23.05.2017 KOLOKWIUM

* istota XML
* poprawność składniowa dokumentu
* poprawność strukturalna dokumentu
* narzędzia zapisywania struktury dokumentu
* przestrzenie nazwowe
* aplikacje:
* MathML
* XForms
* XLink
* Xinclude
* XSLT

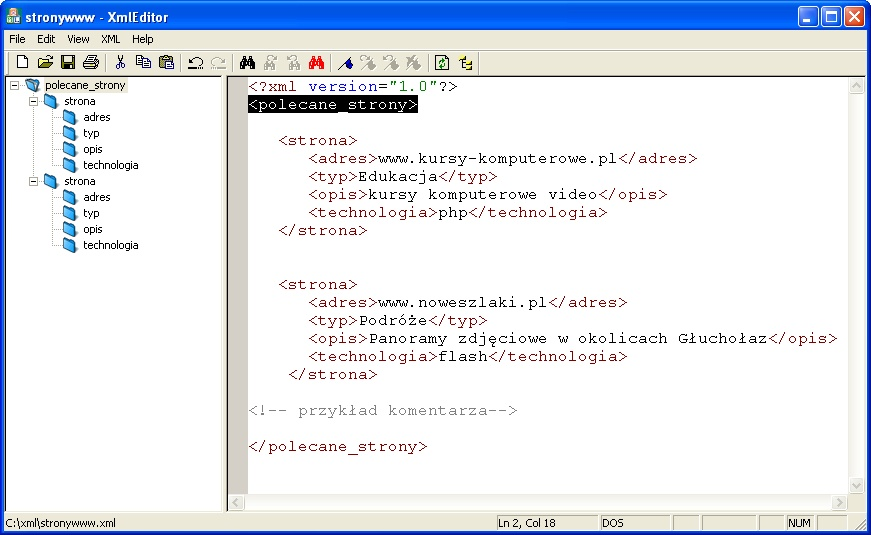
**Istota XML**

* XML (skrót od eXtensible Markup Language - rozszerzalny język znaczników) to otwarty standard opracowany przez W3C.
* XML nie jest kolejnym językiem do przechowywania konkretnych danych, jak np. język HTML opisujący wygląd stron sieciowych.
* XML to język opisujący dane (metajęzyk). Służy do tworzenia innych języków (aplikacji XML) służących do przechowywania informacji. Jeśli masz potrzebę zapisywania określonych danych o określonej strukturze, XML okaże się najlepszym narzędziem, bez względu jakie by te dane nie były.
* W przeciwieństwie do np. HTML, XML nie ma ograniczonej liczby znaczników, bo pozwala przechowywać dowolne dane i to w jak najbardziej wygodny dla nas sposób, bo sami go określamy. Sami określamy strukturę danych.

Znaczniki XML wyglądają podobnie do znaczników HTML. Pomiędzy znacznikami znajduje się nazwa znacznika, która może być dowolnie definiowana. Powinna ona odnosić się i opisywać zawartość znacznika, a nie sposób jego prezentacji.

Znaczniki XML mogą posiadać atrybuty. Jest to para nazwa-wartość, która jest powiązana ze znacznikiem startowym elementu. Nazwy są oddzielone od wartości znakiem "=". Wartość powinna być otoczona pojedynczym lub podwójnym apostrofem.

**Poprawność składniowa dokumentu**



* Standardowa deklaracja dokumentu xml, zwana **prologiem**.
* **Element główny** dokumentu xml czyli "polecane\_strony".
* Elementy zagnieżdżone.
* Komentarz, który jest pomijany przez parser.
* Białe znaki
* Instrukcja sterująca

[więcej - [załącznik 1.pdf](zalacznik1.pdf)]

**Poprawność strukturalna dokumentu**

Warunkiem koniecznym poprawności strukturalnej jest poprawność składniowa. Poprawność składniowa wymaga spełnienia wzmiankowanych wcześniej warunków, pozwalających oprogramowaniu przetwarzającemu zidentyfikować znaczniki i odtworzyć na ich podstawie hierarchię elementów dokumentu.

Z kolei poprawność strukturalna wymaga określenia **schematu dokumentu**, z którym zawartość dokumentu jest zgodna. Budowa takiego dokumentu obejmuje: **prolog**, który przy stosowaniu DTD zawiera deklarację albo odwołanie do zewnętrznej *definicji typu dokumentu* - DTD (*Document Type Definition*). Warunkiem poprawności strukturalnej jest wówczas zgodność treści dokumentu (nazwy i struktura elementów) z podaną specyfikacją DTD.

