Dokumenty elektroniczne

Tomasz Przechlewski

styczeń 2007

Dokument

Document = samodzielna jednostka informacji, przeznaczona do interpretacji przez człowieka. W odróżnieniu od **danych** które są zorientowane na przetwarzanie przez maszynę i z reguły nie są zrozumiałe [baza danych, rekord w bazie danych].
Elementy składowe: znaki graficzne, rysunki, układ graficzny dokumentu, tj. rozmieszczenie treści na medium prezentacyjnym.
Różne media: tradycyjne = książka, monitor komputerowy, wyświetlacz PDA albo telefonu komórkowego.
Znaki graficzne = fonty. Rodzina albo krój, odmiana (**gruba**, *pochyła*, *kursywa*, ***gruba pochyła***, itd...)

Przykład 1: Wybrane kroje (odmiana prosta)

Pod źdźbłem spał żółw śnięty Times New Roman 24pt

Pod ¹d¹bªem spaª »óªw ±ni¦ty Garamond 24pt

Pod źdźbłem spał żółw śnięty Helvetica 24pt

Pod ¹d¹bªem spaª »óªw ±ni¦ty Univers 24pt

Pod źdźbłem spał żółw śnięty Computer Modern 24pt

Przykład 2: Różne odmiany kroju (Computer Modern)

Pod źdźbłem spał żółw śnięty Prosta

Pod źdźbłem spał żółw śnięty Kursywa

Pod źdźbłem spał żółw śnięty Pochyłe

Pod źdźbłem spał żółw śnięty Grube

Pod źdźbłem spał żółw śnięty Gruba poszerzona

Pod źdźbłem spał żółw śnięty G/k poszerzona

Przykład 2: Stopień pisma (Computer Modern)

Pod źdźbłem spał żółw śnięty 5pt/24

Pod źdźbłem spał żółw śnięty 10pt/24

Pod źdźbłem spał żółw śnięty 17pt/24

Grafika: mapa bitowa (obrazki, zdjęcia), obiedniowa (diagramy, mapy)
Fonty są traktowane specjalnie: formaty fontów: type 1, TrueType, OpenType.
Fonty = programy komputerowe, zawierające opisy znaków graficznych oraz dodatkowe informacje o wymiarach znaków, parach kernowych i ligaturach. Hinting. Kodowanie.
Grafika obwiedniowa (wektorowa): SVG, WMF, EMF, CDR. **Grafika rastrowa** (bitmapowa): JPEG, GIF, PNG. **Modele kolorów**: CMYK i RGB.

Układ graficzny

Kolumna tekstu, Paginy górne i dolne. Żywa Pagina. Marginalia. Przypisy i odnośniki.

Akapity, śródtytuły, wyliczenia, wzory matematyczne, tabele, rysunki.
Spisy treści i inne spisy, skorowidz.
Register kolumny, register wiersza.

Formaty dokumentów elektronicznych

Formatowanie i oznakowanie (markup, ew. adiustacja). Oznakowanie logiczne i formatowanie wizualne.
A document is a self-contained [samodzielny] unit of information, intended to be communicated to a human. The data that is meant to be machine processable (fragmented) is not a document (example: database record)
Formaty dokumentów elektronicznych: MS Word, HTML, Wiki, TeX/LaTeX (wszystkie są pozbawione struktury).

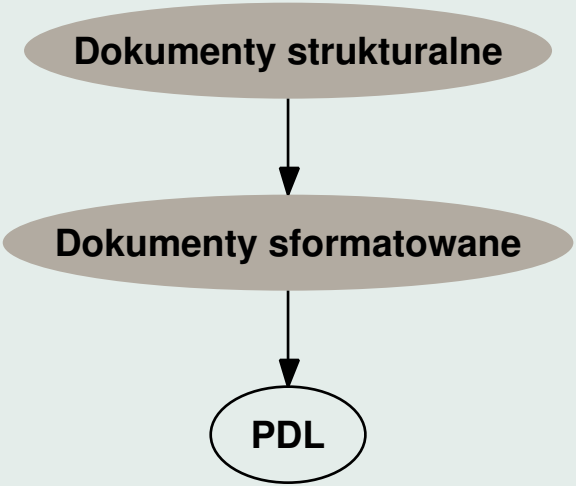
Format wiki

akapit = fragment tekstu oddzielony pustym wierszem

```
''tekst pochylony''
'''tekst wytłuszczony'''
```

==Rozdział==, ===podrozdział===, itd...

```
[[link]], [[link|tekst]], [URL tekst]
[[de:link do strony na wikipedia.de]]
```




```
* element listy nienumerowanej
# element listy numerowanej
[[Grafika:plik.jpg|podpis]]
[[Grafika:plik.jpg|right|thumb|300px|tekst]]
{{szablon}}
```

Format latex

akapit = fragment tekstu oddzielony pustym wierszem
`\footnote{przypis}`
`\ref{odnośnik}`, `\pageref{odnośnik}`, `\label{odnośnik}`.
śródtytuły: `\chapter{tekst}`, `\section{tekst}`, `\subsection{tekst}`.
wyliczenia:
`\begin{itemize} \item treść \item treść ... \end{itemize}`
wzory matematyczne:
`\begin{equation} x^2 + y_2 = z_2^2 \end{equation}`
tabele, rysunki.
Spisy treści i inne spisy, skorowidz: tworzone automatycznie na podstawie zawartości innych elementów.

