Interfejs użytkownika I

Wykład prowadzi:
Marek Wojciechowski

Plan wykładu

- Formatowanie HTML za pomocą arkuszy stylów CSS
- Język XML
  - ogólna struktura dokumentów
  - opis struktury za pomocą DTD
  - przestrzenie nazw
- Język XHTML
- Formatowanie i transformacja dokumentów XML za pomocą arkuszy stylów XSL
  - transformacje XSLT
  - obiekty formatujące XSL-FO

Celem tego i kolejnego wykładu jest przedstawienie podstawowych technologii, które oprócz języka HTML służą do tworzenia interfejsu użytkownika w aplikacjach internetowych.

W ramach niniejszego wykładu w pierwszej kolejności omówione będą arkusze stylów CSS i ich wykorzystanie do formatowania dokumentów HTML. Następnie, w podstawowym zakresie, przedstawiony będzie język XML z uwzględnieniem ogólnej struktury dokumentu, opisu struktury za pomocą DTD i przestrzeni nazw. Kolejne zagadnienie to język XHTML, będący wersją HTML dostosowaną do reguł języka XML. Na zakończenie wykładu przedstawiony będzie język XSL służący do transformowania i formatowania dokumentów XML. Omówione zostaną zarówno część XSL dotycząca transformacji dokumentów (XSLT) jak i część dotycząca opisu sposobu formatowania (XSL-FO).

Cascading Style Sheets (CSS)

- Preferowany mechanizm do definiowania stylu prezentacji dokumentów w Internecie
- Mechanizm oparty o arkusze stylów w postaci listy reguł
- Ułatwia spójne formatowanie zbioru dokumentów
- Umożliwia dokładniejszą kontrolę sposobu wyświetlania elementów niż znaczniki HTML
- Mimo powszechnego wsparcia, ciągle drobne różnice między przeglądarkami

Kaskadowe arkusze stylów (ang. Cascading Style Sheets, CSS) to mechanizm (a jednocześnie język) służący do definiowania stylu prezentacji dokumentów w Internecie. CSS może być wykorzystany do formatowania dokumentów HTML, XML i XHTML. Od wersji 4.0 specyfikacji języka HTML, CSS jest preferowanym sposobem opisu formy prezentacji dokumentów HTML, a wykorzystywanie do tego celu znaczników HTML i ich atrybutów jest zdecydowanie niezalecane. Język HTML powinien być wykorzystany do zdefiniowania struktury dokumentu, a formatowanie poszczególnych elementów powinno być określone za pomocą CSS.

Formatowanie definiowane jest w tzw. arkuszu stylów (ang. style sheet), mającym postać listy reguł stylistycznych (kolejność reguł ma znaczenie). Arkusz stylów może być zawarty w formatowanym dokumencie lub zapisany w odrębnym pliku. Podstawową zaletą arkuszy stylów w odrębnych plikach jest możliwość związania ich z wieloma dokumentami. Dzięki temu łatwo jest zdefiniować, a następnie w sposób spójny modyfikować formatowanie zbioru dokumentów.

CSS pozwala na specyfikację czcionek, stylu tekstu, kolorów, marginesów, obramowania, stylu linii, położenia elementów, itd. Należy podkreślić, że CSS umożliwia znacznie dokładniejszą kontrolę sposobu wyświetlania elementów, aniżeli sam HTML.

Standard CSS jest ciągle rozwijany. Obecnie (wiosna 2006) obowiązuje wersja CSS2 - będąca rozszerzeniem starszej specyfikacji CSS1 (z drobnymi modyfikacjami). Niestety, nie wszystkie mechanizmy CSS1/CSS2 są zaimplementowane w dostępnych obecnie przeglądarkach, a do tego ciągle zdarzają się drobne różnice między przeglądarkami w sposobie interpretacji reguł CSS. Finalizowane są prace nad CSS2.1 – okrojoną wersją CSS2, pozbawioną opcji niezaimplementowanych w aktualnych przeglądarkach, np. dźwiękowej interpretacji dokumentów (uwzględnionej w powstającym CSS3).

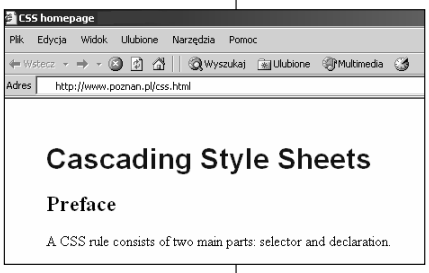
Dołączanie reguł do dokumentu HTML

- Reguły stylistyczne w oddzielnym pliku, wskazanym znacznikiem <LINK REL="stylesheet">
- Reguły stylistyczne zagnieżdżone w dokumencie za pomocą znacznika <STYLE> w sekcji HEAD
- Atrybut STYLE elementów w sekcji BODY, np. <P STYLE='{color: blue}'>

Istnieją trzy sposoby dołączania reguł formatujących CSS do dokumentu HTML. Pierwszy ze sposobów to przygotowanie arkusza stylów w zewnętrznym pliku, a następnie związanie go z dokumentem HTML poprzez umieszczenie w nim odpowiedniego znacznika <LINK>. W tym wypadku ten sam arkusz stylów może być wykorzystany przez wiele dokumentów HTML. Drugi, mniej zalecany sposób polega na zawarciu arkusza stylów wewnątrz dokumentu HTML za pomocą elementu <STYLE> w nagłówku dokumentu (<HEAD>). Trzeci sposób to zawarcie reguły formatującej w definicji elementu za pomocą atrybutu STYLE. Sposób ten można wykorzystać do określania sposobu specyficznego formatowania konkretnego elementu, w uzupełnieniu do ogólnego arkusza stylów związanego z dokumentem.

HTML + CSS – Przykład 1

```
<HTML>
 <HEAD> <TITLE>CSS homepage</TITLE>
  <STYLE type="text/css">
   H1 { color: blue }
   BODY { font-family: "Times";
          font-size: 12pt;
          margin: 3em }
  </STYLE>
 </HEAD>
 <BODY>
  <H1 style='font-family: "Helvetica"'>
      Cascading Style Sheets</H1>
  <H2>Preface</H2>
  <P> A CSS rule consists of two main
      parts: selector and declaration. </P>
 </BODY></HTML>
```



Slajd pokazuje przykład dokumentu HTML z arkuszem stylów CSS zawartym w dokumencie (w elemencie <STYLE> w nagłówku dokumentu). Arkusz stylów określa, że elementy <H1> mają mieć kolor niebieski, a zawartość ciała dokumentu ma być przedstawiona czcionką z rodziny Times o rozmiarze 12 punktów z marginesem o rozmiarze trzech znaków. W ciele dokumentu dla elementu <H1> z tekstem „Cascading Style Sheets” w jego atrybucie STYLE została umieszczona reguła, która nadpisuje ustawienie domyślnego kroju czcionki dla dokumentu, wskazując, że ten konkretny element ma być wyświetlony przy użyciu czcionki z rodziny Helvetica.

HTML + CSS – Przykład 2

```
<HTML>
 <HEAD> <TITLE>CSS homepage</TITLE>
 <LINK rel="stylesheet" href="styl.css" type="text/css">
 </HEAD>
 <BODY>
  <H1 style='font-family: "Helvetica"'>
      Cascading Style Sheets</H1>
  <H2>Preface</H2>
  <P> A CSS rule consists of two main
      parts: selector and declaration. </P>
 </BODY></HTML>
```

styl.css

```
H1   { color: blue }
BODY { font-family: "Times";
       font-size: 12pt;
       margin: 3em }
```


Przykład na slajdzie ilustruje sposób wykorzystania arkusza stylów zapisanego w zewnętrznym pliku. Sam dokument HTML i arkusz stylów są takie same jak w poprzednim przykładzie. Tym razem arkusz jest umieszczony w pliku styl.css, a dokument HTML odwołuje się do niego poprzez element <LINK> w nagłówku. Znaczenie kolejnych atrybutów elementu <LINK> jest następujące:

• REL – informacja o tym że <LINK> wskazuje arkusz stylów, zgodnie z którym sformatowany ma być dokument,

• HREF – adres dokumentu (nazwa pliku jeśli arkusz na tym samym serwerze w tym samym katalogu),

• TYPE – typ MIME arkusza stylów (obecnie dla HTML zawsze text/css).

Format reguł CSS (1/2)

- Reguła = selektor + deklaracja

```
H1 { color: blue }
```

selektor deklaracja

- Selektor zawiera:
  - nazwy elementów (np. H1)
  - nazwy klas (np. .parzysty)
  - identyfikatory (np. #txt15)
  - łączniki (odstęp, >, +)

Reguły stylistyczne CSS składają się z dwóch części. Pierwszą jest selektor wskazujący elementy dokumentu, które mają być sformatowane zgodnie z daną regułą. Gdy wszystkie warunki selektora są spełnione dla danego elementu, mówimy, że element spełnia (ang. matches) selektor. Drugą częścią reguły jest deklaracja opisująca formatowanie.