Dokument XML, jako dokument poprawny strukturalnie powinien być wyposażony w formalny opis jego części składowych (nie jest to konieczne ale zalecane). Taki opis w dużej mierze pozwala na uzyskanie pełnej kontroli nad powstającymi w oparciu o niego, wypełnionymi treścią egzemplarzami dokumentu. Klasyfikacja dokumentów pod względem typów, ułatwia też ich przetwarzanie.

Obecnie wzorce struktur, opisujące typy dokumentów możemy definiować na dwa sposoby: używając odziedziczonego po SGML mechanizmu określanego jako DTD (*Document Type Definition*) lub nowszego, rozszerzającego możliwości DTD zapisu XML Schema.

[więcej - [załącznik 1.pdf](zalacznik1.pdf)]

**Narzędzia zapisywania struktury dokumentu - DTD i Schema**

**DTD**

Deklaracja elementu DTD:

DTD opisuje formalną strukturę dokumentu, definiuje

* elementy - ich nazwy, miejsce wystąpienia, typy danych elementów, liczbę wystąpień elementów,
* atrybuty - informacje, czy atrybuty są wymagane, jakie są wartości domyślne, w jakich elementach występują

Definicje DTD mogą być umieszczone w samym dokumencie XML (wewnętrzne DTD) lub w osobnym pliku (DTD zewnętrzne)

Składa się ze:

* słowa kluczowego ELEMENT,
* nazwy danego elementu (np wydzial w przykladzie),
* opisu zawartości.

Uwaga: Element może być zadeklarowany tylko raz, nazwa musi zaczynać się od litery, znaku podkreślenia lub dwukropka, dalej może zawierać także cyfry, kropki, myślniki.

<!ELEMENT nazwa (model\_zawartości)>

Opis zawartości:

W ramach opisu zawartości można użyć:

* (#PCDATA) - oznacza dowolną zawartość tekstową,
* EMPTY - oznacza pusty element (pozbawiony treści)
* ANY - element może zawierać dowolne wartości, należy używać rozważnie!

Deklarowanie podelementów:

Do deklarowania podelementów może być zastosowany model

* sekwencyjny - istotna jest kolejność!,
* wyboru,
* połączenie sekwencji i wyboru
* Dopuszczalne wskaźniki liczby wystąpień:
* ? - zero lub jedno wystąpienie,
* £ - dowolna liczba wystąpień,
* + liczba wystąpień większa od zera,

Atrybuty:

Atrybuty służą do umieszczania dodatkowej informacji w elemencie. Sa definiowane w deklaracji listy atrybutów, która jest identyfikowana przez słowo kluczowe **ATTLIST.** Po słowie kluczowym jest nazwa elementu, następnie nazwa atrybutu, typ, wartość domyślna lub określenie konieczności wystąpień.

Wartośc domyślna zostaje automatycznie przyjęta, jeżeli inna wartośc nie jest podana. Pozostałe możliwości:

* #REQUIRED - atrybut jest obowiązkowy, bez określania wartości,
* #IMPLIED - atrybut jest opcjonalny,
* #FIXED ”wartość ” wartość ustalona

Typy atrybutów:

* CDATA - łańcuch znaków,
* NMTOKEN - pojedyńcze słowo, może zawierać symbole takie jak nazwa, nie może zawierać białych znaków (spacji, tabulacji itd),
* NMTOKENS - lista słów pooddzielanych znakami spacji
* ENTITY - jednostka (encja) - np. plik, tekst itd.
* ENTITIES - lista jednostek oddzielonych spacjami,
* ID - identyfikator, do którego można się odwołać przez IDREF w innym atrybucie, dla danego elementu może być tylko jeden atrybut typu ID,
* IDREF - wskaźnik umożliwiający odwołanie do innego elementu, wartośc atrybutu musi odpowiadać wartości ID w innym elemencie danego dokumentu,
* IDREFS - lista odsyłaczy typu IDREF oddzielonych spacjami,
* wyliczeniowy - atrybut przyjmuje jedną z wymienionych wartości, lista jest umieszczona w nawiasach okrągłych, elementy oddzielone znakiem |

**<!ATTLIST kierunek rodzaj**

**(dzienne|podyplomowe|zaoczne|wieczorowe) ”dzienne”>**

atrybut o nazwie rodzaj dla elementu kierunek ma domyślną wartość, poza tym są dopuszczalne 3 inne.