Format html

W pewnym sensie podobny do latexa [patrz dalej].

SGML/XML

SGML: International Standard (ISO 8879). Information Processing–Text and Office Systems–Standardized Generalized Markup Language.

Znakowanie powinno opisywać strukturę dokumentu (...) a nie określać sposób przetwarzania dokumentu".
XML: Oparty na SGML standard oznakowania dokumentów opracowany przez konsorcjum W3C w 1998 r.

Składniki systemu

1. Deklaracja (*SGML declaration*). Definiuje zestaw znaków wykorzystywany w dokumencie oraz znaki specjalne służące do definiowania i oznaczania znaczników (<, >, &).
2. Definicja dokumentu (DTD – *Document Type Definition*). Definiuje gramatykę (słownik oraz reguły syntaktyczne) formalnego języka, który służy do opisania konkretnego typu dokumentu.
3. Konkretny dokument.

Przykładowy dokument

MEMORANDUM
To: Comrade Napoleon
From: Snowball
In Animal Farm, George Orwell says: "...the pigs had to expend enormous labour every day upon mysterious things called files, reports, minutes and memoranda. These were large sheets of paper

which had to be closely converted with writing, and as soon as they were so converted, they were burnt in the furnace...". Do you think SGML would have helped the pigs?
Comrade Snowball

Przykładowy dokument

```
<?xml version='1.0' encoding='iso-8859-2' ?>
<!DOCTYPE Memo SYSTEM "C:/MYDIR/MEMO.DTD">
<Memo>
<To>Comrade Napoleon</To>
<From>Snowball</From>
<Body>
<P>In Animal Farm, George Orwell says: <Q>...the
pigs had to expend enormous
labour every day upon mysterious things called
files, reports, minutes and memoranda. These were
large sheets of paper which
had to be closely converted with writing, and as
soon as they were so converted, they were
burnt in the furnace...</Q>. Do you
think SGML would have helped the pigs?
</P>
</Body>
```

```
<Close>Comrade Snowball</Close>
</Memo>
```

Przykład DTD

```
<?xml version='1.0' encoding='iso-8859-2' ?>
<!-- DTD for simple office memoranda            -->
<!-- ... -->
<!ELEMENT memo       (to,  from, body, close?)    >
<!ELEMENT to         (#PCDATA)                    >
<!ELEMENT from       (#PCDATA)                    >
<!ELEMENT body       (p*)                         >
<!ELEMENT p          (#PCDATA | q | pref )*       >
<!ELEMENT q          (#PCDATA)                    >
<!ELEMENT pref       EMPTY                        >
<!ELEMENT close      (#PCDATA)                    >
<!-- ... -->
<!--      ELEMENTS NAME   VALUE        DEFAULT   -->
<!ATTLIST Memo STATUS (confiden | public) "public" >
<!ATTLIST p        id     ID           #IMPLIED    >
<!ATTLIST pref     refid  IDREF        #REQUIRED   >
```

DTD

Deklaracja elementu:

```
<!ELEMENT  name   (content-model)  >
```

Content model (model zawartości) składa się z *modelu grup* oraz *wyjątków*. Model grup to lista elementów oraz *operatorów powtórzeń* połączonych *operatorami połączeń*.
Operatory powtórzeń (*occurance indicators*):
? – co najwyżej raz (0, 1);
+ – co najmniej raz (1, 2,...);
∗ – wiele razy lub wcale (0, 1, 2,...).
Operatory połączeń (*connectors*):
`e1,e2` – najpierw e1 potem e2;
`e1|e2` – albo e1 albo e2.
Przykłady:

```
<!ELEMENT  memo   ((to , from), body, close?) >
<!ELEMENT  dl     (dt*,dd?)+  >
```

Atrybuty

Deklaracja atrybutu:

```
<!ATTLIST  element  attrib-name-1 (values) "default"
                    attrib-name-1 (values) "default"
                    ...     >
```

values — może to być lista wartości albo typ atrybutu, gdzie typ atrybutu określamy za pomocą jednej z podanych wartości:
`CDATA` – dane tekstowe (dowolne znaki).
`ID` – *unique identifier*. Identyfikator.
`IDREF` – wartość zdefiniowanego identyfikatora.
`NAME` – nazwa SGML-owa (ciąg znaków `a-zA-Z0-9.-` z których pierwszy jest literą, maksymalna długość ciągu 8 znaków).
`NUMBER` – liczba (ciąg składający się wyłącznie z cyfr).
default – wartość domyślna. Jest to jedna z wartości wyspecyfikowanych w polu *values* albo jedno z predefiniowanych słów kluczowych:
`#REQUIRED` wartość musi zostać określona przez użytkownika.
`#CURRENT` jeżeli wartość nie zostanie podana to przyjmowana jest ostatnio wyspecyfikowana.
`#IMPLIED` jeżeli wartość nie zostanie podana to zostanie nadana automatycznie przez parser.
Przykład:

```
<IMG SRC="~/gifs/sowa.gif" ALT="Sowa uszata"/>
```