Selektory zbudowane są z nazw elementów, nazw klas i identyfikatorów, opcjonalnie połączonych łącznikami. Dostępne łączniki to odstęp (spacja) oraz znaki „>” i „+”.

Format reguł CSS (2/2)

- Deklaracja = nazwa właściwości: wartość
- Grupowanie selektorów

```
H1, H2 { color: blue }
```

- Grupowanie deklaracji

```
H1 { color: blue;
font-style: italic }
```


Druga część reguły stylistycznej – deklaracja, obejmuje nazwę właściwości np. „color” i wartość np. „blue”. Dla każdej z właściwości standard CSS określa dopuszczalne wartości.

W przypadku, gdy kilka reguł posiada taką samą deklarację, można zapisać je skrótowo korzystając z możliwości grupowania selektorów, oddzielając je przecinkami w definicji „zgrupowanej” reguły. Podobnie zgrupowane mogą być deklaracje w przypadku kilku reguł posiadających taki sam selektor. W takim wypadku deklaracje są oddzielone od siebie średnikami we wspólnych nawiasach klamrowych.

Selektory (1/2)

- * - każdy element
- E - każdy element E (np. BODY)
- E F - każdy element F zagnieżdżony wewnątrz elementu E
- E > F - każdy element F zagnieżdżony bezpośrednio wewnątrz elementu E
- E + F - każdy element F, który następuje bezpośrednio za elementem E
- A:link - każdy link <A>, który nie został jeszcze odwiedzony
- A:visited - każdy link <A>, który został już odwiedzony

Ten i następny slajd przedstawiają znaczenie najważniejszych konstrukcji selektorów:

• * - spełniony przez każdy element,

• E - spełniony przez każdy element E (np. BODY),

• E F - spełniony przez każdy element F zagnieżdżony wewnątrz elementu E,

• E > F - spełniony przez każdy element F zagnieżdżony bezpośrednio wewnątrz elementu E,

• E + F - spełniony przez każdy element F, który następuje bezpośrednio za elementem E,

• A:link - spełniony przez każdy link <A>, który nie został jeszcze odwiedzony,

• A:visited - spełniony przez każdy link <A>, który został już odwiedzony,

Selektory (2/2)

- E:active - element E właśnie naciskany myszką
- E:hover - element E, nad którym właśnie znajduje się wskaźnik myszki
- E[atr] - elementy E, które posiadają ustawioną (dowolną) wartość atrybutu atr
- E[atr="val"] - elementy E, które posiadają atrybut atr o wartości val
- E[atr~="val"] - elementy E, których wartością atrybutu atr jest lista słów, zawierająca val
- E.val - elementy E, które posiadają atrybut CLASS o wartości val (skrót dla E[class~="val"])
- E#val - elementy E, posiadające atrybut ID o wartości val

• E:active - spełniony przez każdy element E, który jest właśnie naciskany myszką,

• E:hover - spełniony przez każdy element E, nad którym właśnie znajduje się wskaźnik myszki,

• E[atr] - spełniony przez te elementy E, które posiadają ustawioną (dowolną) wartość atrybutu atr,

• E[atr="val"] - spełniony przez te elementy E, które posiadają atrybut atr o wartości val,

• E[atr~="val"] - spełniony przez te elementy E, których wartością atrybutu atr jest lista słów, zawierająca val,

• E.val - spełniony przez te elementy E, które posiadają atrybut CLASS o wartości val (skrót dla E[class~="val"]),

• E#val - spełniony przez te elementy E, które posiadają atrybut ID o wartości val .

W przypadku gdy dwa ostatnie typy selektorów mają być stosowane do wszystkich rodzajów elementów z konkretnej klasy lub o konkretnym identyfikatorze, stosowana jest notacja: *.val i *#val, którą można jeszcze uprościć do postaci .val i #val. Gdy element ma mieć przypisane jednocześnie dwie lub więcej klas stosowana jest notacja .klasa1.klasa2.

Właściwości: czcionka, tekst, kolory i tło

- Właściwości czcionki:
  - font, styl, wariant, grubość, szerokość, rozmiar
- Właściwości tekstu:
  - wcięcie, wyrównanie, dekoracja, cień, odstępy, transformacja wielkości liter
- Właściwości koloru i tła:
  - kolor pierwszego planu, kolor tła, obrazek jako tło

```
BODY { font-family: helvetica, sans-serif;
       text-align: justify; text-decoration: underline;
       color: blue; background-color: yellow; }
```


Właściwości czcionki w CSS to:

(a) rodzina czcionek (font-family); np. serif, sans-serif, konkretne nazwy fontów;

(b) styl (font-style); wartości: normal, italic, oblique;

(c) wariant (font-variant); np. normal, small-caps;

(d) grubość (font-weight); np. normal, bold, 100, 200, 300, itd.;

(e) szerokość (font-stretch); np. condensed, normal, expanded;

(f) rozmiar (font-size); np. small, medium, large, bezwzględny lub względny rozmiar czcionki.

Właściwości tekstu w CSS to:

(a) wcięcie (text-indent); np. 3em (em to rozmiar znaku dla aktualnie wybranej czcionki);

(b) wyrównanie (text-align); wartości: left, right, center, justify;

(c) dekoracja (text-decoration); wartości: none, underline, overline, line-through, blink;

(d) cień (text-shadow); np. 0.2em 0.2em;

(e) odstępy między literami (letter-spacing) i wyrazami (word-spacing); np. 0.1em, 1em;

(f) transformacja wielkości liter (text-transform); wartości: capitalize, uppercase, lowercase;

Właściwości koloru i tła w CSS to:

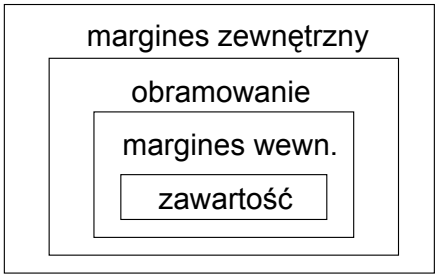
(a) kolor pierwszego planu (color) i tła (background-color); np. red, rgb(255,0,0), #ff0000;

(b) obrazek w tle (background-image, background-repeat, background-attachment, background-position, background); np. background-image: url("marble.gif");

Przykładowa reguła na slajdzie mówi, że ciało dokumentu ma być wyświetlone czcionką „helvetica” lub inną bezszeryfową, tekst ma być wyrównany do lewej i prawej oraz podkreślony, w kolorze niebieskim na żółtym tle.

Właściwości: Box model

- Dotyczy formatowania prostokątów wyznaczonych przez:
  - zawartość
  - margines wewnętrzny
  - obramowanie
  - margines zewnętrzny
- Formatowanie rozmiaru marginesów, grubości, stylu i koloru obramowania




```
P { padding: 3px; margin: 1em 2em
    border-style: solid; border-color: blue;
    border-width: medium; }
```


Model formatowania CSS wiąże z każdym elementem jeden lub wiele zagnieżdżonych prostokątów (stąd angielska nazwa „Box model”). W modelu tym, prostokąt reprezentujący zawartość jest opcjonalnie otoczony:

(a) marginesem wewnętrznym (ang. padding),

(b) obramowaniem (ang. border),

(c) przezroczystym marginesem zewnętrznym (ang. margin).

Dla marginesów (właściwości: padding, margin) CSS pozwala na wyspecyfikowanie ich rozmiaru (bezwzględnego lub w procentach względem elementu otaczającego). Dla obramowania można wskazać grubość linii, jej kolor oraz styl (właściwości: border-width, border-color, border-style).

Istnieje możliwość określenia różnego formatowania dla marginesów i obramowania z poszczególnych stron. Pierwszym sposobem jest podanie od 2 do 4 wartości dla danej właściwości w kolejności: top, right, bottom, left (gdy podane są 2 lub 3 wartości brakująca wartość jest taka sama jak dla przeciwległej strony). Drugim sposobem jest wykorzystanie wariantów właściwości, które swą nazwą wskazują konkretną stronę np. padding-top, padding-right, padding-bottom, padding-left.

Przykładowa reguła przedstawiona na slajdzie mówi, że akapit powinien posiadać z każdej strony margines wewnętrzny o szerokości trzech pikseli, margines zewnętrzny górny i dolny o szerokości jednego znaku, margines zewnętrzny prawy i lewy o szerokości dwóch znaków, a obramowanie linią ciągłą w kolorze niebieskim o średniej grubości.