Kiedy atrybuty, kiedy podelementy

* podelementy stosuje się do opisu hierarchii elementów (relacja podrzędności między elementami)
* podelementy stosuje się, gdy informacje można traktować jako odrębny, niezależny obiekt,
* atrybuty warto stosować do opisu informacji dodatkowych,
* z atrybutów warto korzystać, aby opisać zależności między elementami
* dla elementów nie można stosować typu wyliczeniowego, czyli zastosowanie atrybutów jest celowe, aby dopuścić tylko ustalone wartości,
* atrybuty stosujemy także wtedy gdy informacja ma stałą wartość,
* gdy istnieje potrzeba zapewnienia nepowtarzalności wartości stosujemy atrybut typu ID.

Przykład:

<!ELEMENT ARTYKUL (TYTUL, TRESC)>

<!ELEMENT TYTUL (EMPTY)>

<!ELEMENT TRESC (EMPTY)>

<!ELEMENT ARTYKUL (TYTUL, TRESC)>

<!ELEMENT TYTUL (#PCDATA)>

<!ELEMENT TRESC (#PCDATA)>

<!ELEMENT ARTYKUL (TYTUL, TRESC)>

<!ATTLIST ARTYKUL

dataPubl NMTOKEN #REQUIERD

autor ID #REQUIERD>

<!ELEMENT TYTUL (#PCDATA)>

<!ELEMENT TRESC (#PCDATA)>

<!ELEMENT KSIAZKA (AUTOR+, TYTUL, TRESC, OPIS, OPINIE)>

<!ELEMENT AUTOR (IMIE, NAZWISKO, EMAIL?, OPISDZIALALNOSCI?)>

...

<!ELEMENT OPINIE (OPINIA | ODNOSNIK)\*>

* Związanie DTD z egzemplarzem dokumentu wymaga poprzedzenia go deklaracją DOCTYPE. Deklaracja taka zawiera: wskazanie elementu głównego stanowiącego korzeń drzewa struktury oraz

\* właściwą definicję struktury:

<!DOCTYPE nazwaElementuKorzenia [ ...tu definicja ... ]>

\* lub odwołanie do zawierającej ją encji zewnętrznej:

<!DOCTYPE nazwaElementuKorzenia SYSTEM ścieżkaDoPliku.dtd >

**XML Schema**

Do definicji typu dokumentu w formacie XML Schema wykorzystywana jest standardowa składnia XML-a. Dla uniknięcia konfliktów nazw, składniki definicji należą do przestrzeni nazw XML Schema, wyróżnianej prefiksem xsd.

Całą definicja zawarta jest w elemencie głównym xsd:schema, zaś odpowiednikami deklaracji i atrybutów z DTD są elementy xsd:element i xsd:attribute.

Typy danych: proste i złożone

Proste definiują zbiory wartości atomowych (bez wewnętrznej struktury XML). Są nimi wszystkie typy wbudowane (np. liczba, napis, wartość logiczna) oraz typy stworzone na ich bazie. Używa się ich do określania dopuszczalnych wartości atrybutów i zawartości elementów.

Typy złożone określane przez definicje modeli zawartości + atrybuty.

XML Schema (Schemat XML) jest językiem opisu struktury i zawartości dokumentu XML, rozszerzającym możliwości DTD. XML Schema zapewnia: — rozbudowane typy danych, — ponowne użycie struktur danych, — rozbudowane modele zawartości, — samoopis i samowalidację, — składnię XML.

