzchołka nie wystarczają do ograniczenia wyboru, można użyć **predykatorów**. Predykator to wyrażenie logiczne ujęte w nawiasy kwadratowe ([]). Każdy wierzchołek, który poprawnie przechodzi ten test zostaje uwzględniony w wynikowym zbiorze wierzchołków. Wierzchołki, które nie przechodzą testu (wyliczona wartość predykatu jest fałsz) są pomijane. W tabeli 2.1.1 zawarte niektóre przykłady.

Ścieżka	**Dopasowanie**
// cytat[@nr="c3"]/tekst	element tekstowy w trzecim elemencie *cytat*
// cytat[źródło]	wszystkie elementy *cytat* oprócz trzeciego, który nie posiada elementu *źródło*
// cytat[not(źródło)]	trzeci element *cytat*. Funkcja *not()* zwróci wartość prawdy, gdy nie wystąpi element *źródło*.
/*[@inr="c5"]/preceding-sibling::*/źródło	element *źródło* z treścią „Thomas Jefferson"
//*[źródło=' Thomas Jefferson'][@nr='c6']	nic. Wyliczana wartość to iloczyn logiczny *and* obu predykatów
/*/*[position()=last()]	ostatni element *cytat*. Funkcja *position()* zwraca pozycje ostatniego kroku wśród adekwatnych kandydatów. Funkcja *last()* zwraca całkowitą liczbę kandydata.
//cytat[position() != 2]	wszystkie elementy *cytat* oprócz drugiego
//cytat[4]	czwarty element *cytat*
//cytat[@rodzaj=' polityczne' or @rodzaj=' łacińskie']	pierwszy, drugi oraz piąty element *cytat*

Tabela 2.4: Predykaty języka XPath

Ścieżki lokalizacji stanowią podzbiór bardziej ogólnej koncepcji wyrażeń XPath. Są to instrukcje , które umożliwiają wydobywanie przydatnych informacji z drzewa. Zamiast po prostu znajdować wierzchołki można je zliczać, dodawać wartości numeryczne, porównywać ciągi znaków i wykonywać inne

działania. Istnieje pięć typów wyrażeń :

- `logiczne` wyrażenia o dwóch możliwych wartościach: *true* i *false*,
- `zbiór wierzcholków` to kolekcja wierzchołków, które odpowiadają kryteriom wyrażenia,
- `liczbowe` to wartość liczbowe przydatne do zliczania wierzchołków i wykonywania prostych operacji arytmetycznych,
- `ciąg znaków` to fragment tekstu, który może pochodzić z drzewa wejściowego, przetworzonego,
- `wynikowe drzewo częściowe` inaczej tymczasowe drzewo wierzchołków, które posiada własny wierzchołek korzenia ale nie może być indeksowane do wykorzystania przez ścieżki lokalizacji.

Wyrażenia logiczne W języku XPath typy są determinowane przez kontekst. Operator lub funkcja może przekształcać jeden typ wyrażenia do drugiego w miarę potrzeby. Pewne operacje (wymienione w tabeli 2.1.1) służą do porównywania wartości numerycznych, czego wynikiem są wyrażenia logiczne. S

Operator
wyr = wyr
wyr != wyr
wyr < wyr
wyr > wyr
wyr <=wyr
wyr >=wyr

Tabela 2.5: Operatory porównania

ą to *porównania szczegółowe*, co oznacza, że testują one wszystkie wierzchołki danego zbioru wierzchołków w celu określenia, czy któryś z nich spełnia warunek porównania. W tabeli 2.1.1 zawarto funkcje, które zwracają wartości logiczne.

Funkcja
wyr and wyr
wyr or wyr
true()
false()
not(wyr)

Tabela 2.6: Funkcje logiczne

Wyrażenia liczbowe
XPath dopuszcza numeryczne wyliczania wartości wyrażenia, co jest przydatne w przypadku porównania pozycji w zbiorze, dodawanie wartości elementów numerycznych, zwiększania liczników i tak dalej.

Funkcja	**Zwracana wartość**
count(zbior_wierzch)	liczba elementów w zbiorze *zbior_wierzch*
generate-id(zbior_wierzch)	ciag znaków zawierających unikatowy identyfikator dla pierwszego wierzchołka w zbiorze *zbior_wierzch* lub dla wierzchołka kontekstowego, jeżeli pominie się argument wywołania
last()	numer ostatniego wierzchołka w kontekstowym zbiorze wierzchołków
local-name(zbior_wierzch)	nazwa pierwszego wierzchołka w zbiorze *zbior_wierzch* bez prefiksu przestrzeni nazw. W przypadku pominięcia argumentu wywołania zwracana jest lokalna nazwa wierzchołka kontekstowego
name(zbior_wierzch)	adres URI przestrzeni nazw dla pierwszego wierzchołka w zbiorze *zbior_wierzch* wraz z prefiksem przestrzeni nazw
namespace-uri(zbior_wierzch)	adres URI przestrzeni nazw dla pierwszego wierzchołka w zbiorze *zbior_wierzch*
position()	numer wierzchołka kontekstowego w kontekstowym zbiorze wierzchołków

Tabela 2.7: Funkcje zbioru wierzcholków

W celu manipulowania wartościami liczbowymi można korzystać z wielu operatorów i funkcji. Wymieniono je w tabeli 2.1.1

Funkcja	Zwracana wartość
wyr + wyr	suma dwóch wyrażeń liczbowych
wyr – wyr	różnica dwóch wyrażeń liczbowych
wyr * wyr	iloczyn dwóch wyrażeń liczbowych
wyr div wyr	iloraz dwóch wyrażeń liczbowych
wyr mod wyr	reszta z dzielenia pierwszego wyrażenia liczbowego przez drugie
round(wyr)	wartość wyrażenia zaokrąglenia do najbliższej liczby całkowitej
floor(wyr)	wartość wyrażenia zaokrąglenia w dół do najbliższej liczby całkowitej
ceiling(wyr)	wartość wyrażenia zaokrąglenia w górę do najbliższej liczby całkowitej
sum(zbior_wierzch)	suma wartości wierzchołków za *zbior_wierzch*

Tabela 2.8: Operatory i funkcje liczbowe

2.1.2 XSL Transformations

Na wejściu procesor XSLT (inaczej motor XSLT) pobiera dwa elementy : arkusz styli XSLT, który kieruje i kontroluje proces transformacji, oraz dokument wejściowy (*drzewo wejściowe*). Na wyjściu otrzymujemy dokument wyjściowy (*drzewo wynikowe*). Procesor XSLT to maszyna stanów. Na stan składają się zdefiniowane zmienne oraz zbiór *wierzchołków kontekstowych*, czyli wierzchołków, które są kolejnymi, jakie mają zostać poddane przetwarzaniu. Cały proces ma charakter rekurencyjny, co oznacza, ze w przypadku każdego przetwarzanego wierzchołka mogą istnieć jego potomki, które również wymagają przetworzenia. W takim przypadku bieżący zbiór wierzchołków kontekstowych zostaje chwilowo zawieszony do momentu zakończenia wywołania rekurencyjnego. Rozpoczynając od wierzchołka korzenia motor XSLT znajduje reguły, wykonuje je i kontynuuje działanie do momentu, gdy zabraknie więcej wierzchołków w zbiorze wierzchołków kontekstowych. W tym momencie przetwarzanie zostaje zakończone a motor XSLT przekazuje na wyjście dokument wynikowy. W dodatku A przykład A.1 umieszczono kod XML[1]. Dokument sformatujemy do postaci kodu HTML, przy użyciu transformacji XSLT. Proces rozpocznie się od elementu `podręcznik`, który zostanie określony jako „powłoka" dokumentu (element `html`, tytuł oraz metadane). Z `lista-części` utworzona zostanie lista pozycji. Z każdej `części` z atrybutem `etykieta` powstanie element `li` na liście. Natomiast każda `część` z atrybutem `ref` przekaże sam tekst: etykieta i inny tekst. Element `instrukcja` stanowi listę numeryczną. Na wyjście zostanie przekazany jej element kontenera. Każda pozycja z listy instrukcji powstanie z elementu `etap`. Arkusz stylów z dodatku A przy-

[1]przykład pochodzi z książki "XML. Wprowadzenie.Wydanie II" Erik T.Ray

kład A.2 stanowi realizację powyższych reguł, gdzie dla każdej z nich jest tworzony szablon (ang. *template*).