Właściwości: pozycjonowanie elementów

- Trzy schematy pozycjonowania:
  - Normal flow
  - Floats
  - Absolute positioning
- Algorytm pozycjonowania uwzględnia właściwości:
  - position (static, relative, absolute, fixed)
  - float (left, right, none)

CSS oferuje trzy schematy pozycjonowania:

• Normal flow – normalne, domyślne, pozycja kolejnego elementu jest determinowana położeniem poprzedniego;

• Floats – przesunięcie elementu do lewej/prawej z możliwością „opływania” go zawartością kolejnych elementów;

• Absolute positioning – położenie wskazane poprzez bezwzględne współrzędne (tak naprawdę względem otaczającego pozycjonowanego elementu).

Algorytm pozycjonowania prostokątów związanych z elementami uwzględnia właściwości position i float. Dostępne wartości dla właściwości position to:

• static – wartość domyślna, wybiera normalne pozycjonowanie (normal flow);

• relative – pozycja wyznaczana jak w normal flow, a następnie prostokąt przesuwany względem tej pozycji;

• absolute – pozycja wskazana poprzez podanie współrzędnych wewnątrz otaczającego elementu;

• fixed – ustalona pozycja wskazana poprzez podanie współrzędnych wewnątrz okna w przeglądarce (ang. viewport).

Dostępne wartości dla właściwości float to:

• left – przesunięcie elementu do lewej, zawartość „opływa” go z prawej strony;

• right – przesunięcie elementu do prawej, zawartość „opływa” go z lewej strony;

• none – domyślnie, brak opływania.

W przypadku korzystania z właściwości float konieczne jest wskazanie szerokości elementu właściwością width.

Uwagi o pozycjonowaniu elementów

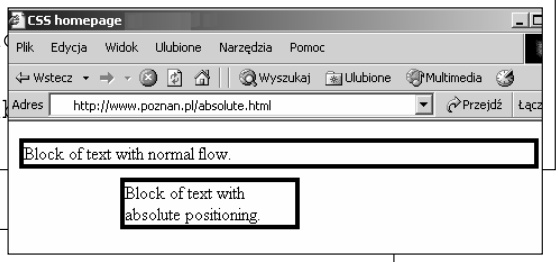
- Pozycjonowanie poprzez CSS a nie tabelki i ramki HTML
  - tabelki dla danych tabelarycznych
  - ramki niezalecane, ale zastępujące je mechanizmy CSS słabo wspierane przez przeglądarki

Wykorzystanie CSS jest zalecaną techniką wymuszania wzajemnego położenia elementów na stronie względem siebie. Przed pojawieniem się CSS twórcy stron często wymuszali konkretne ułożenie elementów na stronie za pomocą tabelek HTML. Obecnie nie jest to zalecane. Tabelki powinny służyć do przedstawiania danych w postaci tabelarycznej. Użycie ich do pozycjonowania komplikuje strukturę dokumentu, a także może powodować problemy z prezentowaniem stron na urządzeniach o małym ekranie (np. telefonach czy PDA)

Pozycjonowanie CSS stanowi też alternatywę dla ramek HTML. Jednym z podstawowych zastosowań ramek było wyświetlanie menu obok zawartości w taki sposób, aby menu pozostawało na swoim miejscu gdy użytkownik przewija ramkę z zawartością. Ten sam efekt można uzyskać poprzez ustalone pozycjonowanie bloku tekstu zawierającego menu. Niestety w chwili obecnej (wiosna 2006) niewiele przeglądarek obsługuje ten mechanizm.

Pozycjonowanie w CSS – Przykład

```
<HTML>
 <HEAD> <TITLE>CSS homepage</TITLE>
 <LINK rel="stylesheet" href="styl.css" type="text/css">
 </HEAD>
 <BODY>
 <DIV class='normalny'>
      Block of text with n
 <DIV class='absolutny'>
      Block of text with a
 </BODY></HTML>
```



styl.css

```
DIV { border: solid }
DIV.absolutny { position: absolute; top: 50 px;
                left: 100 px; width: 10 em }
```


Przykład na slajdzie ilustruje absolutne pozycjonowanie elementów za pomocą CSS. Przykładowy dokument HTML zawiera dwa bloki tekstu (elementy <DIV>). Pierwszemu z nich przypisano klasę „normalny”, a drugiemu „absolutny”. Arkusz stylów określa, że wszystkie elementy DIV mają mieć widoczne obramowanie. Ponadto, dla bloków tekstu DIV o klasie „absolutny” specyfikuje się ich położenie poprzez podanie współrzędnych lewego górnego narożnika w pikselach oraz szerokość w znakach.

Dziedziczenie reguł

- Dziedziczenie właściwości z elementów nadrzędnych

Dokument HTML

```
<H1>Oto przykład <EM>dziedziczenia</EM></H1>
```

Arkusz stylów

```
H1 { color: blue }
```

Oto przykład *dziedziczenia*

Wartości niektórych właściwości są dziedziczone z elementu nadrzędnego w strukturze dokumentu, w przypadku, gdy nie zostały podane jawnie. Specyfikacja CSS określa, dla których właściwości wartości powinny być dziedziczone, a dla których nie. W przypadku, gdy autor arkusza stylów chce podkreślić, że wartość danej właściwości ma być odziedziczona lub wskazać, że odziedziczona ma być wartość właściwości, która normalnie nie podlega dziedziczeniu, może podać „inherit” jako wartość właściwości.

Przykład na slajdzie ilustruje dziedziczenie wartości właściwości koloru pierwszego planu, która standardowo podlega dziedziczeniu. W dokumencie HTML element <EM> jest zawarty w elemencie <H1>. Arkusz stylów specyfikuje jedynie, że element <H1> ma być zaprezentowany niebieskim tekstem, a nie ma reguły dotyczącej koloru elementu <EM>. Zatem, element <EM> dziedziczy kolor z elementu <H1> go otaczającego.

Kaskada

- Kaskada reguł z trzech źródeł:
  - autor
  - użytkownik
  - Przeglądarka
- Ważne reguły

```
H1 { color: blue ! important}
```

- Importowanie reguł

```
@import url("ogolny.css")
```


Rozwinięcie skrótu CSS to „kaskadowe arkusze stylów”. Termin „kaskada” dotyczy wzajemnej interakcji reguł stylistycznych pochodzących z trzech możliwych źródeł:

1. reguły autora – zdefiniowane w dokumencie lub arkuszu stylów, do którego dokument się odwołuje;

2. reguły użytkownika – w zależności od przeglądarki dostarczane przez użytkownika w formie pliku lub poprzez interaktywne ustawienie preferencji stylu wyświetlania;

3. reguły przeglądarki – „wbudowane” w przeglądarkę, zgodne z zaleceniami CSS odnośnie domyślnego formatowania elementów.

Ogólna zasada mówi, że reguły autora mają większą wagę niż reguły użytkownika, a te z kolei mają większą wagę niż reguły przeglądarki. Zarówno autorzy jak i użytkownicy mogą wzmacniać siłę swoich reguł wskazując je jako ważne (ang. important). Reguły ważne mają większą wagę niż reguły normalne z tego samego źródła. Ważna reguła użytkownika ma większą wagę niż ważna reguła autora (od CSS2).

Arkusze stylów mogą importować reguły z innych arkuszy za pomocą słowa kluczowego „@import”, po którym następuje adres URL wskazujący importowany arkusz. Mechanizm ten pozwala na strukturalizację złożonych arkuszy stylów.

Algorytm kaskady

- Znajdź wszystkie deklaracje odnoszące się do elementu
- Sortuj wg oznaczenia ważności i pochodzenia
  - deklaracje przeglądarki
  - normalne deklaracje użytkownika
  - normalne deklaracje autora
  - ważne deklaracje autora
  - ważne deklaracje użytkownika

  waga
- Sortuj wg specyficzności selektorów reguł
- Sortuj wg kolejności wystąpienia

Slajd przedstawia kompletny algorytm kaskady determinujący ostateczną wartość właściwości danego elementu. W pierwszym kroku znajdowane są wszystkie deklaracje odnoszące się do elementu. Następnie znalezione deklaracje są sortowane wg oznaczenia ważności i pochodzenia. W rosnącym porządku następują po sobie kolejno: deklaracje przeglądarki, normalne deklaracje użytkownika, normalne deklaracje autora, ważne deklaracje autora, ważne deklaracje użytkownika. W kroku trzecim reguły o tej samej ważności i tym samym pochodzeniu są sortowane wg współczynnika specyficzności. Ideą stojącą za współczynnikiem specyficzności jest preferowanie reguł, których selektory w sposób bardziej szczegółowy wskazują dany element (szczegółowy sposób wyznaczania wartości współczynnika na następnym slajdzie). Ostatecznie, jeśli dwie deklaracje mają jednakowe pochodzenie, ważność i specyficzność, decyduje kolejność reguł. Wygrywa reguła wyspecyfikowana jako ostatnia, przy czym przyjmuje się, że reguły z zaimportowanych arkuszy poprzedzają reguły z bieżącego arkusza.