W przeciwieństwie do DTD, Schemat XML jest dokumentem XML

**2. XML Schema (Version 2)**

1. Deklaracja XML Schema

<?xml version=”1.0” encoding=”UTF-8”>

<xsd:schema xmlns:xsd=”http://www.w3c.org/2001/XMLSchema”>

…

</xsd:schema>

1. Typy danych
   1. Proste – typ prosty definuje element zawierający tylko i wyłącznie tekst

Definiowanie typu prostego:

<xsd:simpleType name=”nazwa”/>

Definiowanie typu prostego wraz z orgraniczeniami:

<xsd:simpleType name=”bardzoOgraniczonyTyp”>

<xsd:restriction base=”xsd:string”> (ograniczamy do typu string)

<xsd:pattern value=”JestemBardzoOgraniczony”/>

<xsd:pattern value=”JestemBardziejOgraniczony”/>

</xsd:restriction>

</xsd:simpleType>

Deklarowanie listy:

<xsd:simpleType name="typListowy">

<xsd:list>

<xsd:simpleType name="">

<xsd:restriction base="xsd:string">

<xsd:maxLength value="20"/>

</xsd:restriction>

</xsd:simpleType>

</xsd:list>

</xsd:simpleType>

* 1. Złożone – typ złożony może składać się z tekstu oraz innych elementów

Definiowanie typu złożonego

<xsd:complexType name=”….”>

<rodzaj struktury>

Deklaracja elementów

</rodzaj struktury>

</xsd:complexType>

Dostępne rodzaje struktury:

<xsd:sequence> sekwencja

<xsd:choice> lista wyboru

<xsd:all> wszystkie elementy w dowolnej kolejności

1. Deklaracja elementów

<xsd:element name=”nazwa”/>

Rozbudowany o typ

<xsd:element name=”nazwa” type=”nazwatypu”/>

Inne ograniczenia:

* minOccurs
* maxOccurs
* fixed
* default
* required

Jest możliwe definiowanie typu danych od razu po deklaracji elementu np.:

<xsd:element name=”nazwa”>

<xsd:simpleType>

…

</xsd:simpleType>

</xsd:element>

1. Definiowanie atrybutów w ramach elementu

<xsd:attribute name=”nazwa” use=”rodzaj użycia”/>

Dopuszczalne rodzaje użycia:

* required
* optional
* prohibited

**Przestrzenie nazwowe**

XML Namespaces - przestrzenie nazw

* W obrębie dowolnej przestrzeni nazw każda nazwa musi być niepowtarzalna
* Przestrzeń nazw jest kontekstem, w którym słowo musi być jednoznacznie przypisane do reprezentowanego przez nie pojęcia
* Przestrzenie nazw w XML są zdefiniowane w odrębnej rekomendacji W3C
* Zostały stworzone, aby zapobiec problemom spowodowanym przez powtarzanie się tych samych elementów w różnych językach
* Przestrzenie nazw w XML to przestrzenie nazw, które identyfikowane są przez referencje IRI lub URI
* Identyfikatory przestrzeni nazw XML są identyczne tylko gdy sekwencja znaków jest identyczna
* Przestrzeń nazw XML można zadeklarować przez dodanie do dowolnego elementu atrybutu xmlns (w różnych wariantach), którego wartością będzie jej identyfikator
* Aby zadeklarować główną przestrzeń nazw, powinien być to tylko atrybut xmlns
* Aby zadeklarować inną przestrzeń nazw, powinien być to atrybut xmlns:prefiks
* Przestrzeń nazw XML jest zadeklarowana tylko dla elementu, którego atrybut ją deklaruje oraz dla jego potomków
* Wszystkie elementy, których nazwy nie zostały poprzedzone prefiksem, trafiają do głównej przestrzeni nazw
* Wszystkie atrybuty, których nazwy nie zostały poprzedzone prefiksem, trafiają do przestrzeni nazw używanej przez element do którego należą

Jeśli naprzemiennie wykorzystywanych jest więcej przestrzeni nazw, można zdefiniować dla każdej z nich przedrostek np. :

<root xmlns:xsl="http://www.w3.org/1999/XSL/Transform"

xmlns:svg="http://www.w3.org/2000/svg" >

<xsl:element>

...

</xsl:element>

<svg:element>

...

</svg:element>

</root>

<studenci xmlns='http://example.org/studenci'

xmlns:u='http://example.org/uczelnia'>

<imie>Jan</imie>

<nazwisko>Kowalski</nazwisko>

<telefon>99999999</telefon>

<u:uczelnia>

<u:nazwa>PB</u:nazwa>

<u:telefon>12345678</u:telefon>

</u:uczelnia>

</studenci>

**MathML**

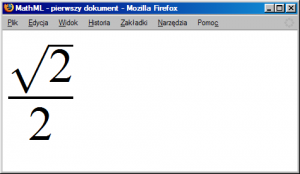
Mathematical Markup Language to język znacznikowy, który umożliwia osadzanie wzorów matematycznych na stronach WWW. Artykuł opisuje, w jaki sposób przygotowywać dokumenty XHTML zawierające wstawki MathML, tak by były zgodne ze standardami oraz poprawnie wyświetlane w przeglądarkach Firefox i IE.