Każda reguła opisu słownego posiada odpowiedni element `<template>`, zawierający poprawnie sformatowany fragment kodu XML. Przestrzenie nazw pomagają procesorowi w odróżnieniu instrukcji XSLT od zbioru znaczników, które mają się pojawić w wyjściowym drzewie wynikowym. W przypadku opisywanym instrukcje XSLT są elementami, które posiadają prefiks przestrzeni nazw `xsl`. Atrybut `match` w każdym elemencie `template` przypisuje go do fragmentu drzewa źródłowego za pomocą `wzorca XSLT`, który bazuje na języku XPath. `Szablon` to połączone znaczniki, treść tekstowa oraz instrukcje XSLT. Dane instrukcje mogą być warunkowymi (jeżeli warunek jest spełniony to dany element jest przekazywany na wyjście), funkcjami formatowania treści lub instrukcjami, które przekazuje przetwarzanie do innych wierzchołków. Przykładem może być element `apply-templates`, który informuje motor XSLT, że przetwarzanie należy przekierować do nowego zbioru wierzchołków kontekstowych-potomków bieżącego wierzchołka. Sformatowana strona w języku HTML (kod znajduje się w dodatku A przyklad A.3) będzie wynikiem uruchomionej transformacji powyższego dokumentu oraz arkuszu styli XSLT (bez dokładności do białych znaków). Plik dostępny jest również pod adresem `http://theta.uwb.edu.pl/ mpod`. Widać na tym przykładzie jak elementy drzewa źródłowego zostały odwzorowane na różne elementy drzewa wynikowego. Język XSLT opracowano początkowo z myślą o tworzeniu sformatowanych dokumentów możliwych do odczytywania przez użytkowników, a nie jako narzędzie służące do przeprowadzania ogólnych transformacji. Stąd też obsługuje on wiele opcji formatowania. Globalnym ustawieniem , które można zawrzeć w arkuszu stylów jest element `output`. Steruje on sposobem konstruowania przez motor XSLT drzewa wynikowego poprzez wymuszenie znacznika początkowego i końcowego, obsługę białych znaków w określony sposób itd. Jest to element wysokopoziomowy, który powinien znajdować się poza jakimkolwiek szablonem. Dostępne są trzy opcje: `xml`,`html`oraz `text`. Jeżeli dokument wynikowy będzie aplikacją XML, w arkuszu stylów należy umieścić następującą dyrektywę:

```
<xsl:output metod="xml"/>
```

Elementem dokumentu jest `stylesheet`, chodć można również użyć elementu `transform`. Element ten jest miejscem, w którym należy zadeklarować przestrzeń nazw i wersję języka XSLT. Identyfikator przestrzeni nazw ma postać

```
http://www.w3c.org/1999/XSL/Transform.
```

Zarówno atrybut przestrzeni nazw, jak i wersja są wymagane. Istnieje możliwość rozszerzenia XSLT, aby można było wykonywać specjalne funkcje, a nie

zostały one zawarte w specyfikacji. Przykładem może być funkcja do przekierowania danych wejściowych do wielu plików. Rozszerzenia takie są identyfikowane przez odrębne przestrzenie nazw, które musza być zadeklarowane jeśli chcemy z nich korzystać. Dodatkowo należy ustawić atrybut `extension-element-prefixes`, tak aby zawierał prefiksy przestrzeni nazw rozszerzalnych. Przykładową deklarację elementy `stylesheet` mamy poniżej.

```
<xsl:stylesheet
xmlns:xsl="http://www.w3c.org/1999/XSL/Transform"
xmlns:czc="http://www.mojxslt.org.pl/czcionki"
extension-element-prefixes ="czc"
version=""1.0
>
```

Widać tu deklarację dla elementów sterujących XSLT (prefiks `xsl`) oraz elementów specyficznych dla implementacji (prefiks `czc`), na końcu nastąpiła określona wersja 1.0 języka XSLT. Przestrzeń nazw, reprezentowana jako `xsl`, jest używana przez procesor transformacji do określenia, które elementy sterują procesem. Każdy element lub atrybut nie zawarty w tej przestrzeni nazw rozszerzeń będzie interpretowany jako dane, które mają się znaleźć w drzewie wynikowym.

Arkusze stylów XSLT stanowią kolekcje szablonów. Szablon kojarzy warunek z połączeniem danych wyjściowych i instrukcji. Instrukcje zwiększają czytelność i umożliwiają przekierowywanie przetwarzania, rozszerzając prosty mechanizm dopasowywania , tak aby umożliwić użytkownikowi pełna kontrolę nad transmisją każdy szablon odpowiada za wykonanie trzech działań:

1. dopasowuje (ang. *match*) klasę wierzchołków. Atrybut `match` zawiera wzorzec XSLT, który podobnie jak wyrażenie XPath, dopasowuje wierzchołki.

2. szablon określa wartość priorytetu, która pomaga procesorowi w podjęciu decyzji, który spośród dostępnych szablonów będzie najlepszy. Szablon, który odpowiada bieżącemu wierzchołkowi z największą wartością priorytetu jest tym, który zostanie użyty do przetworzenia wierzchołka. Szablony o większej ilości szczegółowych informacji przewyższa pod tym względem taki, który jest bardziej ogólny. Można po prostu określać procesorowi swój priorytet za pomocą atrybutu `priority`. Jest to przydatne w przypadku, gdy chce się wymusić użycie szablonu w sytuacji, kiedy zostałby pominięty.

3. określanie struktury drzewa wynikowego. Treść szablonu zawiera elementy i dane znakowe, które mają zostać przekazane na wyjście w drzewie wynikowym.

Taki model opisu transformacji ma ogromne zalety. Szablony są zwartymi fragmentami kodu, które łatwo odczytywać i którymi łatwo zarządzać. Dokładne określenie, kiedy każdy szablon ma być użyty jest możliwe dzięki atrybutom `match` oraz `priority`. Arkusze stylów transformacji są modularne i mogą być ze sobą łączone w celu poprawienia lub zmodyfikowania przepływu transformacji.

Język XSLT definiuje zestaw reguł, które ułatwiają pisanie arkuszy stylów. Jeżeli nie da się dopasować żadnej reguły z arkusza stylów, domyślne reguły stanowią „system awaryjny". Ich ogólne zachowanie polega na przekazywaniu wszelkich danych znakowych w elementach z drzewa źródłowego do drzewa wynikowego oraz na przyjęciu przez domyślmy element `xsl:apply-templates` możliwości przetwarzania rekurencyjnego. Atrybuty nie posiadające dopasowanych szablonów nie są przetwarzane.
Model wzorców transformacji tworzy oddzielne obszary znaczników. Potrzebny jest pewien sposób ich połączenia, tak aby przetwarzanie było kontynuowane przez cały dokument. Zgodnie z regułami domyślnymi, dla każdego elementu, który nie posiada szablonu, motor XSLT powinien przekazać jego wartość tekstową na wyjście. Wymaga to przetworzenia nie tylko wierzchołków tekstowych, ale również wszystkich potomków w przypadku, gdy również zawierają wartości tekstowe.
Jeżeli szablon odpowiada pewnemu elementowi, nie jest wymagane wykonywanie jakichkolwiek działań na elemencie lub jego treści. W rzeczywistości często jest pożądane, aby pewne elementy były ignorowane. Być może zawierają one metadane, które nie powinny być zawarte wśród sformatowanych danych. Dozwolone jest więc pozostawienie szablonu pustego. W poniższym przykładzie element `zeszyt` zostanie przekazany dalej przez procesor XSLT bez zmian (chyba że będzie pasować do niego inna reguła o wyższym priorytecie):

```
<xsl:template match="zeszyt"/>
```

Element `apply-templates` powoduje przerwanie bieżącego procesu przetwarzania w szablonie i wymuszanie na motorze XSLT przejścia do potomków bieżącego wierzchołka. Umożliwia to działanie rekurencyjne, tak aby proces przetwarzania był kontynuowany w dół drzewa dokumentu. Nosi on nazwe `apply-templates` (zastosuj szablon), ponieważ procesor musi znaleźć nowy szablon, służący do przetwarzania wierzchołków potomnych.