Specyficzność reguł

- Współczynnik specyficzności (ang. specificity) wyznaczany dla selektora reguły uwzględnia:
  - (a) czy do wskazania elementu wykorzystany jest selektor czy atrybut style
  - (b) liczbę identyfikatorów w selektorze
  - (c) liczbę innych atrybutów i pseudo-klas w selektorze
  - (d) liczbę nazw elementów i pseudo-elementów w selektorze
- Współczynnik ma postać liczby (a)(b)(c)(d)

```
UL OL LI.red {...}
/* a=0 b=0 c=1 d=3 -> specificity =   13 */
```


Współczynnik specyficzności (ang. specificity) wyznaczany jest dla selektora reguły i uwzględnia:

(a) czy do wskazania elementu wykorzystany jest selektor czy atrybut STYLE,

(b) liczbę identyfikatorów w selektorze,

(c) liczbę innych atrybutów i pseudo-klas w selektorze,

(d) liczbę nazw elementów i pseudo-elementów w selektorze.

Współczynnik ma postać liczby (a)(b)(c)(d) w systemie o odpowiednio dużej podstawie.

U dołu slajdu pokazany został przykład wyznaczenia wartości współczynnika specyficzności dla złożonego selektora reguły. Ponieważ przykładowy selektor zawiera trzy nazwy elementów i jedną nazwę klasy, jego współczynnik specyficzności wynosi 13.

Extensible Markup Language (XML)

- Prosty, elastyczny format tekstowy
- Wywiedziony z języka SGML
- Opisuje obiekty nazywane dokumentami XML, na które składają się dane i znaczniki
- Założenia:
  - format użyteczny w Internecie
  - format łatwy w przetwarzaniu maszynowym
  - format zrozumiały dla człowieka
  - łatwość tworzenia dokumentów

XML (ang. Extensible Markup Language) to prosty, elastyczny format tekstowy. Formalnie, XML stanowi podzbiór języka Standard Generalized Markup Language (SGML) (ISO 8879:1986). Nie jest więc pomysłem nowym, ale w przeciwieństwie do SGML szybko stał się bardzo popularny i powszechnie wykorzystywany ze względu na znaczące uproszczenie w porównaniu z SGML i zorientowanie na Internet.

XML opisuje klasę obiektów (ang. data objects) nazywanych dokumentami XML. Dokumenty XML zawierają dane i znaczniki. Są więc podobne do dokumentów HTML, ale między HTML i XML istnieją fundamentalne różnice. Znaczniki w XML służą do reprezentacji struktury i znaczenia danych i nie są z nimi związane domyślne reguły prezentacji i formatowania danych (jak w HTML). XML, w przeciwieństwie do HTML, nie posiada predefiniowanego zestawu znaczników o określonym znaczeniu.

Pierwszym założeniem twórców XML było to, aby format był odpowiedni do wykorzystania w Internecie. Kolejne istotne założenia to łatwość w przetwarzaniu maszynowym, czytelność dla człowieka i łatwość tworzenia dokumentów. Wymienione założenia XML z całą pewnością spełnia. W połączeniu z XSL umożliwia przygotowanie danych do publikacji w Internecie z możliwością automatycznej transformacji do różnych formatów prezentacji. XML jest łatwy do przetwarzania w aplikacjach, ze względu do bogactwo bibliotek parserów dla popularnych języków programowania. Jako prosty format tekstowy oparty o znaczniki opisujące znaczenie danych i strukturę dokumentu, jest to format zrozumiały dla człowieka. Tworzenie dokumentów XML jest łatwe zarówno za pomocą edytorów tekstowych (zwykłych i zorientowanych na wsparcie XML) jak i programowo z poziomu aplikacji. Dodatkowo istnieją narzędzia i standardy dotyczące eksportu danych w formacie XML np. z relacyjnych baz danych. W połączeniu z łatwością parsowania, czyni to z XML idealny format wymiany danych.

Zastosowania XML

- Źródłowy format do publikowania informacji
- Format wymiany danych
- Format składowania danych w bazach danych
- Format plików konfiguracyjnych

Pierwszym planowanym zastosowaniem XML było publikowanie danych w Internecie. XML spełnia to zadanie w połączeniu z językiem XSL, dotyczącym transformacji i formatowania dokumentów XML. XML nie jest w stanie wyprzeć języka HTML, ale jest dobrym rozwiązaniem gdy te same dane mają być publikowane na wiele sposobów, w tym w Internecie, w postaci dokumentów HTML. W tym wypadku dane do publikacji mogą być przygotowane w formie dokumentów XML, a sposób ich prezentacji w różnej formie może być opisany w języku XSL.

Najważniejszym obecnie zastosowaniem XML jest jednak wymiana danych między systemami i aplikacjami. XML ze względu na łatwość programowego generowania dokumentów oraz później ich parsowania w aplikacji jest wręcz idealnym formatem wymiany danych i w związku z tym w dużym stopniu wyparł z rynku inne, konkurencyjne w tym zakresie rozwiązania. XML jest wykorzystywany zarówno do wymiany danych między aplikacjami poprzez sieć Internet jak i off-line na zasadzie eksportu danych do pliku i następnie importowania go do innego systemu.

Zyskującym na znaczeniu w ostatnich latach zastosowaniem formatu XML jest wykorzystanie go jako formatu składowania danych w bazach danych. Po pierwsze, XML umożliwia reprezentację danych o strukturze określonej nie tak ściśle jak wymagają tego relacyjne i obiektowe bazy danych (tzw. danych semistrukturalnych). Po drugie, powszechność istniejących dokumentów XML stworzyła zapotrzebowanie na składowanie i przeszukiwanie kolekcji dokumentów XML. Składowanie danych XML w postaci XML pozwala uniknąć kosztu transformacji danych XML przy wstawianiu ich i pobieraniu do/z bazy danych.

Mówiąc o zastosowaniach XML nie można zapomnieć o jego powszechnym wykorzystaniu jako formatu plików konfiguracyjnych różnego rodzaju aplikacji i narzędzi. Zalety XML w tym obszarze zastosowań to czytelność dla człowieka i łatwość parsowania w aplikacji.