Deklarowanie zawartości elementów

`PCDATA` – (*parsed character data*). Dane tego typu są normalnie analizowane przez parser XML-owy.

```
<!ELEMENT  From  (#PCDATA)              >
<!ELEMENT  P     (#PCDATA | Q | Pref)   >
```

`EMPTY` – Element o tym typie jest pusty, tj. nie ma żadnej zawartości (ale może posiadać atrybuty). PRZYKŁADY:

```
<!ELEMENT IMG   EMPTY >
<!ATTLIST IMG SRC %URI;  #REQUIRED
          ALT CDATA #IMPLIED
          ALIGN (top | middle | bottom) #IMPLIED >

<IMG SRC="~/gifs/sowa.gif" ALT="Sowa uszata" />
```

`ANY` – element może zawierać tekst (tj. dane typu `#PCDATA`) lub dowolny element zdefiniowany w DTD. Używanie ANY prowadzi do definiowania dokumentów niestrukturalnych i dlatego ten typ nie powinien być stosowany.

Encje

Encje są odpowiednikiem pojęcia *zmiennej* w językach programowania. Pozwalają *nazwać* porcję danych. Tą porcją danych

może być: kawałek tekstu, kawałek DTD lub plik zewnętrzny zarówno tekstowy jak i zawierający dane nietekstowe.
encje wewnętrzne (*internal entities*)

```
<!ENTITY UG "Uniwersytet Gdański" >
<!ENTITY EP  'Electronic Publishing' >
...
&UG; &EP;
```

encje znakowe (*general character entities*)

```
<!ENTITY amp  CDATA "&#38;" >
...
&amp;
```

encje parametryczne (*parameter entities*)

```
<!ENTITY % heading "H1|H2|H3|H4|H5|H6" >
<!ENTITY % list "UL | OL | DIR | MENU" >
<!ENTITY % tekst "#PCDATA | A| IMG | BR" >
...
<!ELEMENT (%heading)  - - (%tekst;)+ >
```

encje zewnętrzne (*external entities*)

```
<!ENTITY CHP1 SYSTEM "/usr/tomek/chap1.sgml" >
<!ENTITY SU SYSTEM "/standard/sowauszata.gif" NDATA GIF87A>
<!ENTITY iso1 PUBLIC "..." "...">
```

W definicji encji zewnętrznej należy dodać słowo SYSTEM po nazwie encji, a następnie nazwę pliku lub słowo PUBLIC po którym występują dwa napisy. W przypadku encji określających plik z danymi nietekstowymi część NDATA określa typ zawartości.

Deklaracja notation

Określa typ danych w pliku binarnym.

```
<!NOTATION GIF87A SYSTEM "GIF">
<!NOTATION TEX PUBLIC
"+//ISBN 0-201-13448-9::Knuth//NOTATION The TeXbook//EN">
```

Instrukcje formatujące

Processing instructions (PIs) umożliwiają przesłanie dodatkowej informacji formatującej do aplikacji. PI mają ogólną postać `<?nazwa treść?>`. Nazwa określa aplikację, aplikacja powinna przetwarzać znaną jej instrukcję przetwarzającą oraz ignorować pozostałe.

W standardzie XML nazwy PI names rozpoczynające się od xml są zarezerwowane.

Pozostałe konstrukcje: CDATA Sections, komentarze

Sekcje CDATA

```
<![CDATA[ .... ]]>
```

Komentarze

```
<!-- ...  -->
```

Cechy dokumentów XML

Well-formed – dokument poprawnie sformatowany (*parser niewalidujący*)
Valid – dokument poprawny składniowo, tj. posiadający deklarację `DOCTYPE` i zgodny z tą deklaracją (*parser walidujący*)
Poprawność sformułowania: dokument musi zaczynać się od instrukcji przetwarzającej (sterującej)

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

Wszystkie elementy muszą posiadać znacznik otwierający i znacznik zamykający; elementy o zawartości pustej muszą być oznaczone jako `<element/>`
Istnieje tylko jeden element, wewnątrz którego jest zawarty cały dokument
Wartości atrybutów są umieszczone wewnątrz cudzysłowów i muszą być unikalne dla każdego elementu (atrybut o tej samej wartości może być podany tylko raz)

Przestrzenie nazw (*Namespaces in XML*)

Różne DTD mogą stosować te same nazwy dla różnych celów.
Rozwiązanie: nazwy elementów/atrybutów kawlifikowane za pomocą prefiksów URI.
Przykładowo, dokument opisujący asortyment w sklepie AGD może używać elementu widelec z określonym URI, a dokument opisujący części rowerowe z innym:

```
<{http://www.agd.com/asortyment}widelec />
<{http://www.rowery.com/xml}widelec />
```

Ponieważ parsery XML zgodne z wersją 1.0 specyfikacji nie byłyby zdolne do walidacji ww. elementów; dodanie przestrzeni nazw jest w sposób pośredni.