[2.1] apply-templates

Przykład 2.1.

```
<xsl:template match="podręcznik">
  <html>
     <head><title>Instrukcja użytkownika</title></head>
     <body >
       <h1>Instrukcja użytkownika</h1>
        <xsl:apply-templates/>
     </body>
  </html>
 </xsl:template>
```

W czasie przetwarzania tego szablonu motor XSLT najpierw przekazuje na wyjście zbiór znaczników, poczynając od znacznika początkowego `html` i kończąc na znaczniku końcowym elementu `h1`. W momencie dotarcia do elementu `xsl:apply-templates` następuje przejście do potomków bieżącego elementu (`podręcznik`) i ich przetworzenia przy użyciu ich własnych szablonów: atrybuty `typ` i `id`, następnie elementy `lista-części` oraz `instrukcje`. Po przetworzeniu ich wszystkich motor XSLT powraca do swojej pracy związanej z powyższym szablonem i przekazuje na wyjście znaczniki końcowe elementów `body` oraz `html`.
Można ograniczyć zbiór potomków podlegających przetwarzaniu za pomocą atrybutu `select`. Jako wartość pobiera on ścieżkę lokalizacji XPath, co daje ogromne możliwości wyboru. Przykładowo, można by przepisać przykład 2.1 w następujący sposób :

```
<xsl:template match="podręcznik">
  <html>
     <head><title>Instrukcja użytkownika</title></head>
     <body >
       <h1>Instrukcja użytkownika</h1>
        <xsl:apply-templates select="lista-części"/>
     </body>
  </html>
 </xsl:template>
```

Teraz tylko element `lista-części` zostanie przetworzony. Wszystkie inne potomki elementu `podręcznik`, w tym jego atrybuty oraz element `instrukcje` zostaną pominięte. Można również pominąć elementy określonego typu, na przykład:

```
<xsl:template match="podręcznik">
  <html>
     <head><title>Instrukcja użytkownika</title></head>
     <body >
       <h1>Instrukcja użytkownika</h1>
        <xsl:apply-templates select="not(lista-części)"/>
     </body>
  </html>
 </xsl:template>
```

W tym przypadku zostanie obsłużone wszystkie elementy oprócz elementu `lista-części`.

Element `for-each` tworzy szablon w szablonie. Zamiast polegać na motorze XSLT w kwestii znalezienia pasujących szablonów, dyrektywa obejmuje własny obszar znaczników. Wewnątrz tego obszaru zbiór wierzchołków kontekstowych jest predefiniowany do postaci innego zbioru wierzchołków, określonego przez atrybut `select`. W momencie wyjścia poza element `for-each` jest przywracany poprzedni zbiór wierzchołków. Rozważmy fragment kodu XML:

```
<książka>
  <tytuł>Bajki</tytuł>
    ...
<rozdział>
     <tytuł> Królewna Śnieżka</tytuł>
    ...
</rozdział>
<rozdział >
       <tytuł> Brzydkie kacząтko</tytuł>
    ...
</rozdział>
<rozdział >
       <tytuł> Kopciuszek</tytuł>
    ...
</rozdział>
</książka>
```

oraz następujący szablon:

```
<xsl:template match="książka">
<xsl:for-each select="rozdział">
<xsl:text> Rozdział </xsl:text>
<xsl:value-of  select="position()"/>
<xsl:text> . </xsl:text>
<xsl:value-of  select="tytuł"/>
<xsl:text>
</xsl:text>
     </xsl:for-each>
     </xsl:apply-templates/>
 </xsl:template>
```

Powoduje on utworzenie spisu treści dla powyższego dokumentu XML. Element `for-each` powoduje przejście przez element `książka` i pobranie każdego elementu potomnego typu `rozdział`. Ten zbiór staje się nowym zbiorem wierzchołków kontekstowych i w ramach instrukcji `for-each` nie wiadomo nic o starszych wierzchołkach kontekstowych. Pierwszy element `value-of` przekazuje na wyjście ciąg znaków zwrócony przez wyrażenie XPath `position()`, które jest pozycją rozdziału w rozpatrywanym zbiorze w bieżącej iteracji pętli. Kolejny element `value-of` przekazuje na wyjście tytuł tego rozdziału. Należy zauważyć, że jest to potomek elementu `rozdział`, a nie elementu `książka`. Z uwagi na fakt, że na wyjście jest przekazywany zwykły tekst, należało wstawić drugi element `text` w celu przekazania znaku nowego wiersza. Wynik powyższej transformacji będzie miał postać podobną do poniższej:

```
Rozdział 1. Królewna Śnieżka
Rozdział 2. Brzydkie kacząтko
Rozdział 3. Kopciuszek
```

Kiedy dyrektywie `for-each` nie uda się dopasować żadnego wierzchołka procesor XSLT nie przechodzi do obszaru elementu a zamiast tego kontynuuje przetwarzanie szablonu. W przypadku instrukcji `for-each` nie istnieje ewentualność „w przeciwnym wypadku", ale można osiągnąć tego rodzaju funkcjonalność poprzez użycie instrukcji `if` oraz `choose`. Może być wymagane na przykład przekazanie na wyjście wartości atrybutu albo ciągu znaków bez żadnych znaczników. W takim przypadku należy użyć elementu `value-of`. Element ten wymaga użycia atrybutu `selekt`, który pobiera jako swoją wartość wyrażenie XPath. Element `value-of` przekazuje po prostu wartość tekstową takiego wyrażenia. W przykładzie A.2 użyto następującego szablonu :

```
<xsl:template match="część[@etykieta]">
      <dt>
       <xsl:value-of select="@etykieta"/>
     </dt>
     <dd>
       <xsl:apply-templates/>
     </dd>
 </xsl:template>
```

Służy on do pobrania wartości atrybutu `etykieta` i przekazanie jej w niezmienionej postaci jako treść `dt`. Ułatwieniem oferowanym przez język XSLT jest możliwość tworzenia obszarów służących o przechowywania tekstu, zwanych *zmiennymi* (ang. *variables*). W rzeczywistości są to stałe, których wartość określa się raz a odczytuje wielokrotnie. Zmienna musi być zdefiniowana przed użyciem. Można w tym celu użyć elementu `variable`. W przykładzie A.2 zostało to wykorzystane, gdzie zmiennej nadano wartość tekstową.

```
<xsl:template match="część[@ref]">
  <xsl:variable name="etykieta">
     <xsl:value-of select="@ref"/>
  </xsl:variable>
    <xsl:value-of select="//część[@etykieta=$etykieta]"/>
       <xsl:text>(Część </xsl:text>
        <xsl:value-of select="@ref"/>
       <xsl:text>) </xsl:text>
 <!-- np. w tekcie powstanie (Części A)-->
 </xsl:template>
```

Podobnie jak w przypadku parametrów, odwołanie do zmiennej musi zawierać jako prefiks znak dolara ($) a kiedy odwołanie do niej występuje poza wartościami atrybutów XPath, musi być ujęty w nawiasy klamrowe (). Zmienne mogą być używane w innych deklaracjach ale należy uważać, aby nie utworzyć zapętlonych definicji. Poniżej przedstawiono przykład niebezpiecznego zbioru przypisań zmiennych odwołujących się do siebie wzajemnie:

```
<xsl:variable name="rzecz1" select="rzecz2"/>
<xsl:variable name="rzecz2" select="rzecz1"/>
```

Sortowanie

Elementy muszą być często posortowane, aby były przydatne. Weźmy pod uwagę książkę telefoniczną, sortowanie według trzech kluczy: nazwiska, imienia oraz miasta. Dokument ma następującą postać:

```
<książka-telefoniczna>
...
<wpis lp="14">
   <nazwisko>Nowak</nazwisko>
   <imie>Karol</imie>
   <miasto>Warszawa</miasto>
   <telefon>555-5555</telefon>
</wpis>
<wpis lp="15">
   <nazwisko>Kowalski</nazwisko>
   <imie>Jan</imie>
   <miasto>Kielce</miasto>
   <telefon>555-1234</telefon>
</wpis>
...
</ książka-telefoniczna >
```

Domyślnie transformacja przetwarza każdy wierzchołek w kolejności, w jakiej występuje w dokumencie. Tak więc wpis z identyfikatorem lp=”14” jest przekazywany na wyjście przed wpisem z lp=”15”. Oczywiście nie byłaby to kolejność alfabetyczna, tak wiec trzeba posortować wyniki. Można tego dokonać używając elementu o nazwie `sort`. Na poniższym przykładzie widać jak można używać tej reguły:

```
<xsl:template match="książka-telefoniczna">
  <xsl:apply-templates>
     <xsl:sort select="miasto"/>
     <xsl:sort select="nazwiskko"/>
     <xsl:sort select="imie"/>
  </xsl:apply-templates>
</xsl:template>
```

Określono trzy osie sortowania. Najpierw otrzymane elementy wynikowe są sortowane według nazwy miasta. Następnie wpisy zostaną posortowane według nazwiska a wkońcu według imienia. W dodatku A został umieszczony przykład, który demonstruje użycie dotąd omówionych pojęć. Dodatkowo jest on dostępny na stronie internetowej pod adresem `http://theta.uwb.edu.pl/ mpod`. Po pierwsze przykład A.4 przedstawia przykładowy dokument XML, reprezentujący zestawienie dochodów i wydatków.