Struktura dokumentu XML

- Prolog (opcjonalny) + ciało

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<company>
 <department symbol="M">
  <dname>Marketing</dname>
  <employees>
   <employee>Jones</employee>
   <employee>Smith</employee>
  </employees>
 </department>
</company>
```


Każdy dokument XML posiada ciało w postaci odpowiednio zagnieżdżonych elementów. Elementy definiowane są za pomocą znaczników (otwierających i zamykających). Elementy mogą posiadać zawartość i atrybuty. Całe ciało dokumentu musi być zawarte w jednym elemencie najwyższego poziomu. W przykładzie na slajdzie takim elementem jest <company>.

Ciało dokumentu może być poprzedzone prologiem. Prolog może zawierać następujące, opcjonalne konstrukcje:

(a) deklaracja XML (zawsze w pierwszym wierszu dokumentu) określająca wersję XML (version) i opcjonalnie kodowanie znaków (encoding);

(b) deklaracja typu dokumentu (ang. Document Type Declaration - DTD), opisująca użyte znaczniki oraz ich gramatykę;

(c) instrukcje przetwarzania np. instrukcja wskazująca arkusz stylów, zgodnie z którym ma być sformatowany dokument.

Przykładowy dokument na slajdzie zawiera prolog, w którym obecna jest jedynie deklaracja XML informująca o tym, że dokument jest w wersji XML 1.0, a jego kodowanie znaków to ISO-8859-1.

Obowiązujące wersje XML to 1.0 i 1.1. Wersja 1.1 różni się od 1.0 obsługą Unicode, nie polegając na konkretnej wersji Unicode, ale zawsze umożliwiając wykorzystanie najświeższej. Zalecane jest tworzenie i generowanie dokumentów w wersji 1.0 jeśli nie korzysta się z właściwości 1.1. W przypadku gdy dokument XML nie zawiera deklaracji XML, przyjmuje się że jest on w wersji 1.0. Domyślne kodowanie znaków dla XML to UTF-8.

Dokumenty poprawne i dobrze sformułowane

- Dokument XML jest poprawny (ang. valid) jeśli posiada DTD i jest z nim zgodny
- Alternatywą dla DTD jest XML Schema
- Jeśli dokument nie posiada DTD, można jedynie sprawdzić czy jest poprawnie sformułowany (ang. well-formed)
  - poprawne sformułowanie jest warunkiem bycia dokumentem XML

Dokument jest poprawny (ang. valid) jeśli posiada DTD i jest z nim zgodny. DTD jest sposobem opisu struktury dokumentów (gramatyki języka), obejmującym dopuszczalne znaczniki, ich atrybuty oraz sposób ich zagnieżdżania. DTD jest odpowiedni do definiowania struktury dokumentów tekstowych, gdyż nie posiada pełnoprawnego systemu typów danych, pozwalającego np. na rozróżnienie między tekstami, liczbami i datami. Alternatywnym, nowszym i bardziej elastycznym sposobem opisu struktury dokumentów XML są schematy XML (XML Schema). Standard XML Schema jest wykorzystywany przede wszystkim w bazodanowych zastosowaniach XML-a. Podobnie jak w przypadku DTD, jeśli dokument posiada schemat XML Schema, można stwierdzić czy dokument jest poprawny w kontekście tego schematu czy nie.

Jeśli z dokumentem nie został związany opis jego struktury (w postaci DTD i/lub XML Schema), można jedynie stwierdzić czy dokument XML jest poprawnie sformułowany (ang. well-formed). Poprawne sformułowanie wg reguł XML jest tak naprawdę warunkiem koniecznym, aby dokument tekstowy ze znacznikami był dokumentem XML.

Reguły poprawnego sformułowania dla formatu XML to:

(a) wszystkie znaczniki zamknięte (np. <P>...</P>). Dla elementów pustych dopuszczalny specjalny skrócony zapis np.
;

(b) elementy prawidłowo zagnieżdżone np. „<B> one <I> two </B> three </I>” jest niepoprawne;

(c) nazwy znaczników otwierających i zamykających zgodne pod względem wielkości liter np. „<B><i> one </I></B>” jest niepoprawne;

(d) wartości atrybutów w apostrofach/cudzysłowach, nawet gdy wartość jest liczbą np. <IMG SRC=”icon.gif” WIDTH=”10” HEIGHT=”20”>;

(e) dokument zawiera dokładnie jeden główny element (niezawarty w innym elemencie).

Document Type Declaration (DTD)

company.xml

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<!DOCTYPE company SYSTEM "company.dtd">
<company>
<department symbol="M">
<dname>Marketing</dname>
<employees>
<employee>Jones</employee>
<employee>Smith</employee>
</employees>
</department>
<department>
...
</department>
</company>
```

+: 1 lub więcej
*: 0 lub więcej
?: 0 lub 1

company.dtd

```
<!ELEMENT company (department)+>
<!ELEMENT department (dname, employees)>
<!ELEMENT dname (#PCDATA)>
<!ELEMENT employees (employee)+>
<!ELEMENT employee (#PCDATA)>
<!ATTLIST department symbol CDATA #REQUIRED>
```


Slajd przedstawia przykładowy DTD oraz dokument XML, który się do niego odwołuje i jest w jego kontekście poprawny. DTD znajduje się w oddzielnym pliku, który w dokumencie jest wskazany jako zewnętrzny DTD poprzez jego lokalizację w dyrektywie !DOCTYPE.

Znaczenie poszczególnych wierszy przykładowego DTD przedstawionego na slajdzie jest następujące:

1. Element <company> zawiera jeden lub więcej elementów <department> („+” 1 lub więcej wystąpień elementu, „*” 0 lub więcej wystąpień, a „?” 0 lub 1 wystąpienie).

2. Element <department> zawiera następujące po sobie elementy <dname> i <employees> (kolejność ma znaczenie).

3. Element <dname> ma zawartość tekstową.

4. Element <employees> zawiera jeden lub więcej elementów <employee>.

5. Element <employee> ma zawartość tekstową.

6. Element <department> posiada jeden atrybut „symbol”, typu tekstowego, obowiązkowy.

Przykład na slajdzie pokazuje DTD zewnętrzny w stosunku do dokumentu. Alternatywą jest zawarcie całego DTD wewnątrz dokumentu XML lub wykorzystanie połączenia zewnętrznego i wewnętrznego DTD.

Przestrzenie nazw (XML Namespaces)

- Rozwiązanie problemu konfliktu nazw znaczników

```
<html xmlns="http://www.w3.org/HTML/1998/html4"
      xmlns:emp="http://mysite.com/employees">
 <head><title>Employees</title></head>
 <body>
  <emp:list>
   <emp:head>Employees</emp:head>
   <table>
    <tr><td><emp:lastname>Scott</emp:lastname></td>
        <td><emp:salary>2300</emp:salary></td></tr>
   </table>
  </emp:list>
 </body>
</html>
```


Format XML nie zawiera predefiniowanego zestawu znaczników. Nazwy znaczników i ich znaczenie są ustalane przez twórców dokumentów i aplikacji. Powstaje wiele słowników dziedzinowych w postaci zbiorów znaczników. Autorzy dokumentów powinni móc korzystać w jednym dokumencie z wielu słowników i dodawać własne. W takim wypadku pojawia się ryzyko konfliktów nazw znaczników.

Rozwiązaniem problemu potencjalnych konfliktów nazw znaczników są przestrzenie nazw (XML Namespaces). W przypadku wykorzystywania w dokumencie kilku definiowanych niezależnie zbiorów znaczników, z każdym zbiorem związana jest przestrzeń nazw określająca ich pochodzenie. Z przestrzeniami nazw wykorzystywanymi w dokumencie wiązane są prefiksy, którymi następnie poprzedzane są nazwy znaczników. Jedna przestrzeń nazw może być wskazana jako domyślna – znaczniki z niej pochodzące nie będą poprzedzane prefiksem.

Definiowanie przestrzeni nazw odbywa się przez podanie specjalnego atrybutu xmlns:prefiks_przestrzeni. Wartością atrybutu jest URI przestrzeni nazw (nazwa przestrzeni). Przestrzeń zdefiniowana w elemencie nadrzędnym może być wykorzystywana w podrzędnych.

Slajd przedstawia przykład dokumentu XML wykorzystującego znaczniki z dwóch przestrzeni nazw. Przestrzenią domyślną (bez prefiksu) jest "http://www.w3.org/HTML/1998/html4". Druga przestrzeń to "http://mysite.com/employees" z odpowiadającym jej prefiksem „emp”. Obie przestrzenie definiują znacznik <head> przypisując mu inne znaczenie. Dzięki przestrzeniom nazw i odpowiadającym im prefiksom możliwe jest jednoczesne wykorzystanie obu znaczników <head> w sposób umożliwiający ich rozróżnienie aplikacji, która będzie przetwarzać dokument.

Extensible HyperText Markup Language (XHTML)

- HTML sformułowany jako XML
- Następca HTML 4.01
- Łączy siłę HTML 4 z mocą XML
  - łatwość przetwarzania maszynowego
  - zagnieżdżanie MathML, SMIL, SVG
- XHTML Basic – rozszerzalny podzbiór XHTML opracowany dla urządzeń takich jak telefony, PDA
- Różnice w składni między HTML i XHTML

Extensible HyperText Markup Language (XHTML) to HTML sformułowany jako XML, w pewnych obszarach okrojony, a w innych rozszerzony. Rekomendacja (W3C) XHTML 1.0 może być traktowana jako następca HTML 4.01 (HTML4 jest aplikacją SGML, ale nie XML!).

XHTML „łączy siłę HTML 4 z mocą XML”. HTML 4 jest sprawdzonym językiem do publikacji treści w Internecie. Z kolei niekwestionowaną zaletą XML jest bardziej uporządkowana składnia i w konsekwencji łatwość przetwarzania maszynowego. Ponadto, dzięki dostosowaniu składni do reguł XML, XHTML w przeciwieństwie do HTML daje autorom możliwość równoczesnego wykorzystania w dokumencie innych języków znaczników opartych o XML, np. MathML – do reprezentacji wyrażeń matematycznych; SMIL – do tworzenia interaktywnych prezentacji audiowizualnych; SVG – do grafiki wektorowej.

Język XHTML jest bardzo ważny z punktu widzenia aplikacji dla urządzeń mobilnych typu telefony komórkowe czy PDA. W ramach języka XHTML wyróżniono jego podzbiór: XHTML Basic, który ma stanowić bazę dla rozszerzeń zorientowanych na poszczególne typy specjalistycznych urządzeń. Rozszerzenie o nazwie XHTML-MP (Mobile Profile) ma szansę stać się następcą języka WML, wykorzystywanego w aplikacjach dla urządzeń mobilnych.