Jeżeli nazwa dokumentu zawiera dwukropek, to część przed dwukropkiem jest traktowana jako prefiks określający przestrzeń nazw a część po jako nazwa lokalna:

```
<agd:widelec xmlns:agd="http://www.agd.com/asortyment" />
```

W analogiczny sposób można zdefiniować kwalifikowane nazwy atrybutów:

```
<widelec rowery:rodzaj="karbon"
  xmlns:rowery="http://www.rowery.com/xml">Time</widelec>
```

Jeżeli znacznik zawiera prefiks, ale nie zawiera atrybutu xmlns, to za wartość określającą przestrzeń nazw przyjmowana jest wartość xmlns określona dla pierwszego elementu-przodka w hierachii dokumentu. Często większość elementów w dokumencie posiada nazwy z pewnej przestrzeni nazw, a tylko niewielka część pochodzi spoza niej. Atrybut xmlns określa URI związane ze wszystkimi elementami nie poprzedzonymi żadnym prefiksem:

```
<rower xmlns='urn:com:rowery'>
<widelec typ="alu">
<fajka dl="13">
</rowery>

<{urn:com:rowery}:rower >
```

```
<{urn:com:rowery}widelec typ="alu">
<{urn:com:rowery}fajka dl="13">
</rowery>
```

Zastosowania

– parsery: rxp, IE, Mozilla; Edytory: OpenOffice, Emacs, XMLmind editor;
– CALS (Computer-aided Acquisition and Logistic Support);
– HTML (`http://www.w3.org/TR/html`);
– XML (`http://www.w3.org/TR/REC-xml`);
– Text Encoding Initiative;
– Docbook (`http://www.docbook.org`).

Docbook

Deklaracja `DOCTYPE`:

```
<?xml version="1.0" encoding="iso-8859-2" standalone="no"?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook XML V4.1.2//EN"
     "http://www.oasis-open.org/docbook/xml/4.1.2/docbookx.dtd">
```

Formal public identifier (FIP):
`prefix//owner-id//text-class description//lang//display-version`

gdzie: *prefix* to + lub - w zależności od tego czy dokument jest zarejestrowany czy nie; *doc-class* – klasa dokumentu, typowe określenia to: DOCUMENT, DTD, ENTITIES; *description* – opis; *lang* – język dokumentu; *display-version* – rzadko używany.
Ogólna struktura dokumentu typu book:
tytulatura: `<title>`;
metainformacje: `<bookinfo>`;
dedykacja: `<dedication>`,
spisy: `<toc>`, `<lot>`, `<index>`, `<indexdiv>`;
podział na części: `<preface>`, `<chapter>`, `<sect1>`...`<sect5>`, `<appendix>`, `<glossary>`, `<glossdiv>`.
Elementy typu *block*:
listy (`<itemizedlist>`, `<orderedlist>`, `<variablelist>` + 4 inne); ostrzeżenia/uwagi/notki (`<caution>`, `<important>`, `<note>`, `<tip>`, `<warning>`); akapity (`<para>` + 2 inne); rysunki i tabele, równania wyeksponowane (`<equation>`); elementy zachowujące odstępy/podział na wiersze zawartości (`<literallayout>`, `<programlisting>`, `<screen>`, itp).
elementy typu *inline*:
`<emphasis>`, `<footnote>`, `<quote>`; odsyłacze (`<link>`, `<ulink>`, `<xref>`); *markup* (`<literal>`, `<markup>`, `<prompt>`, `<replaceable>`, `<userinput>`, `<sgmltag>`; matematyka (`<subscript>`, `<superscript>`, `<inlineequation>`); interfejsy (`<guibutton>`, `<guimenu>`,

<guisubmenu>, itp...); j. programowania (<classname>, <constant>, <function>, itp...) inne...
Szkielet dokumentu typu book:

```
<?xml version="1.0" encoding="iso-8859-2" standalone="no"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.1.2//EN"
     "http://www.oasis-open.org/docbook/xml/4.1.2/docbookx.dtd">
<book>
<bookinfo>
  <title>My First Book</title>
  <author>
    <firstname>Jane</firstname>
    <surname>Doe</surname>
   </author>
  <copyright><year>1998</year>
   <holder>Jane Doe</holder>
  </copyright>
</bookinfo>
<preface><title>Foreword</title> ... </preface>
<chapter> ... </chapter>
<chapter> ... </chapter>
<chapter> ... </chapter>
<appendix> ... </appendix>
```

```
<appendix> ... </appendix>
<index> ... </index>
</book>
```