Napisany do przykładu A.4 arkusz XSLT dokona zamiany tego typu dokumentu na elegancko sformatowaną stronę HTML. Jako dodatkową korzyść osiągniemy to, że nasz arkusz stylów zsumuje dokonane transakcje i wydrukuje bilans zamknięcia (zakładając, że bilans otwarcia był 0).

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

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

<xsl:template match="zestawienie-doch-wyd">
  <html>
    <head/>
    <body>
      <h3>
        <xsl:text>Dochody za okres od </xsl:text>
        <xsl:value-of select="child::*[1]/data"/>
        <xsl:text> do </xsl:text>
        <xsl:value-of select="child::*[last()]/data"/>
        <xsl:text>:</xsl:text>
      </h3>
      <xsl:apply-templates select="lokata"/>
      <h3>
        <xsl:text>Wydatki za okres od </xsl:text>
        <xsl:value-of select="child::*[1]/data"/>
        <xsl:text> do </xsl:text>
        <xsl:value-of select="child::*[last()]/data"/>
        <xsl:text>, przedstawione od najwyższej do najniższej
                  kwoty transakcji:</xsl:text>
      </h3>
      <xsl:apply-templates select="opłata">
        <xsl:sort data-type="number" order="descending" select="kwota"/>
      </xsl:apply-templates>
      <h3>Bilans</h3>
      <p>
        <xsl:text>Stan konta na dzień </xsl:text>
        <xsl:value-of select="child::*[last()]/data"/>
        <xsl:text> to </xsl:text>
        <tt><b>
          <xsl:choose>
            <xsl:when test="sum( opłata/kwota ) > sum( lokata/kwota )">
              <font color="red">
                <xsl:value-of select="sum( lokata/kwota )
                                      - sum( opłata/kwota )"/>
                <xsl:text> zł</xsl:text>
              </font>
            </xsl:when>
            <xsl:otherwise>
              <font color="blue">
                <xsl:value-of select="sum( lokata/kwota )
                                      - sum( lokata/kwota )"/>
```

```
              <xsl:text> zł</xsl:text>
            </font>
          </xsl:otherwise>
        </xsl:choose>
      </b></tt>
    </p>
    <xsl:if test="sum( opłata/kwota ) > sum( lokata/kwota )">
      <p>
        <font color="red">
          <xsl:text>OSTRZEŻENIE! Szybko zasil konto!</xsl:text>
        </font>
      </p>
    </xsl:if>
  </body>
 </html>
</xsl:template>
```

Pierwszy szablon dotyczy najbardziej zewnętrznej struktury strony HTML. Zawiera również sekcję opisującą rozchody z konta. Lista transakcji zostanie posortowana od najwyższej do najniższej w tym celu użyjemy elementu `sort`. W końcu wyświetlimy stan konta. Użyjemy elementu `number` w celu obliczenia sumy transakcji. Wymagane są dwa składniki `sum( )`: jeden dla całkowitej kwoty wydatków i drugi dla całkowitej kwoty dochodów. Później odejmujemy całkowite wydatki od całkowitych dochodów. W celu pokazania, czy użytkownik jest na debecie, czy nie, obliczony wynik będzie formatowany różnym kolorem w zależności od otrzymanej wartości ora zostanie wyświetlone ostrzeżenie, jeżeli będzie on mniejszy od zera. Teraz potrzebne nam są pewne reguły służące do obsługi elementów `opłata` i `lokata`. Pierwsza z nich, przedstawiona poniżej, numeruje każdą opłatę i podsumowuje ją elegancko w postaci zdania:

```
<xsl:template match="opłata">
  <p>
    <xsl:value-of select="position()"/>
    <xsl:text>. W dniu </xsl:text>
    <xsl:value-of select="data"/>
    <xsl:text> użytkownik zapłacił kwotę </xsl:text>
    <tt><b>
      <xsl:value-of select="kwota"/>
      <xsl:text> zł</xsl:text>
    </b></tt>
    <xsl:text> dla </xsl:text>
    <i>
      <xsl:value-of select="beneficjent"/>
    </i>
    <xsl:text> za </xsl:text>
    <xsl:value-of select="opis"/>
```

```
    <xsl:text>.</xsl:text>
  </p>
</xsl:template>
```

Sprawdza się to dobrze w przypadku większości typów opłat, jednak nie sprawdza się, kiedy typem jest `typ="bankomat"`. Należy zauważyć, w realizacji dokumentu opłata `bankomat` nie zawiera żadnego opisu dotyczącego beneficjenta, gdyż zakładamy, że to autor zestawienia podejmuje środki pieniężne. Poniżej utworzony specjalną regułę tylko dla tego przypadku:

```
<xsl:template match="opłata[@typ='bankomat']">
  <p>
    <xsl:value-of select="position()"/>
    <xsl:text>. W dniu </xsl:text>
    <xsl:value-of select="data"/>
    <xsl:text> użytkownik wypłacił kwotę </xsl:text>
    <tt><b>
      <xsl:value-of select="kwota"/>
      <xsl:text> zł</xsl:text>
    </b></tt>
    <xsl:text> z bankomatu z przeznaczeniem na </xsl:text>
    <xsl:value-of select="opis"/>
    <xsl:text>.</xsl:text>
  </p>
</xsl:template>
```

W końcu określamy regułę dla elementu `lokata`:

```
<xsl:template match="lokata">
  <p>
    <xsl:value-of select="position()"/>
    <xsl:text>. W dniu </xsl:text>
    <xsl:value-of select="data"/>
    <xsl:text> kwota </xsl:text>
    <tt><b>
      <xsl:value-of select="kwota"/>
      <xsl:text> zł</xsl:text>
    </b></tt>
    <xsl:text> została wpłacona na konto użytkownika przez </xsl:text>
    <i>
      <xsl:value-of select="płatnik"/>
    </i>
    <xsl:text>.</xsl:text>
  </p>
</xsl:template>
</xsl:stylesheet>
```

Poniżej przedstawiono wynikowy plik HTML

2.1.3 XSL Formatting Objects

Język XSL-FO uzupełnia trójkę standardów, które składają się na język XSL. Akronim FO oznacza *Formatting Objects*,które są kontenerami treści zachowującymi strukturę i wiążącymi ze sobą informacje prezentacyjne. XSLT przygotowuje dokument do formatowania za pomocą języka XPath, rozkładające go na poszczególne części i tworząc tymczasowy plik XSL-FO, który jest wykorzystywany przez formater.
Standard CSS (ang. *Cascading Style Sheets*- kaskadowe arkusze stylów) miał duży wpływ na opracowywanie XSL. Jest prosty ale wiele dostępnych instrukcji właściwości sprawia, że jest łatwy do opanowania. Wiele z tych właściwości przeniesiono do XSL-FO. Elegancko i wydajny model ramkowy CSS stanowi podstawę modelu obszarów XSL-FO. To, co dodano do XSL-FO to mechanizmy obsługi rozkładu graficznego stron oraz złożonych systemów pisma. Podstawowe zalety XSL-FO w porównaniu z CSS to:

- Mechanizmy związane z zagadnieniami drukarskimi, na przykład łamaniem stron. CSS także zamierza w tym kierunku, tak więc ta różnica nie jest zbyt istotna.
- Przetwarzanie bez określonej kolejności. W przypadku CSS wszystkie elementy w dokumencie są przetwarzane w kolejności od początku do końca. Nie zapewnia to żadnego sposobu wyboru pewnych sekcji i odrzucenia innych lub pobierania informacji z różnych części dokumentu, czy też przetworzenie tego elementu więcej niż raz.
- Składnia XML. CSS wykorzystuje własną składnię, co utrudnia przetwarzanie.

XSL nie zawsze jest lepszym rozwiązaniem od CSS. Ten ostatni standard wybrano by w sytuacji, gdy istotną sprawą jest zachowanie prostoty i szybkości formatowania. W przeciwieństwie do języka CSS, który po prostu bezpośrednio stosuje style względem elementów w pojedynczym przebiegu, XSL daje możliwość dokonania poważnych zmian w strukturze dokumentu. Dzieje się to kosztem prostoty. W celu ułatwienia działań programistów projektanci języka XSL rozdzielili cały proces na dwie części: transformację i formatowanie. Transformacja zmienia strukturę dokumentu wejściowego i dodaje informacje prezentacyjne w hybrydowym formacie składającym się z obiektów formatujących. Obiekt formatujący jest kontenerem treści, który wiąże style i instrukcje renderujące z treścią. Jest zwarty i łatwy do zrozumienia przez użytkownika. Obiekty formatujące tworzą strukturę drzewiastą, która zachowuje strukturę używaną w czasie tworzenia końcowej prezentacji. Wynikiem transformacji jest tymczasowy plik, który się przekazuje się do formatu XSL-FO. W serii skomplikowanych etapów formater oblicza końcową postać i wygląd prezentacji, a następnie tworzy plik nadający się wydruku lub wyświetlenia na ekranie. Po zakończeniu działania obiekty formatujące są usuwane z pamięci i można

usunąć tymczasowy plik FO. Istotną rzeczą jest zrozumienie, że nie oczekuje się od użytkownika tworzenia własnych znaczników XSL-FO.
Formater jest odpowiedzialny za wykonanie złożonych operacji. W pierwszej fazie formatowania formater tłumaczy dokument do postaci reprezentacji obiektowej przechowywanej w pamięci, w procesie noszącym nazwę *obiektyfikacji*. Taka struktura, *drzewo obiektów formatujących*, jest podobna do drzewa wynikowego ale charakteryzuje się zmianami pewnych szczegółów. Po dokonaniu obiektyfikacji następuje faza druga, *uściślanie*. Formater rozpoczyna obliczanie faktycznych rozmiarów i położenia obszarów w prezentacji, zastępując wartości względne i ograniczania konkretnymi liczbami. Przykładowo, szerokość komórki tabeli, którą określono jako wartość procentową, zostaje określona w pikselach. Wartość procentowa szerokości względnej w tabeli zostanie zastąpiona obliczoną liczbą pikseli. Wynikiem działania tej fazy jest *uściślone drzewo obiektów formatujących*. Następnie przechodzimy do fazy *formatowania*, w której właściwości obiektów formatujących zostają przetłumaczone na wirtualny obraz dokumentu zwany *drzewem obszarów*. Ta przechowywana w pamięci reprezentacja dokumentu wyjściowego składa się z nakładających się na siebie i zagnieżdżonych kanw, zwanych *obszarami*. Każdy obszar to region dokumentu z zestawem *wyróżników*, które opisują jak powinna być renderowana ich treść. Obszar może być tak duży jak ramka lub tak mały jak pojedynczy glif. W końcu formater przechodzi do fazy *renderowania*. Drzewo obszarów zostaje bezpośrednio przetłumaczone na pewien format wyjściowy, taki jak PDF lub troff, który dokładnie odpowiada oryginalnej specyfikacji. Wreszcie można zobaczyć efekt działań, który powinien wyglądać tak jak to zakładano na początku. Aby bliżej przyjrzeć się całemu procesowi, poniżej przyjrzymy się krótkiemu przykładowi.

```
<dokument>
      <tytuł>Hello world !!! </tytuł>
       <wiadomość>
           Jestem <wyróżnienie>taka</wyróżnienie> mądra
      </wiadomość>
</dokument>
```

Pierwszym etapem użycia XSL-FO jest napisanie arkusza stylów XSLT, który wygeneruje drzewo obiektów formatujących. W przekładzie 2.2 przedstawiono przykład arkusza stylów. Zawiera on w sumie pięć szablonów. Pierwszy z nich tworzy stronę główną- archetyp rzeczywistych stron, które będą tworzone w miarę wstawiania tekstu – określają formę regionów treści. Drugi szablon kojarzy obiekt przepływu ze stroną główną. Przepływ to niejako podajnik bagaży, który umieszcza walizki w zwartej przestrzeni, pasującej do wymiarów określonych w stronie głównej. Reszta szablonu służy do utworzenia bloków oraz elementów wplatanych, które zostaną umieszczone w ramach przepływu.

[2.2]
Przykład 2.2.
Arkusz XSLT służący do zamiany dokumentu mójdok na drzewo obiektów formatujących

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

Element korzenia, gdzie ustawiana jest strona z pojedynczym regionem. Element <layout-master-set> może zawierać wiele stron głównych ale w tym przypadku zdefiniowano tylko jedną. Element <simple-page-master> określa podstawowy typ strony, której wysokość, szerokość, marginesy i nazwa służą do późniejszego odwoływania się w przepływie.

```
<xsl:template match="/">
  <fo:root>
    <fo:layout-master-set>
      <fo:simple-page-master
            master-name="the-only-page-type"
            page-height="4in" page-width="4in"
            margin-top="0.5in" margin-bottom="0.5in"
            margin-left="0.5in" margin-right="0.5in">
        <fo:region-body/>
      </fo:simple-page-master>
    </fo:layout-master-set>
    <xsl:apply-templates/>
  </fo:root>
</xsl:template>
```

Pierwszy element blokowy, gdzie wstawiamy przepływ dokumentu. Element <page-sequence> określa realizację typu strony, który zdefiniowano powyżej. Element <flow> zawiera wszystkie obiekty blokowe, ustawiając je tak aby mieściły się w zdefiniowanym regionie strony. Przepływ zawiera blok, który definiuje nazwę czcionki, rozmiar, wyrównanie tekstu oraz otaczającą treść w buforze dopełnienia o szerokości 0,25 cala.

```
<xsl:template match="mójdok">
  <fo:page-sequence master-reference="the-only-page-type">
    <fo:flow flow-name="xsl-region-body">
      <fo:block
        font-family="helvetica, sans-serif"
        font-size="24pt"
        text-align="center"
        padding="0.25in"
      >
        <xsl:apply-templates/>
      </fo:block>
    </fo:flow>
```

```
  </fo:page-sequence>
</xsl:template>
```

Drugi element blokowy, tytuł, jest pogrubiony, pisany czcionką o rozmiarze 10 punktów oraz wstawia pod spodem wolny obszar o rozmiarze 1 em.

```
<xsl:template match="tytuł">
  <fo:block
    font-weight="bold"
    font-size="10pt"
    space-after="1em"
  >
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

Ostatni element blokowy, treści wiadomości. Dopełnienie ustawiono na 0,25 cala a krawędź jest widoczna.

```
<xsl:template match="wiadomość">
  <fo:block
    padding="0.25in"
    border="solid 1pt black"
  >
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
```

Wplątany element wyróżnienia ustawiany jako pisany kursywą.