Dokument XHTML musi być poprawnie sformułowanym dokumentem XML, stąd szereg różnic składniowych między HTML a XHTML. Przykładowo, ponieważ w XML wielkość liter ma znaczenie przyjęto, że w XHTML nazwy elementów i atrybutów są pisane małymi literami.

XHTML - Przykład

1
```
<?xml version="1.0" encoding="windows-1250"?>
```
2
```
<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"DTD/xhtml1-strict.dtd">
```
3
```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="pl">
 <head><title>Przykład XHTML</title></head>
 <body>
  <p>
   Wybierz kolor: <br/><br/>
   <select name="kolor">
    <option selected="selected">Czerwony</option>
    <option>Biały</option><option>Zielony</option>
   </select>
  </p>
 </body>
</html>
```
4 5 6

Na slajdzie pokazano przykład dokumentu XHTML, ilustrujący różnice składniowe między HTML a XHTML. Różnice podkreślone w przykładzie to:

1. Deklaracja XML (dokumenty XHTML są dokumentami XML).

2. Deklaracja typu dokumentu (z odwołaniem do publicznego DTD opisującego gramatykę XHTML, opublikowanego przez W3C).

3. Wskazanie przestrzeni nazw dla znaczników XHTML (potencjalnie umożliwia użycie w dokumencie znaczników innych języków opartych o XML, z innych przestrzeni nazw).

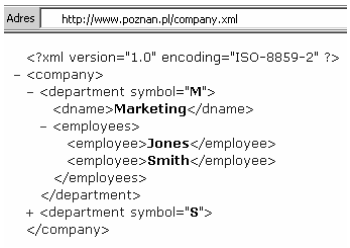
4. Notacja skrócona dla znacznika bez zawartości.

5. Atrybut w XML musi mieć podaną wartość.

6. Jeśli pojawił się znacznik otwierający element, musi pojawić się również znacznik zamykający ten element. Elementy w dokumencie muszą być poprawnie zagnieżdżone.

Formatowanie dokumentów XML

- Znaczniki XML opisują strukturę i semantykę danych
- Nie ma w XML predefiniowanych znaczników opisujących formatowanie
  - brak domyślnego sposobu prezentacji



- Sposób prezentacji dokumentu XML określa dołączony arkusz stylów
  - CSS – Cascading Style Sheets
  - XSL – Extensible Stylesheet Language

W języku XML znaczniki opisują strukturę i semantykę danych, a nie sposób ich prezentacji. Nie ma w XML predefiniowanych znaczników opisujących formatowanie dokumentu. Ze znacznikami XML nie jest też związany żaden domyślny sposób prezentacji. Edytor czy przeglądarka, w której zostanie otwarty dokument XML, zaprezentuje jego źródło w niezmienionej postaci, ewentualnie wykorzystując różne kolory do podkreślenia składni XML. Przykład dokumentu XML oglądanego w przeglądarce Microsoft Internet Explorer został pokazany na slajdzie. Microsoft Internet Explorer prezentuje dokument XML, uwypuklając składnię kolorami, a ponadto umożliwiając zwijanie i rozwijanie poszczególnych elementów, co ułatwia nawigowanie po dużych dokumentach.

Pożądany sposób prezentacji dokumentu XML określa się poprzez dołączenie do dokumentu arkusza stylów. Arkusze stylów dla XML można wprawdzie tworzyć w omawianym wcześniej języku CSS, ale lepszym rozwiązaniem jest wykorzystanie do tego celu opracowanego specjalnie dla XML języka XSL. Podstawową przewagą XSL nad CSS jest to, że umożliwia on nie tylko wskazanie sposobu formatowania elementów źródłowego dokumentu XML, ale również transformację jego struktury na potrzeby prezentacji.

Extensible Stylesheet Language (XSL)

- Standard do definiowania transformacji i prezentacji dokumentów XML
- Składa się z 3 części:
  - XSL Transformations (XSLT) – język opisu transformacji dokumentów
  - XML Path Language (XPath) – język wyrażeń do wskazywania poszczególnych części dokumentu
  - XSL Formatting Objects (XSL-FO) - słownik znaczników do opisu formatowania
- Arkusze stylów XSL są dokumentami XML

XSL (ang. Extensible Stylesheet Language) to standard do definiowania transformacji i prezentacji dokumentów XML. XSL składa się z 3 części:

1. języka opisu transformacji dokumentów - XSL Transformations (XSLT),

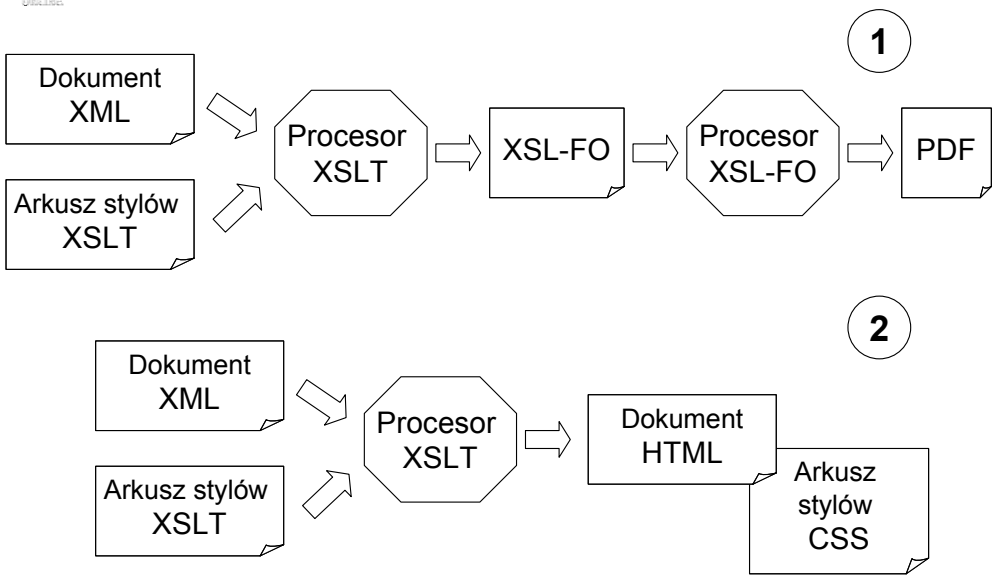
2. języka wyrażeń do wskazywania poszczególnych części dokumentu - XML Path Language (XPath),

3. słownika znaczników do opisu formatowania –XSL Formatting Objects (XSL-FO).

Arkusze stylów są tworzone w języku XSLT i opisują transformację dokumentu źródłowego w dokument z zawartymi informacjami o jego formatowaniu. Formatowanie dokumentu wynikowego może być zrealizowane poprzez umieszczenie w nim (w wyniku transformacji) znaczników XSL-FO. XPath pełni rolę usługową i jest wykorzystywany w konstrukcjach XSLT do adresowania (wyszukiwania) poszczególnych części dokumentu źródłowego. XPath jako język wyrażeń jest wykorzystywany nie tylko w ramach XSL. Stał on się podstawą dla języka zapytań dla danych XML o nazwie XQuery.

Arkusze stylów XSL są dokumentami XML, a więc dokument źródłowy ma ten sam format co arkusz opisujący sposób jego prezentacji (inaczej niż w przypadku HTML i CSS!).

Transformacje XSLT



Sposoby transformacji dokumentów XML są definiowane w języku XSLT w formie arkuszy stylów. Transformacja realizowana jest przez tzw. procesory XSLT, które wczytują dokument XML oraz arkusz stylów XSLT i generują wyjściowy dokument XML, HTML lub zwykły dokument tekstowy. Na slajdzie przedstawione są dwa typowe scenariusze transformacji XSLT:

1. Pierwszy scenariusz to transformacja w pełni wykorzystująca możliwości standardu XSL w oparciu o współpracę XSLT i XSL-FO. W tym wypadku wynikiem transformacji XSLT źródłowego dokumentu jest dokument XML zawierający znaczniki formatujące XSL-FO. Dokument XSL-FO musi być następnie przetworzony procesorem XSL-FO. Wynikiem działania procesora XSL-FO może być wizualizacja dokumentu na ekranie, wydruk na drukarce lub plik w formacie odpowiednim do druku, typowo PDF.