Szkielet dokumentu typu article

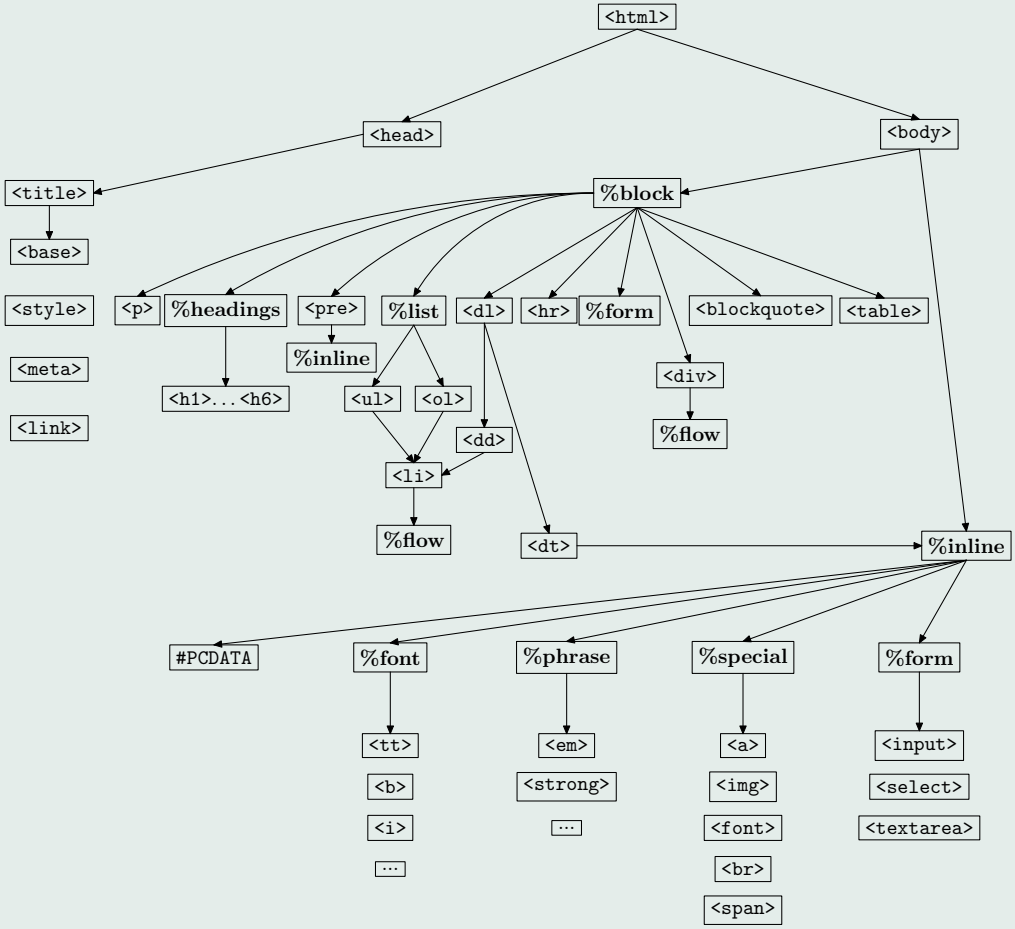
```
<?xml version="1.0" encoding="iso-8859-2" standalone="no"?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook XML V4.1.2//EN"
     "http://www.oasis-open.org/docbook/xml/4.1.2/docbookx.dtd">
<article>
<artheader>
  <title>My Article</title>
  <author><honorific>Dr</honorific>
    <firstname>Emilio</firstname>
    <surname>Lizardo</surname></author>
</artheader>
<para> ... </para>
<sect1><title>On the Possibility of Going Home</title>
<para> ... </para>
</sect1>
<bibliography> ... </bibliography>
</article>
@\mitabrev
```

HTML

HTML (HyperText Markup Language, hipertekstowy język znaczników): [nietypowa] aplikacja SGMLa będąca podstawowym formatem dokumentów publikowanych w sieci WWW.
Ostatnią wersją HTMLa jest wersja 4.01, która próbuje wydzielić zarządzanie wyglądem strony do kaskadowych arkuszy stylów (CSS).
Element główny każdego dokumentu HTML jest html. Element główny zawiera dwa kolejne elementy: head (nagłówek dokumentu) i body (treść dokumentu). W3C zaprzestało rozwoju HTML i promuje oparty na XML standard XHTML.
Deklaracja typu dokumentu:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
```

Liczba elementów/atrybutów:


<html>
<head>
<body>
<title>
%block
<base>
<style>
<p>
%headings
<pre>
%list
<dl>
<hr>
%form
<blockquote>
<table>
<meta>
%inline

<link>
<h1>...<h6>

%flow
<dd>

%flow
<dt>
%inline
#PCDATA
%font
%phrase
%special
%form
<tt>

<a>
<input>
<b>

<img>
<select>
<i>
...
<font>
<textarea>

Nazwa DTD	rok	elementy	atrybuty
HTML 2.0	95	49	32
HTML 3.2	96	69	x
HTML 4.0.1/Transitional	99	89	x
HTML 4.0.1/Strict	99	77	92
Docbook 4.0	?	375	100

akapit = <P>
przypis = ??
odnośnik = a< href="URL", <ele id="ID">
śródtytuły: <h1>, ... <h7>.
wyliczenia: ..., ...,
<dl><dt>...</dt><dd>...</dd> ... </dl>
wzory matematyczne: ??
tabele: <table><tr><td> ... </table>
rysunki: <img src="URL">
spisy treści i inne spisy: tworzone przez aplikację
paginy: ??
elementy div oraz span

HTML HEAD

```
<!ENTITY % version "version CDATA #FIXED '%HTML.Version;'">
```

```
<!ENTITY % html.content "HEAD, (FRAMESET|BODY)">
<!ELEMENT HTML  O O (%html.content) >
<!ATTLIST HTML
  %version;
  %i18n;    -- lang, dir --
  >