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

</xsl:stylesheet>
```

Po wykonaniu transformacji dokumentu źródłowego zostanie wygenerowany następujące drzewo obiektów formatujących:

```
<?xml version="1.0"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="the-only-page-type"
```

```
          page-height="3in" page-width="4in" margin-top="0.5in"
          margin-bottom="0.5in" margin-left="0.5in"
          margin-right="0.5in"
          >
      <fo:region-body />
     </fo:simple-page-master>
  </fo:layout-master-set>
  <fo:page-sequence master-reference="the-only-page-type">
  <fo:static-content flow-name="xsl-region-before">
  <fo:flow flow-name="xsl-region-body">
    <fo:block font-family="helvetica, sans-serif" font-size="24pt"
              text-align="center" padding="0.25in"
              >
     <fo:block font-weight="bold" font-size="10pt" space-after="1em">
        Hello world!!!
     </fo:block>
       <fo:block padding="0.25in" border="solid 1pt black">Jestem
         <fo:inline font-style="italic"> taki </fo:inline> bystry.
      </fo:block>
    </fo:block>
  </fo:flow>
  </fo:page-sequence>
</fo:root>
```

W przypadku języka XSL wszelkie działania związane z pozycjonowaniem, kształtowaniem i rozdzielaniem elementów na stronie są wykonywane w ramach *obszarów*. Obszar to atrakcyjna struktura, używana do reprezentowania fragmentów sformatowanego dokumentu. Zawiera wszystkie informacje geometryczne i stylistyczne, potrzebne do umiejscowienia i renderowania należących do niego elementów. Obszary są zagnieżdżone, co prowadzi do możliwości użycia pojęcia drzewa obszarów. Drzewo obszarów całkowicie opisuje sformatowany dokument od wierzchołka korzenia do obszarów zawierających glify i obrazy. Każdy obszar odwzorowuje *prostokąt treści* – referencyjną ramkę, w stosunku do której formater pozycjonuje elementy podrzędne i rozmieszcza odstępy. Dzieli on zdefiniowaną przestrzeń na dwa fragmenty: *margines* na zewnątrz oraz *dopełnienie* wewnątrz. Widocznym zobrazowaniem prostokąta treści jest krawędź. Od zewnątrz ogranicza ją *prostokąt krawędzi* a od wewnątrz *prostokąt dopełnienia*.
Obiekty formatujące to podstawowy materiał budulcowy transformacji drzewa wynikowego, który napędza działanie formatera. Są one zwartymi kontenerami treści i stylu wraz z wszystkimi informacjami potrzebnymi do wygenerowania sformatowanego dokumentu gotowego do prezentacji. Istnieją dwa główne rodzaje obiektów formatujących. *Obiekty przepływu* tworzą obszary i występują wewnątrz przepływów. *Obiekty układu*, inaczej zwane *obiektami pomocniczymi*, pomagają w tworzeniu obszarów, przekazując pewne parametry. Obiekt `block` tworzy region treści, która ma zostać wstawiona do przepływu,

tak więc jest on traktowany jako obiekt przepływu. Z kolei obiekt formatujący `initial-page-number` powoduje wyzerowanie licznika numerów stron. Ze względu na fakt, że udostępnia on jedynie pewne informacje pomagające w procesie przetwarzania i nie tworzy własnych regionów, jest to obiekt układu. Struktura dokumentu obiektów formatujących ma postać drzewa, podobnie jak ma to miejsce w przypadku wszystkich innych dokumentów XML każdy jego element jest obiektem formatującym, tak więc określa się je mianem drzewa FO. Korzeniem tego drzewa jest element `root`. Do jego potomków należą:

layout-master-set jest zawarta specyfikacja podziłu na strony oraz układu. Istnieją dwa jego rodzaje. Obiekt `page-master` definiuje właściwości typu strony: jej cechy geometryczne oraz sposób podziału. Obiekt `page-sequence-master` kontroluje sposób występowania typu stron w sekwencji.

declarations opcjonalny element, który zawiera ustawienia globalne, mające wpływ na ogólne formatowanie.

page-sequence jeden lub kilka tych elementów zawiera obiekty przepływu, które przechowują treść dokumentu.

Strona główna definiuje prostokąt treści dla strony oraz sposób jego podziału na regiony. Jednym z obiektów stron głównych w specyfikacji XSL w wersji 1.0 jest `simple-page-master`. Jego atrybut `master-name` stanowi uchwyt dla sekwencji stron głównych. Inne atrybuty służą do definiowania prostokąta treści, strony, na przykład:

```
<fo:simple-page-master
            master-name="strona-tytułowa"
            page-height="11in" page-width="8.5in"
            margin-top="1in" margin-bottom="1in"
            margin-left="1.2in" margin-right="1.2in">
  ...
</fo:simple-page-master>
```

Powyższy obiekt formatujący deklaruje typ strony głównej, której nazwą jest `strona-tytułowa`. Jej wysokość wynosi 11 cali a szerokość 8,5 cala. Prostokąt treści znajduje się wewnątrz tego obszaru, jeden cal od góry i od dołu oraz 1,2 cala od prawej i lewej. Po zdefiniowaniu prostokąta treści kolejnym zadaniem strony głównej jest określenie regionów dla treści na stronie. Obiekt `simple-page-master` pozwala podzielić stronę na maksymalnie pięć różnych obszarów, noszących następujące nazwy: *region główny* (ang. *body region*), *region poprzedzający* (ang .*before region*), *region następujący* (ang. *after region*), *region początkowy* (ang. *start region*) oraz *region końcowy* (ang. *end region*). Blokowe obiekty formatujące są obiektami, które wypełniają przepływ. Każdy blok posiada co najmniej jeden obszar pod swoją kontrolą. Przykładowo, paragraf w dokumencie zostaje odwzorowany na obiekt blokowy w drzewie

obiektów formatujących. Zostaje on umieszczony w ramach przepływu jako pojedynczy obszar z marginesami, dopełnieniem, czcionką i innymi właściwościami. Niektóre właściwości wspólne dla wszystkich bloków można podzielić na następujące kategorie: tło, krawędź , czcionki, marginesy oraz dopełnienia.

Wiele typów bloków, takich jak parametry I tytuły, zawiera tekst. Strumienie tekstu przebiegają od jednej krawędzi prostokąta treści do drugiej, gdzie przechodzą do kolejnego wiersza do momentu osiągnięcia dolnej krawędzi bloku. Wśród tych wierszy znajdują się elementy, które przesłaniają domyślne ustawienia czcionek lub zostają zastąpione ogólną treścią. Są to elementy wplątane. Obiekt formatujący typu `inline` to ogólny kontener tekstu, który pozwala przesłaniać aktualne właściwości tekstu. Wykorzystuje wszystkie właściwości czcionki i koloru związane z blokami oraz pewne właściwości odstępów. W poniższym przykładzie szablon XSLT odwzorowuje element `wyróżnienie` na element `inline`, który określa jako właściwości treści kolor zielony oraz kursywę.

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

Rozdział 3

Dokumentacja techniczna w XML

w poniższym rozdziale zostanie zaprezentowana lista znaczników, które będą opisywały dokumentację techniczną. Ideą jest, aby każdy napisany dokument zgodny z listą znaczników można było trasformować na format HTML oraz PDF. W tym celu zostanie wykorzystany artykuł pochodzący ze strony http://solaris-x86.org/documents/tutorials/network.mhtml. Dokument został opisany w sposób logiczny przy użyciu XML. Całe kody plików nie zostały zamieszczone w pracy ze względu na ich obszerność.
Cały dokument spina znacznik <Article>, w którym można wyróżnić dwie części główne: część nagłówkową oraz dokument właściwy. W części nagłówkowej mogą znaleźć się takie znaczniki jak <Title>, w którym zawarty jest tytuł dokumentu. W znaczniku <Author> umieszczamy dane autora na przykład w znaczniku <First_name> zawiera się imię w znaczniku <Last_name> nazwisko oraz w znaczniku <Homepage> adres strony domowej autora. Dodatkowo można umieścić <Create_date>, w którym znajduje się data powstania dokumentu oraz w znaczniku <Modify_Date> można zawrzeć datę ostatniej modyfikacji. Użycie powyższych znaczników zaprezentowane są w dodatku B przykładzie B.1.
Część właściwa dokumentu zawarta jest w znaczniku <BODY>. Znacznik <Steps> może być wykorzystany jeżeli opisywana czynność składa się z kilku kroków, gdzie każdy krok będzie w oddzielnym znaczniku <Step>. Znacznik <Step> może dodatkowo zawierać atrubut names, który może na przykład przechowywać nazwę rozdziału. Przykładowe użycie w dodatku B przykład B.2.
Opisanie ścieżek należy zawierać w znaczniku <Path>. Znacznik ten może mieć dodatkowo atrybut type przyjmujący wartości file lub directory w zależności czy dana ścieżka jest do pliku czy folderu. W znaczniku <Path> może zawierać sie znacznik <part>, który może przyjąć atrybut name. Przykład użycia zamieszczono w dodatku B przykład B.3.
Do opisywania komend służy znacznik <Command>. Zawiera on znacznik <Program> oraz <Argument>. Pierwszy ze znaczników przechowuje nazwę pro-

gramu. Natomiast w drugim znajdują się argumenty jakie należy podać. Ponieważ znacznik <Argument> może posiadać atrybut name jesteśmy w stanie rozróżnić czy dany argument jest ścieżką do pliku lub folderu czy też inną wartością. Przykład umieszczono w dodatku B przykład B.4.
Przydatnym znacznikiem może okazać sie <Config_file>. Dodatkowo znacznik ten może zawierać atrybut path, w którym przechowywana będzie ścieżka do danedo pliku. W znaczniku <Config_file> mogą zawietać się inne znaczniki na przykład <line>, który reprezentuje konkretną linię z dokumentu. Linia może składać się z konkretnych pól, które powinny byc wyróżnione możemy w tym celu skorzystać ze znacznika <field>. Znacznik <field> może dodatkowo zawierać atrybut name. Stosowny przykład zaprezentowany jest w dodatku B przykład B.5.
Przydatnymi znacznikami mogą być <notice>(przykład B.6), który służy do przechowywania uwag, <Error> (przykład B.7), wyróżnienie błędu, <emph> (przykład B.2), służy do podkreślenia zawartej w niej treści. Znacznik <Netmask> (przykład B.9) zawiera netmaskę, znacznik <IP_adress> (przykład B.9) przechowuje IP adres oraz znacznik <link> (przykład B.8) zawiera link do konkretnej strony na przykład **WWW**. Często wykorzystywanymi elementemi są listy na przykład aby wymienić składniki korzysta się ze znacznika <list>, każdą wypunktowaną rzecz należy zawierać w znaczniki <item>. Najlepiej zobrazuje to przykład zawarty w dodatku B przykład B.10.
Często dane są przedstawiane w postaci tabeli. Najbardziej zewnętrzym znacznikiem jest <Table>, który zawiera całą zawartość dokumentu. Można wyróżnić w tabeli trzy główne części: część w której deklarujemy ilość i stosunek szerokość kolumn, nagłówek tabeli oraz część właściwą tabeli. Znacznikiem opisujący pierwszą część jest <ColSpec>, który w atrybucie width zawiera liczby używane do obliczenia na podstawie stosunku szerokości kolumny. Część nagłówkową tabeli zawieramy w znaczniku <TableHeader> natomiast część właściwą w <TableBody>. Każdy z powyższych znaczników może składać się z <Row>, który reprezentuje pojedyńczy wiersz w tabeli oraz <Col> reprezentujący już konkretną komórkę. Dodatkowo znacznik <Col> może zawierać atrybut name, który może zawierać nazwę kolumn tabeli. Przykład przedstawiony został w dodatku B przykład B.11.

Dodatek A

XSL Transormations - przykłady

[A]
XSLT przyklady

[A.1]
Instrukcja użytkownika

Przykład A.1.