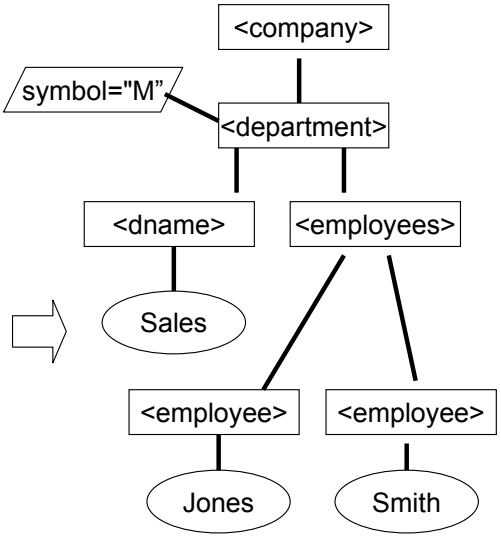
2. Scenariusz alternatywny to transformacja XSLT do postaci dokumentu HTML. Wygenerowany dokument HTML może odwoływać się do dostarczonego niezależnie arkusza stylów CSS. Jest to wygodna metoda definiowania sposobu prezentacji dokumentów XML w Internecie. Często zamiast HTML generowany jest XHTML, gdyż nie wszystkie procesory XSLT muszą obsługiwać HTML jako format wyjściowy. Określając ten drugi scenariusz jako alternatywny, należy przyznać, że historycznie był on wykorzystywany wcześniej niż scenariusz pierwszy i jest do dziś bardziej powszechny. Powodem takiego stanu rzeczy jest fakt, że specyfikacja XSLT i narzędzia ją wspierające były dostępne kilka lat wcześniej niż została zatwierdzona specyfikacja XSL-FO. Również dziś wsparcie dla XSLT jest o wiele bardziej powszechne niż dla XSL-FO.

Dostępnych jest wiele procesorów XSLT, zarówno komercyjnych jak i open source. Procesory XSLT są implementowane jako biblioteki dla różnych języków programowania i systemów operacyjnych np. Xalan dla Javy i C++, SAXON dla Javy, MSXML dla systemu Microsoft Windows (dostępny na platformie .NET). Procesory XSLT są również wbudowane w popularne przeglądarki internetowe (Internet Explorer, Mozilla, Opera).

Co widzi procesor XSLT?

- Procesor XSLT „widzi” drzewo dokumentu XML

```
<company>
 <department symbol="M">
  <dname>Sales</dname>
  <employees>
   <employee>Jones</employee>
   <employee>Smith</employee>
  </employees>
 </department>
</company>
```

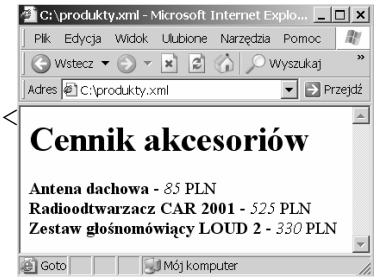


Dokument XML, ze względu na ograniczenia dotyczące poprawnego zagnieżdżania elementów, może być przedstawiony w formie drzewa. Podstawowa metoda parsowania dokumentu XML generuje w pamięci odpowiadającą mu strukturę drzewiastą. Drzewo zawiera różnego rodzaju węzły, w tym węzły elementów, węzły atrybutów i węzły tekstowe. Na slajdzie pokazano przykładowy dokument XML i odpowiadające mu drzewo będące wynikiem parsowania dokumentu.

Procesory XSLT są wykorzystywane w połączeniu z parserami XML w taki sposób, że procesor XSLT otrzymuje na wejściu drzewo źródłowego dokumentu XML i drzewo arkusza stylów. Wynikiem transformacji XSLT jest początkowo drzewo dokumentu wynikowego, które następnie jest wysyłane na wyjście w formie tekstowej.

XSLT – Przykład (1/3)

```
<?xml version="1.0" encoding="windows-1250" ?>
<?xml-stylesheet type="text/xsl" href="produkty.xsl"?>
<cennik>
<produkt>
  <nazwa>Antena dachowa</nazwa>
  <symbol>1709765</symbol>
  <cena>85</cena>
 </produkt>
 <produkt>
  <nazwa>Radioodtwarzacz CAR 2001<
  <symbol>3209765</symbol>
  <cena>525</cena>
 </produkt>
...
</cennik>
```



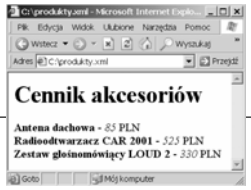
Slajd przedstawia przykładowy dokument XML, zawierający cennik akcesoriów samochodowych oraz docelowy efekt prezentacji dokumentu w przeglądarce Microsoft Internet Explorer. Formatowanie zostało zrealizowane poprzez transformację XSLT do HTML zdefiniowaną w arkuszu stylów umieszczonym w pliku produkty.xsl. Generowany dokument HTML jest w postaci poprawnie sformułowanego dokumentu XML (XHTML).

Wyróżniony fragment dokumentu to instrukcja przetwarzania, wiążąca z dokumentem arkusz stylów XSL, który ma być wykorzystany do jego transformacji. Atrybut „type” wskazuje typ arkusza stylów. Wg specyfikacji XSL jego wartością powinien być „text/xml”, Internet Explorer spodziewa się jednak niestandardowego typu „text/xsl”. Atrybut „href” wskazuje lokalizację arkusza stylów w formie adresu URL. Jeśli, tak jak w przykładzie, arkusz stylów znajduje się w tym samym folderze/katalogu co formatowany dokument, wystarczy podać samą nazwę pliku.

XSLT – Przykład (2/3)

1
2
3

```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet
     xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
     version="1.0">
  <xsl:template match="cennik">
    <HTML><BODY><H1>Cennik akcesoriów</H1>
    <xsl:apply-templates/></BODY></HTML>
  </xsl:template>
  <xsl:template match="produkt">
    <xsl:apply-templates/><BR/>
  </xsl:template>
...
```



Slajd przedstawia pierwszą część arkusza stylów zawierającego definicję transformacji XSLT. Język XSLT jest językiem deklaratywnym, choć zawiera pewne konstrukcje proceduralne (instrukcję pętli FOR). Zalecanym stylem definiowania transformacji XSLT jest styl deklaratywny, w którym opis transformacji ma postać zbioru reguł. Reguła jest definiowana za pomocą elementu <xsl:template>, którego atrybut „match” zawiera wyrażenie XPath, wybierające przetwarzane elementy. W przedstawionym na slajdzie arkuszu stylów wykorzystywane są tylko najprostsze wyrażenia XPath, wybierające węzły elementów dzieci i węzły tekstowe.

Znaczenie poszczególnych fragmentów przykładowego arkusza stylów jest następujące:

1. Element główny arkusza <xsl:stylesheet> deklarujący przestrzeń nazw dla znaczników XSLT. Zwyczajowy prefiks przestrzeni nazw XSLT to „xsl”.

2. Reguła mówiąca, że każde wystąpienie elementu <cennik> ma być zastąpione przez:

<HTML><BODY><H1>Cennik akcesoriów</H1>*</BODY></HTML>,

gdzie w miejscu „*”, ma być umieszczona zawartość będąca wynikiem dopasowania reguł arkusza do zawartości elementu <cennik>.

3. Reguła mówiąca, że każde wystąpienie elementu <produkt> ma być zastąpione przez:

*
,

gdzie w miejscu „*”, ma być umieszczona zawartość będąca wynikiem dopasowania reguł arkusza do zawartości elementu <produkt>.

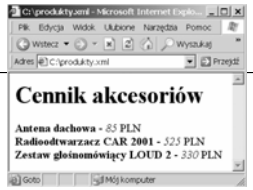
XSLT – Przykład (3/3)

```
...
  <xsl:template match="nazwa">
      <B><xsl:value-of select="text()"/></B> -
  </xsl:template>
  <xsl:template match="symbol">
  </xsl:template>
  <xsl:template match="cena">
      <I><xsl:value-of select="text()"/></I> PLN
  </xsl:template>
</xsl:stylesheet>
```

4

5

6



Slajd przedstawia pozostałą część arkusza stylów, którego początek został przedstawiony na poprzednim slajdzie. Znaczenie poszczególnych fragmentów arkusza stylów jest następujące:

4. Reguła mówiąca, że każde wystąpienie elementu <nazwa> ma być zastąpione przez:

<B>*</B> -,

gdzie w miejscu „*”, ma być umieszczona zawartość tekstowa elementu <nazwa>.

5. Reguła mówiąca, że wystąpienia elementu <symbol> mają nie wpływać na zawartość dokumentu wynikowego. Taka reguła bez ciała jest niezbędna w celu pominięcia w wynikowym dokumencie zawartości elementów <symbol>, ze względu na istnienie domyślnych reguł XSLT, z których jedna mówi, że zawartość elementów, do których nie pasuje żadna reguła jawnie zdefiniowana w arkuszu, ma w formie tekstowej być wyprowadzona na wyjście.