<!-- +-----------------------------------------------+  -->

<!ENTITY % i18n
  "lang NAME     #IMPLIED
   dir (ltr|rtl) # IMPLIED ">

<!-- +-----------------------------------------------+  -->

<!ELEMENT HEAD O O (%head.content) +(head.misc) >
<!ENTITY %head.content "TITLE &ISINDEX? &BASE?">
<!ENTITY %head.misc "SCRIPT|STYLE|META|LINK">
<!ATTLIST HEAD
  %i18n;    -- lang, dir --
  profile   %url   #IMPLIED
  >
```

Przykład:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
      "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd" >
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<link rel="meta" href="tp.rdf" /> <!-- metaifo w pliku zew. -->
<meta http-equiv="content-type" content="text/html; charset=iso-8859-2"
<meta name="ICBM" content="54.44005, 18.55019" />
<meta name="DC.title" content="Tomasz Przechlewski's home page" />

<link rel="stylesheet" type="text/css" href="./style/tp-base.css" title
<link rel="alternate stylesheet" type="text/css" href="./style/tp-bw.cs
<link rel="alternate stylesheet" type="text/css" href="./style/tp-big.c
<script type="text/javascript" src="./skrypty/script/sss.js"></script>

<title>Home page of Tomasz Przechlewski </title>

</head>
```

HTML BODY

```
<!--==================== Generic Attributes ============================
<!ENTITY % coreattrs
```

Strona główna
Strona tytułowa
◀◀ ▶▶
◀ ▶
Strona 27 z 44
Powrót
Full Screen
Zamknij
Koniec

```
 "id          ID              #IMPLIED  -- document-wide unique id --
  class       CDATA           #IMPLIED  -- space-separated list of class
  style       %StyleSheet;    #IMPLIED  -- associated style info --
  title       %Text;          #IMPLIED  -- advisory title --"
  >

<!ENTITY % i18n
 "lang        %LanguageCode; #IMPLIED  -- language code --
  dir         (ltr|rtl)      #IMPLIED  -- direction for weak/neutral te
  >

<!ENTITY % events
 "onclick     %Script;       #IMPLIED  -- a pointer button was clicked
  ondblclick  %Script;       #IMPLIED  -- a pointer button was double c
  ...
  >

<!ENTITY % attrs "%coreattrs; %i18n; %events;">


<!-------------- Text Markup ---------------------->
<!ENTITY % heading "H1|H2|H3|H4|H5|H6">
<!ENTITY % list "UL | OL">
```

```
<!ENTITY % fontstyle "TT | I | B | BIG | SMALL">
<!ENTITY % phrase "EM | STRONG | DFN | CODE |
          SAMP | KBD | VAR | CITE | ABBR | ACRONYM" >

<!ENTITY % special "A | IMG | OBJECT | BR | SCRIPT
     | MAP | Q | SUB | SUP | SPAN | BDO">

<!ENTITY % formctrl "INPUT | SELECT | TEXTAREA | LABEL | BUTTON">

<!-- %inline; covers inline or "text-level" elements -->
<!ENTITY % inline "#PCDATA | %fontstyle;
   | %phrase; | %special; | %formctrl;">

<!ELEMENT (%fontstyle;|%phrase;) - - (%inline;)*>
<!ATTLIST (%fontstyle;|%phrase;)
  %attrs;  -- %coreattrs, %i18n, %events --
  >

<!ELEMENT SPAN - - (%inline;)* -- generic language/style container -->
<!ATTLIST SPAN %attrs;  -- %coreattrs, %i18n, %events --
  %reserved; -- reserved for possible future use --
  >
```

```
<!--================== HTML content models ==========-->
<!--
    HTML has two basic content models:
    %inline; character level elements and text strings
    %block;  block-like elements e.g. paragraphs and lists
-->

<!ENTITY % block
     "P | %heading; | %list; | PRE | DL | DIV | NOSCRIPT |
      BLOCKQUOTE | FORM | HR | TABLE | FIELDSET | ADDRESS">

<!ENTITY % flow "%block; | %inline;">

<!--============= Document Body =====================-->

<!ELEMENT BODY O O (%block;|SCRIPT)+ +(INS|DEL)
   -- document body 4.0.1/strict-->

<!ELEMENT BODY O O (%flow;)* +(INS|DEL)
  -- document body 4.0.1/transitional-->

<!ATTLIST BODY %attrs;  -- %coreattrs, %i18n, %events --
  onload    %Script;   #IMPLIED  -- the document has been loaded --
```

```
  onunload  %Script;   #IMPLIED  -- the document has been removed --
  >

<!ELEMENT DIV - - (%flow;)*  -- generic language/style container -->
<!ATTLIST DIV %attrs;  -- %coreattrs, %i18n, %events --
  %reserved;   -- reserved for possible future use -- >
```

CSS

Prosty, blokowo zorientowany model dokumentu. Każdy element to blok (prostokąt), posiadający marginesy (*margins*), ramkę (*border*), marginesy wewnętrzne (*padding*) i zawartość (*content*).
Z punktu widzenia formatowania elementy dzielą się na *inline* (wiersze) i *block-level* (akapity).
CSS1 zawiera 5 podstawowych rodzajów formatowania (*properties*): kolor tekstu i tła; rodzaj pisma; odstępy między wierszami i wyrazami; bloki (marginesy, ramki itp); określenie kategorii elementu (*block-level*, *inline*).
Styl (szablon) to podpowiedź dla programu interpretującego dokument HTML; wynik jest zależny m.in. od możliwości tegoż programu (UA).
Deklaracja szablonu:

```
<LINK REL="STYLESHEET" TYPE="text/css" HREF="http://...">
```

```
<?xml-stylesheet type="text/css" href="http://..." ?>
```