Rozwiązaniem problemu dostępności treści matematycznych jest **Mathematical Markup Language**, czyli matematyczny język znacznikowy, w skrócie oznaczany jako MathML. Formuły zapisane w **MathML-u** mogą być nie tylko wyświetlane w postaci graficznej na monitorze, ale także automatycznie przetwarzane przez różnego rodzaju oprogramowanie. W ten sposób treści matematyczne opublikowane w sieci WWW będzie można katalogować i przeszukiwać podobnie jak dokumenty HTML.

W języku **MathML** wzory matematyczne są tworzone z licznych znaczników, np. mfrac (ułamek), msqrt (pierwiastek kwadratowy) oraz mn (liczba). Wzór “pierwiastek z dwóch przez dwa” przyjmuje w MathML-u postać:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **<math>**  **<mfrac>**  **<msqrt>**  **<mn>**2**</mn>**  **</msqrt>**  **<mn>**2**</mn>**  **</mfrac>**  **</math>** |

Po wyświetleniu w przeglądarce Firefox powyższy kod MathML będzie wyglądał tak, jak na rys. 1.



### Pierwsza strona WWW z wzorami matematycznymi

W jaki sposób przygotować stronę WWW zawierającą MathML? Wzory matematyczne osadzamy w dokumencie XHTML w postaci wstawek math. Specyfikacja **MathML**-a 2.0 (punkt A.2.3) zaleca następujący typ dokumentu:

|  |  |
| --- | --- |
| 1  2  3 | <!DOCTYPE html  PUBLIC \"-//W3C//DTD XHTML 1.1 plus MathML 2.0//EN\"  \"http://www.w3.org/Math/DTD/mathml2/xhtml-math11-f.dtd\"> |

Każda ze wstawek math powinna mieć atrybut xmlns ustalający przestrzeń nazewniczą (por. specyfikacja MathML-a 2.0, punkt 7.1.1):

|  |  |
| --- | --- |
| 1  2  3 | <math xmlns=\"http://www.w3.org/1998/Math/MathML\">  ...  </math> |

Jeśli przygotujemy dokument przyklad-2.xml przedstawiony na listingu 1, po czym otworzymy go w przeglądarce Firefox, ujrzymy stronę taką, jak na rys. 1. Zwróćmy uwagę na nazwę pliku, gdyż po zmianie rozszerzenia z xml na html przykład nie będzie działał.

|  |  |  |  |
| --- | --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | <!DOCTYPE html  PUBLIC \"-//W3C//DTD XHTML 1.1 plus MathML 2.0//EN\"  \"http://www.w3.org/Math/DTD/mathml2/xhtml-math11-f.dtd\">  <html xmlns=\"http://www.w3.org/1999/xhtml\">  <head>  <title>MathML - przykład #2</title>  <meta http-equiv=\"Content-Type\" content=\"text/html; charset=utf-8\" />  </head>  <body>    <p>  Wzó na wysokość trójkąta równobocznego  </p>    <math xmlns=\"http://www.w3.org/1998/Math/MathML\">  <mrow>  <mi>h</mi>  <mo>=</mo>  <mfrac>  <mrow>  <mi>a</mi>  <msqrt>  <mrow>  <mn>3</mn>  </mrow>  </msqrt>  </mrow>  <mrow>2</mrow>  </mfrac>  </mrow>  </math>    </body>  </html> | | |
| http://magazynt3.pl/wp-content/uploads/2008/09/mathml_-_matematyka_na_stronach_www-a8-300x235.png | | http://magazynt3.pl/wp-content/uploads/2008/09/mathml_-_matematyka_na_stronach_www-3b-300x174.png |

**XForms**

XForms to język, będący aplikacją XML (czyli został stworzony za pomocą języka XML). Służy do tworzenia formularzy na stronach WWW i w różnych innych dokumentach (np. PDF). Twórcy XForms stworzyli go z założeniem, że zastąpi on tradycyjne formularze HTML, które mają wiele niedociągnięć.