```
<?xml version="1.0" encoding="iso-8859-2"?>
<?xml-stylesheet type="text/xml" href="projekt.xsl""?>

<podręcznik typ="montaż" id="model-rakiety">
<lista-części>
      <część etykieta="A" liczba="1"> kadłub, lewa połowa </część>
      <część etykieta="B" liczba="1"> kadłub, prawa połowa </część>
      <część etykieta="F" liczba="4"> brzechwa sterująca </część>
      <część etykieta="N" liczba="3"> dysza rakiety </część>
      <część etykieta="C" liczba="1"> kapsuła załogowa </część>
  </lista-części>

  <instrukcje>
    <etap>
        Sklej części <część ref="A"/> i <część ref="B"/> w celu
        utworzenia  kadłuba.
    </etap>

    <etap>
        Naklej na <część ref="F"/> i wstaw je do otworów
        w kadłubie.
    </etap>

    <etap>
        Przy pomocy niewielkiej ilości kleju przymocuj
        <część ref="N"/> w dolnej części  kadłuba.
    </etap>
```

```
    <etap>
        Przymocuj <część ref="C"/> do górnej części kadłuba.
        Nie używaj kleju, gdyż jest ona zamocowana na spęrzynie,
        umożliwiającej odczepienie się od kadłuba
    </etap>
  </instrukcje>
</podręcznik>
```

[A.2] arkusz XSLT dla instrukcji użytkownika

Przykład A.2.

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

 <!--  Obsługa elementu dokumentu: ustawienia dokumentu HTML-->

 <xsl:template match="podręcznik">
  <html>
     <head><title>Instrukcja użytkownika</title></head>
     <body >
       <h1>Instrukcja użytkownika</h1>
        <xsl:apply-templates/>
     </body>
  </html>
 </xsl:template>

 <!--  Utworzenie listy części-->

 <xsl:template match="lista-części">
  <h2>Część</h2>
     <dl>
       <xsl:apply-templates/>
     </dl>
 </xsl:template>

 <!--  jedno z zastosowań elementu <część> pozycja na liście-->

 <xsl:template match="część[@etykieta]">
      <dt>
       <xsl:value-of select="@etykieta"/>
     </dt>
     <dd>
       <xsl:apply-templates/>
```

```
    </dd>
 </xsl:template>

 <!--kolejne zastosowanie elementu <część>: wygenerowanie nazwy części-->

 <xsl:template match="część[@ref]">
  <xsl:variable name="etykieta">
     <xsl:value-of select="@ref"/>
  </xsl:variable>
    <xsl:value-of select="//część[@etykieta=$etykieta]"/>
       <xsl:text>(Część </xsl:text>
        <xsl:value-of select="@ref"/>
       <xsl:text>) </xsl:text>
 <!-- np. w tekcie powstanie (Części A)-->
 </xsl:template>

 <!--  Określenie listy instrukcji-->

 <xsl:template match="instrukcje">
  <h2>Etapy</h2>
     <ol>
       <xsl:apply-templates/>
     </ol>
 </xsl:template>

 <!--  Obecnie każdej pozycji (<etap>) z listy instrukcji-->

 <xsl:template match="etap">
   <li>
     <xsl:apply-templates/>
   </li>
 </xsl:template>

</xsl:stylesheet>
```

[A.3] kod w HTML

Przykład A.3.

```
<html>
      <head><title>Instrukcja użytkownika</title></head>
 <body>
        <h1>Instrukcja użytkownika</h1>
         <h2>Części</h2>
     <dl>
         <dt>A</dt>
          <dd>kadłub, lewa połowa</dd>
          <dt>B</dt>
```

```
          <dd>kadłub, prawa połowa</dd>
           <dt>F</dt>
           <dd>brzechwa sterująca</dd>
           <dt>N</dt>
           <dd>dysza rakiety</dd>
           <dt>C</dt>
          <dd>kapsuła załogowa</dd>
      </dl>
       <h2>Etapy</h2>
<ol>
  <li>
     Sklej części kadłub, lewa połowa  (Część A) i kadłub, prawa
     połowa (Część B) w celu utworzenia  kadłuba.
  </li>
  <li>
     Naklej na brzechwa sterująca (Część F) i wstaw je do otworów
     w kadłubie.
  </li>
  <li>
     Przy pomocy niewielkiej ilości kleju przymocuj dysza rakiety
     (Część N) w dolnej części kadłuba..
  </li>
  <li>
     Przymocuj kapsuła załogowa (Część C) do górnej części kadłuba.
     Nie używaj kleju, gdyż jest ona zamocowana na spęrzynie,
     umożliwiającej odczepienie się od kadłuba.
  </li>
</ol>
</body>
</html>
```