Określenie formatowania poszczególnych elementów:

```
selektor {właściwość: wartość}
```

np. `H1 {color: pink}`

```
ele1, ele2, ele3 {właściwość: wartość; właściwość: wartość...}
```

rodzaje selektorów:
`*` każdy element
`E` element typu E
`E F` element F potomek elementu E
`E > F` element F bezpośredni potomek elementu E
`E:first-child` element F pierwszy bezpośredni potomek elementu E
`E + F` element F bezpośrednio poprzedzony E np. `h1 h2`
`E[A]` element E z atrybutem A
`E[A='w']` element E z atrybutem A o wartości w; np. `*[lang="pl"]` albo `[lang="pl"]`
`E[A ~='w']` element E z atrybutem A zawierającym w (tj. A jest listą wyrazów jeden z nich jest równy 'w'

```
DIV[class ~="ważne"] DIV.ważne   *.ważne   .ważne
```

`E#id` E z atrybutem ID równym id
pseudo-elementy/pseudo-klasy:
`E:first-child`

```
E:first-line

E:before

P.special:before { content: "Uwaga!" }

E:after
```

Typ medium prezentacyjnego:

```
@import url("a.css") aural;
@media print { ... }
```

W przypadku pliku html można użyć elementu `LINK`:

```
<LINK REL="STYLESHEET" media="print,handheld" TYPE="text/css" HREF="htt
```

all, aural, braille, handheld, print, screen.
Tekst generowany właściwość `content`:

```
content : string | uri | counter | attr(X)
IMG:before { content : attr(alt) }
```

właściwość page:
Umożliwia sformatowanie dokumentu z podziałem na strony, tj. prostokąty o określonej wielkości:

```
@page { size : 8.5in 11in; margin: 1in }
@page : left { }
@page : right { } // także : first
```

```
page-break-after
page-break-before
orphans
windows
```

Inne właściwości:
Kolory: `color`, `background`
Kroje pisma:

```
font-family: Garamond | serif | sans-serif | monospace ...
font-style: normal | italic | oblique,
font-variant: normal | small-caps,
font-size: larger | smaller | 12pt | 150% | 1.5em
font-weight: normal | bold | bolder | lighter | 100..900
```

```
<DIV CLASS="STRONG">Uwaga:</DIV>
DIV.STRONG { font-weight: bolder }
```

Tekst:
Dodatkowy odstęp między wyrazami:

```
word-spacing: normal | 1pt | .3em
```

Dodatk. odstęp między znakami:

```
letter-spacing: normal | 1pt | .1em
```

```
text-decoration: none | undeline | overline | line-through | blink
vertical-align: baseline | sub | super | top
text-transform: capitalize | uppercase | lowercase | none
text-align: left | right | center | justify
text-indent: none | 20pt
line-height: normal | 1.2 | 1.2em | 120%
```

Bloki:
Określenie marginesów wewnętrznych oraz zewnętrznych, ramki, wielkości (obrazki)
Kategorie:

```
display: block | inline | list-item | none
IMG { display: none }
```

XSL= XPath + XSLT + FO

XPath

Język programowania [o specyficznej składni] służący do adresowania elementów dokumentu XML:
typy zmiennych (*object types*): boolean, number, string, node-set (4);
typy węzłów (*node types*): root, element, atrribute, namespace, pi, comment, text (7);
ścieżka dostępu (*location path*): [/] `krok` [/`krok`]
krok (*location step*): `oś::test[predykat]`;
oś (*axis*, 13): `ancestor::` `ancestor-or-self::` `attribute::` `child::` `parent::` `self::` `descendant::` `descendant-or-self::` `following::` `following-sibling::` `namespace::` `preceding::` `preceding-sibling::`
test (*predicate*, 9): `nazwa` `prefiks:nazwa` `*` `prefiks:*` `node()` `text()` `comment()` `processing-instruction()` `processing-instruction(napis);`
składnia uproszczona: 'nic' – `child::` `@` = attribute::
`//` = `/descendant-or-self::node()/` `.` = `self::node()`
`..` = `parent::node()` `/` = root ;
predykat = [wyrażenie]
zmienna = `$qname`
operatory:

– nawiasy do grupowania;
– zbiory-węzłów: |
– logiczne: `<=, <, >=, >, =, =! and or`;
– numeryczne: `*, div, mod, +, -`;
– funkcje;

Przykłady

`para` = elementy `<para>`, dzieci w. bieżącego; `*` = wszystkie elementy dzieci w. bieżącego; `text()` = wszystkie w. tekstowe dzieci w. bieżącego; `@name` = atrybut name, dziecko w. bieżącego; `@*` = wszystkie atrybuty dzieci w. bieżącego; `para[1]` = pierwszy węzeł `para`; `para[last()]` = ostatni węzeł `para`;

Zadania

`/article/chapter[4]/section[2]` = ? ; `chapter//para` = ?; `//para` = ?; `../@lang` = ?; `para[@type='warning'][5]` = ?;
`para[5][@type='warning']` = ?;