6. Reguła mówiąca, że każde wystąpienie elementu <cena> ma być zastąpione przez:

<I>*</I> PLN,

gdzie w miejscu „*”, ma być umieszczona zawartość tekstowa elementu <cena>.

Obiekty formatujące XSL-FO

- Język opisu wyglądu (prezentacji) dokumentów XML
- Wiele rozwiązań wspólnych z CSS2
- W praktyce wykorzystywany w połączeniu z XSLT
- Obecnie używany głównie do przygotowywania dokumentów XML do druku (w tym konwersji do PS, PDF)
- Znacznie mniej popularny niż XSLT

Obiekty formatujące XSL-FO (ang. XSL Formatting Objects) to język znaczników XML służący do opisu sposobu prezentacji dokumentów XML. XSL-FO wykazuje w zakresie możliwości formatowania dużo podobieństw do CSS (w wersji 2.0 i wyższych), choć składnia obu języków jest zupełnie inna. W porównaniu z CSS, XSL-FO umożliwia bardziej szczegółowe zdefiniowanie formatu dla dokumentów przygotowywanych do druku dzięki konstrukcjom do opisu paginacji, układu poszczególnych stron, itp.

Składnia XSL-FO jest dość skomplikowana, a przy tym niezbyt oszczędna. W związku z tym dokumenty XSL-FO są nieco „rozdmuchane” i trudno się je tworzy „ręcznie”. Dokumenty XSL-FO wg twórców standardu XSL mają być generowane automatycznie w wyniku transformacji XSLT i taka jest też praktyka.

Obecnie XSL-FO jest używany głównie do przygotowywania dokumentów XML do druku, poprzez konwersję do formatu PDF lub PostScript. Na potrzeby publikacji dokumentów XML w Internecie wykorzystywana jest transformacja XSLT do HTML. XSL-FO jest zdecydowanie mniej popularny niż XSLT. Specyfikacja XSL-FO powstała znacznie później niż XSLT i wolniej też pojawiają się narzędzia obsługujące XSL-FO. W związku z małą popularnością XSL-FO pojawiają się nawet pytania, czy ten język jest w ogóle potrzebny, skoro do przygotowywania dokumentów tekstowych do druku w oparciu o format tekstowy (choć nie XML) są inne, od lat doskonalone, narzędzia jak np. TeX.

Narzędzi do przetwarzania XSL-FO jest zaledwie kilka, dwa najpopularniejsze to:

XEP (RenderX) – narzędzie komercyjne (http://www.renderx.com/);

FOP (Apache) – narzędzie open source (http://xmlgraphics.apache.org/fop/).

XSL-FO – Przykład (1/2)

xsl-fo.fo

```
<?xml version="1.0" encoding="iso-8859-1"?>

<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <fo:simple-page-master master-name="slowniczek">
      <fo:region-body margin="2in"/>
    </fo:simple-page-master>
  </fo:layout-master-set>

  <fo:page-sequence master-reference="slowniczek">
    <fo:flow flow-name="xsl-region-body">
      <fo:block font-family="Helvetica" font-size="16pt"
                font-weight="bold" color="blue"
                text-align="left" space-after="8pt">
        XSL-FO
      </fo:block>
...
```

1
2
3
4
5

Slajd przedstawia przykład dokumentu XSL-FO. Jest to prosty dokument zawierający nagłówek i jeden krótki akapit tekstu. Ze względu na rozmiar dokumentu, jego zakończenie zostanie przedstawione i omówione na kolejnym slajdzie.

Znaczenie poszczególnych fragmentów kodu jest następujące:

1. Element główny dokumentu z deklaracją przestrzeni nazw XSL Formatting Objects, wiążącą ją z prefiksem „fo”.

2. Element <fo:layout-master-set> zawiera definicje układów stron wykorzystywanych w dokumencie. W tym wypadku zdefiniowany elementem <fo:simple-page-master> jest tylko jeden układ strony o nazwie „slowniczek”. Jest to prosty układ strony, zawierający tylko jeden region – ciało (element <fo:region-body>). Region ten ze wszystkich stron ma być otoczony marginesem o szerokości 2 cali.

3. Sekwencja stron (<fo:page-sequence>), dla których ma obowiązywać zdefiniowany wyżej układ o nazwie „slowniczek”.

4. Element <fo:flow> stanowiący kontener dla tekstu w dokumencie. W tym wypadku wskazuje, że tekst ma znaleźć się w regionie ciała.

5. Pierwszy blok tekstu dokumentu (element <fo:block>), zawierający w tym wypadku nagłówek z hasłem. Tekst ma być przedstawiony czcionką Helvetica o rozmiarze 16 punktów, pogrubioną, w kolorze niebieskim. Tekst ma być wyrównany do lewej, odstęp po nim ma wynosić 8 punktów.

(ciąg dalszy dokumentu na następnym slajdzie)

XSL-FO – Przykład (2/2)

xsl-fo.fo

6

```
...
      <fo:block font-family="Times" font-size="12pt"
                font-weight="normal" color="black"
                text-align="justify">
  XSL Formatting Objects (XSL-FO) is an XML vocabulary for
  specifying formatting semantics in XML documents.
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</fo:root>
```

```
fop xsl-fo.fo xsl-fo.pdf
```

xsl-fo.pdf

XSL-FO

XSL Formatting Objects (XSL-FO) is an XML vocabulary for specifying formatting semantics in XML documents.

Slajd przedstawia dokończenie treści dokumentu XSL-FO z poprzedniego slajdu oraz ilustruje sposób jego przetworzenia do formatu odpowiedniego do druku.

Znaczenie wyróżnionego fragmentu dokumentu jest następujące:

6. Drugi blok tekstu dokumentu (element <fo:block>), zawierający w tym wypadku akapit tekstu z objaśnieniem hasła. Tekst ma być przedstawiony czcionką Times o rozmiarze 12 punktów, normalnej grubości, w kolorze czarnym. Tekst ma być wyrównany do lewej i do prawej.

Przykładowy dokument został zawarty w pliku o nazwie xsl-fo.fo. Przygotowanie go do druku, polegające na konwersji do formatu PDF, zostało zrealizowane za pomocą narzędzia FOP. FOP jest uruchamiany w trybie tekstowym z linii poleceń systemu operacyjnego. Pierwszym parametrem jego wywołania jest nazwa źródłowego pliku XSL-FO, a drugim żądana nazwa wynikowego pliku PDF. Efekt transformacji przykładowego dokumentu XSL-FO do postaci PDF został przedstawiony u dołu slajdu.

Podsumowanie

- CSS jest preferowanym mechanizmem formatowania dokumentów HTML
- XML jest tekstowym formatem ogólnego przeznaczenia
  - format publikacji danych w Internecie
  - format wymiany i składowania danych
- XHTML to wersja HTML dostosowana do reguł XML
- XSL jest językiem do formatowania dokumentów XML
  - XSLT – transformacja dokumentu
  - XSL-FO – słownik znaczników opisujących formatowanie

Język CSS jest preferowanym mechanizmem formatowania dokumentów HTML. Język HTML powinien być wykorzystywany do opisu struktury dokumentu, a wszelki opis sposobu prezentacji dokumentu powinien być realizowany poprzez arkusze stylów CSS.

XML jest tekstowym formatem ogólnego przeznaczenia, opartym o znaczniki opisujące zawartość dokumentu. XML jest najczęściej wykorzystywany jako format publikacji danych w Internecie oraz format wymiany i składowania danych.

XHTML to wersja HTML dostosowana do reguł XML. Dokumenty XHTML są poprawnymi dokumentami XML, dzięki czemu są łatwiejsze w przetwarzaniu.

XSL jest językiem do formatowania dokumentów XML. XSL obejmuje trzy części: XSLT, XSL-FO i pomocniczą – XPath. XSLT służy do opisu automatycznej transformacji dokumentu XML do dokumentu o innej strukturze. XSL-FO to słownik znaczników opisujących formatowanie. XSL-FO funkcjonalnie przypomina CSS, ale jest bardziej zorientowany na przygotowywanie dokumentów do druku.

Materiały dodatkowe

- Cascading Style Sheets Home Page, http://www.w3.org/Style/CSS/
- Extensible Markup Language (XML), http://www.w3.org/XML/
- XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition), http://www.w3.org/TR/xhtml1/
- The Extensible Stylesheet Language Family (XSL), http://www.w3.org/Style/XSL/

•Cascading Style Sheets Home Page, http://www.w3.org/Style/CSS/

•Extensible Markup Language (XML), http://www.w3.org/XML/

•XHTML™ 1.0 The Extensible HyperText Markup Language (Second Edition), http://www.w3.org/TR/xhtml1/

•The Extensible Stylesheet Language Family (XSL), http://www.w3.org/Style/XSL/