[A.4] zestawienie dochodów i wydatków

Przykład A.4.

```
<zestawienie-doch-wyd>

  <lokata typ="lokata-prosta">
    <płatnik>Zakłady Metalowe Boltex</płatnik>
    <kwota>987.32</kwota>
    <data>21-6-00</data>
    <opis kategoria="dochód">Spłata czeku</opis>
  </lokata>

  <opłata typ="czek" numer="980">
    <beneficjent>Sklep sportowy Kimora</beneficjent>
    <kwota>132.77</kwota>
    <data>23-6-00</data>
```

```
    <opis kategoria="rozrywka">wyposażenie do kendo</opis>
  </opłata>

  <opłata typ="bankomat">
    <kwota>40.00</kwota>
    <data>24-6-00</data>
    <opis kategoria="gotówka">kieszonkowe</opis>
  </opłata>

  <opłata typ="debet">
    <beneficjent>Kawiarnia Samotna Gwiazda</beneficjent>
    <kwota>36.86</kwota>
    <data>26-6-00</data>
    <opis kategoria="jedzenie">lunch z Mirkiem</opis>
  </opłata>

  <opłata typ="czek" numer="981">
    <beneficjent>Market Unreal</beneficjent>
    <kwota>47.28</kwota>
    <data>29-6-00</data>
    <opis kategoria="jedzenie">artykuły spożywcze</opis>
  </opłata>

  <opłata typ="debet">
    <beneficjent>Merlin</beneficjent>
    <kwota>58.79</kwota>
    <data>30-6-00</data>
    <opis kategoria="praca">książki z Heliona</opis>
  </opłata>

  <opłata typ="czek" numer="982">
    <beneficjent>Stare i starsze</beneficjent>
    <kwota>800.00</kwota>
    <data>31-6-00</data>
    <opis kategoria="różne">3-nożny kredens antyk, który kiedyś
          należał do Alfreda Hitchcocka</opis>
  </opłata>

</zestawienie-doch-wyd>

<html>
<body>
  <h3>Przychód z okresu od 21-6-00 do 31-6-00:</h3>
  <p>
     1. W dniu 21.6.00 kwota <tt><b>$987.32</b></tt>
        została wpłacona na twoje konto przez <i>
        Zakłady Metalowe Boltex</i>.
```

```
  </p>
 <h3>Wydatki z okresu od 21-6-00 do 31-6-00, przedstawione
     od najwyższej do najniższej kwoty transakcji:
 </h3>
  <p>
     1. W dniu 31-6-00 użytkownik zapłacił kwotę
        <tt><b>$800.00</b></tt> dla <i>Stare i starsze</i>
         za 3-nożny kredens antyk, który kiedyś należał
         do Alfreda Hitchcocka.
  </p>
  <p>
     2. W dniu 23-6-00 użytkownik zapłacił kwotę
        <tt><b>$132.77</b></tt> dla <i>Sklep sportowy Kimora1</i>
        za wyposażenie do kendo.
 </p>
 <p>
     3. W dniu 30-6-00 użytkownik zapłacił kwotę
        <tt><b>$58.79</b></tt> dla <i>Merlin</i>
        za książki z Heliona.
 </p>
 <p>
     4. W dniu 29-6-00 użytkownik zapłacił kwotę
        <tt><b>$47.28</b></tt> dla <i>Market Unreal</i>
        za artykuły spożywcze.
 </p>
 <p>
     5. W dniu 24-6-00 użytkownik wypłacił kwotę
        <tt><b>$40.00</b></tt> z bankomatu z przeznaczeniem
        na kieszonkowe.
 </p>
 <p>
     6. W dniu 26-6-00 użytkownik zapłacił kwotę
        <tt><b>$36.86</b></tt> dla <i>Kawiarnia Samotna Gwiazda</i>
        za lunch z Mirkiem.
 </p>
<h3>Bilans</h3>
  <p>Stan twojego konta na dzień 31-6-00 to
     <tt><b><font color="red">$-128.38
     </font></b></tt>
  </p>
  <p>
   <font color="red">OSTRZEŻENIE! Szybko zasil konto!</font>
  </p>
</body>
</html>
```

Dodatek B

Przykłady

[B]
przyklady

[B.1]
Zewnętrzna struktura strony

Przykład B.1.

```
<Article>
  <Title>Configuring networking</Title>
<Author>
  <First_name>Keith</First_name><Last_name> Parkansky</Last_name>
  <Homepage>http://www.execpc.com/~keithp</Homepage>
</Author>

<Create_date> May 1, 2002</Create_date>

<Modify_Date> May 1, 2002</Modify_Date>

<BODY>

  ....

</BODY>

</Article>
```

[B.2]
Steps

Przykład B.2.

```
<Steps>
  <Step names="aaa">
          Pop up the menu above the "Text Note" icon and click on
          <emph> Text Editor </emph>.
          Use the editor to open the file:
          /boot/solaris/devicedb/master
          and use the Find feature (under Edit on the menu bar)
          to locate the model number of your NIC - example:
          3C905 (3Com) or 9432 (SMC). Note that the Find feature
```

```
            <emph> is case sensitive </emph> in the CDE text editor.
   </Stage>

   <Stage>
   ...
   </Stage>
</abc>
```

[B.3] Ścieżki

Przykład B.3.

```
<Path type="file">
   <part> /etc/hostname. </part>
   <part name="DRIVER"> driver_name </part>
   <part name="NIC">NIC_number</part>
</Path>
```

[B.4] Komenda

Przykład B.4.

```
<Command>
  <Program>
      mv
  </Program>
      <Argument name="path">
           /etc/rc2.d/S88sendmail
      </Argument>
      <Argument name="path">
          /etc/rc2.d/noS88sendmail
     </Argument>
</Command>
```

[B.5] config-file

Przykład B.5.

```
<Config_file path="/etc/hosts">
 <line>
   <field name="IP">         192.168.10.20          </field>
   <field name="hostname">   solarisi               </field>
   <field>                   solarisi.bigsunfan.com </field>
   <field>                   loghost                </field>
 </line>
</Config_file>
```

[B.6] notice

Przykład B.6.

```
<notice>
    Make sure you press Enter after typing
    this in to create a new (blank) line
    beneath it!
</notice>
```

[B.7] error

Przykład B.7.

```
<Error>
   sendmail[nnn]: My unqualified host name (solarisi) unknown;
   sleeping for retry
</Error >
```

[B.8] link

Przykład B.8.

```
<link>
   Configuring Sendmail
</link>
```

[B.9] ip-adress

Przykład B.9.

```
<IP_adress>
     192.168.10.0
</IP_adress>

<Netmask >
   255.255.255.0
</Netmask>
```

[B.10] lista

Przykład B.10.

```
<list>

    <item> hosts, </item>
    <item> resolv.conf, </item>
    <item> nsswitch.conf, </item>
    <item> netmasks, </item>
    <item> defaultrouter, </item>
    <item> ifconfig </item>
    <item> route </item>
    <item> ping. </item>

</list>
```

[B.11] tabela

Przykład B.11.

```
<Table>

   <ColSpec width="1"/>
   <ColSpec width="3"/>
   <ColSpec width="2"/>
   <ColSpec width="3"/>

  <TableHeader>
      <Row>
         <Col name="IP_address">IP Number     Class </Col>
         <Col name="Network_address">Network Address Range</Col>
```

```
     <Col name="Subnet_mask">Subnet Mask</Col>
         <Col name="Private_address">Private Address Range</Col>
      </Row>
  </TableHeader>
  <TableBody>
      <Row>
        <Col name="IP_address">  A </Col>
        <Col name="Network_address">1.0.0.0 to 126.0.0.0</Col>
        <Col name="Subnet_mask">255.0.0.0</Col>
        <Col name="Private_address">10.x.x.x </Col>
      </Row>

      <Row>
        <Col name="IP_address">  B </Col>
        <Col name="Network_address">128.0.0.0 to 191.255.0.0 </Col>
        <Col name="Subnet_mask"> 255.255.0.0 </Col>
        <Col name="Private_address">172.16.x.x to 172.31.x.x  </Col>
      </Row>

      <Row>
        <Col name="IP_address">  C </Col>
        <Col name="Network_address"> 192.0.0.0 to 223.255.255.0 </Col>
        <Col name="Subnet_mask">255.255.255.0</Col>
        <Col name="Private_address">192.168.0.x to 192.168.255.x </Col>
     </Row>
  </TableBody>
</Table>
```

Bibliografia

szmielew [1] Szmielew W., *Od geometrii afinicznej do euklidesowej*, Państwowe Wydawnictwo Naukowe, 1981.