XSLT

Język programowania [o specyficznej składni] służący do transformowania dokumentów XML:

```
<?xml version="1.0" encoding="iso-8859-2" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/199/XSL/Transform"
     version="1.0">

<xsl:import href="tpext.xsl" /> <!-- niższy priorytet -->
<xsl:iinclude href="tpdef.xsl" />

<!-- top level elements: -->
<xsl:output
    method = { "xml" | "html" | "text" | "QName" }
    encoding = "iso-8859-2"
    doctype-public="-//W3C//DTD HTML 4.01 Transitional//EN"
    doctype-system="..."
    indent={"yes" | "no"}
    saxon:character-representation="native:decimal"
    xmlns:saxon="http://icl.com/saxon" />

<xsl:template
   name="QName" match="Pattern" mode="QName" priority="Number">
 <xsl:param>*
```

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

<xsl:variable name="QName" select="Expression"> ... </xsl:variable>

<xsl:param name="QName" select="Expression"> ... </xsl:param>

<xsl:key name="QName" match="Pattern" use="Expression" />

<!-- XSLT instructions: -->
<xsl:apply-templates select="Expression mode="QName" >
   ( <xsl:with-param> | <xsl:sort> ) *
</xsl:apply-templates>

<xsl:call-template name="QName" >
   <xsl:with-param> *
</xsl:call-template>

<xsl:with-param name="QName" select="Expression"> ... </xsl:with-param>

<xsl:vale-of select="expression" />

<xsl:if test="expression> ... </xsl:if>
```

```
<xsl:choose>
  <xsl:when>+
  <xsl:otherwise>?
</xsl:choose>

<xsl:for-each select="Expression">
  <xsl:sort>*
</xsl:for-each>

<xsl:sort select="Expression" order= {"ascending" | "descending" }
   data-type={ "text" | "number" | "QName" } />

<xsl:attribute name="QName">...</xsl:attribute>

<xsl:comment> ... </xsl:comment>

<xsl:element name="QName"> ... </xsl:element>

<xsl:text> ... </xsl:text>

<!-- Funkcje : -->
<!-- konwersja: -->
```

```
 boolean(val) => boolean ; number(val) => number ; string(val) => napis

<!-- arytmetyczne : -->
 ceiling, floor, round

<!-- napisowe: -->
 concat(v1, v2, ... ) => napis
 contains(val, substring) => boolean
 start-with("Pinarello", "Pi") => true
 string-length() => number
 substring("Pinarello Dogma", 11) => "ello Dogma"
 substring("Pinarello Dogma", 6, 3) => "ell"
 substring-after("print=yes", "=") => "yes"
 substring-before("print=yes", "=") => "print"
 translate("XYZ-12:01", "-:", "!&") => "XYZ!12&01"

<!-- agregacja: -->
 count(zbiór-węzłów) => liczba
 sum(zbiór-węzłów) => liczba

<!-- kontekst: -->
 last() => number
 position() => number
```

```
<!-- inne : -->
 generate-id() => napis
 id()
 key()
```

Przykład

```
<?xml version='1.0' encoding='iso-8859-2' ?>

<zestawienie rok="2003">
 <czesc id="m1201" typ="rama">
  <nazwa>GX2 Carbon</nazwa>
  <firma>Merckx</firma>
  <cena>3300</cena>
  <sprzedaz>2</sprzedaz>
 </czesc>
 <czesc id="m2345" typ="rama">
  <nazwa>Team SC</nazwa>
  <firma>Merckx</firma>
  <cena>2150</cena>
  <sprzedaz>2</sprzedaz>
 </czesc>
  ...
```

```
</zestawienie>

<?xml version="1.0" encoding="iso-8859-2"?>
<!-- znajduje ramy drozsze od 2000 $, wypisuje w porzadku
   od najdroższej do najtańszej  -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

  <xsl:output method="text" encoding="iso-8859-2" />

    <xsl:template match="zestawienie">
      <xsl:for-each select="//czesc[@typ='rama'][./cena &gt; 2000]">
        <xsl:sort select="./cena" data-type="number" order='descending
           <xsl:value-of select="./nazwa"/>
           <xsl:text> </xsl:text>
           <xsl:value-of select="./firma"/>
           <xsl:text> : </xsl:text>
           <xsl:value-of select="./cena"/>
           <xsl:text>&#10;</xsl:text>
    </xsl:for-each>
  </xsl:template>

</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="iso-8859-2"?>
<!-- Ile sprzedano ram firmy $firma -->
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

  <xsl:output method="text" encoding="iso-8859-2" />

  <xsl:param name="firma" select="Colnago"/> <!-- źle -->

    <xsl:variable name="obrot"
      select="sum(//czesc[@typ='rama'][./firma=$firma]/sprzedaz)" />

      <xsl:template match='/'>
        <xsl:text>Sprzedano: </xsl:text>
        <xsl:value-of select="$obrot" />
        <xsl:text> ram firmy: </xsl:text>
        <xsl:value-of select="$firma" />
        <xsl:text>&#10;</xsl:text>
      </xsl:template>

</xsl:stylesheet>
```

FO

Standard określający sposób formatowania dokumentu XML (układ elementów na stronie, kolorów, fonty itd.), z uwzględnieniem ekranu komputera oraz wydruku „na papierze”. Pełni tę samą funkcję jak standard CSS, ale ma większe możliwości.