**Mankamenty tradycyjnych formularzy to chociażby:**

duża zależność formularzy HTML od języków skryptowych,

w celu sprawdzenia poprawności wpisanych danych (tzw. walidacji) używa się JęzykaJavascript,

podobnie jeśli chcemy dokonać jakichś obliczeń na podstawie wartości wpisanych do formularza musimy to zrobić za pomocą skryptów,

JavaScript jest odmiennie interpretowany przez różne przeglądarki - podczas pisania skryptu trzeba mieć to na uwadze,

aby wczytać do formularza dane z zewnęrznego pliku trzeba napisać skrypt działający po stronie serwera (np. języka PHP czy ASP),

problem z dynamicznym tworzeniem kontrolek (okienek i przycisków formularza)

brak wbudowanych mechanizmów obsługi błędów oraz komunikatów dla użytkownika

brak łatwego sposobu zapisywania i przesyłania danych z formularza

Oczywiście to tylko niektóre problemy z jakimi spotykają się programiści interfejsów użytkownika dla sieci Web.

**Zalety zastosowania XForms**

Pierwsze co rzuca się w oczy użytkownikowi formularza XForms to nowe kontrolki - kalendarz, wskaźnik zakresu, output (element wyświetlający dane w sposób dynamiczny, np. wyniki obliczeń podczas wpisywania wartości do formularza), podpowiedzi pojawiające się po najechaniu na daną kontrolkę. Formularz jest interaktywny, sposób wyświetlania kontrolek może zmieniać się pod wpływem działań użytkownika.

Dla programisty użycie XForms oznacza istotną zmianę w podejściu do programowania. Wiele funkcji możemy zrealizować bez użycia Javascript, np.: wykonać proste obliczenia po stronie przeglądarki (nie wysyłając danych do serwera), możemy oznaczyć kontrolki jako obowiązkowe do wypełnienia, nadać im określony typ danych, wyświetlać komunikaty, dodawać lub ukrywać kontrolki w odpowiedzi na działania użytkownika.

Za pomocą jednego atrybutu (src) można wskazać zewnętrzny plik XML, w którym znajdują się dane i zostaną one wczytane do formularza. Nie musimy z góry wiedzieć ile jest danych, można zastosować instrukcję pętli w celu stworzenia dla każdego węzła pliku XML kontrolek. Można zapisać dane wpisane do formularza w postaci pliku XML bądź wysłać ten plik do usługi sieci Web lub bazy danych.

**Problemy z XForms**

Chcąc tworzyć XForms należy poznać bliżej język XML, zasady w nim panujące, tzw. przestrzenie nazw, warto też nauczyć się posługiwać arkuszami stylów CSS3.

Formularze XForms na razie (maj 2009r.) są tylko rekomendacją W3C (World Wide Web Consortium), nie są standardem. Aby formularz XForms mógł być wyświetlony w przeglądarce najczęściej trzeba zainstalować odpowiedni dodatek. Jest bardzo niewiele przeglądarek, które mają wbudowaną obsługę XForms natywnie. Jest jednak szansa, że się to zmieni, gdyż następna generacja języka HTML - XHTML 2.0 będzie zawierała XForms. Oznacza to, że producenci przeglądarek najprawdodpodobniej wyposażą je w niedalekiej przyszłości w możliwośc wyświetlania nowych formularzy.

**Jak zacząć tworzyć XForms**

Przede wszystkim trzeba sobie zainstalować odpowiedni dodatek do przeglądarki jakiej używamy. Ten artykuł napisany jest na przykładzie przeglądarki Mozilla Firefox 3.0.10. Dodatek do niej można ściągnąć ze strony Mozilla - dodatki. Należy też zaopatrzyć się w edytor plików tekstowych, może być windowsowy Notatnik, lub np. bardziej zaawansowany - KeD.

XForms możemy osadzać w dokumentach XHTML. Nie ma czegoś takiego jak samodzielny dokument XForms. Dokument powinien mieć rozszerzenie xhtml.

Przykład prostej aplikacji XForms

Język XForms ma bardzo wiele ciekawych możliwości i jak każde narzędzie o rozbudowanej funkcjonalności ma dość skomplikowaną budowę. Przykład przedstawiony w tym punkcie jest bardzo prosty ma za zadanie zilustrować zasady budowy formularzy oraz kilka wybranych funkcji XForms, takich jak: inicjacja formularza z danymi pobranymi z zewnętrznego pliku XML, funkcja pętli (repeat), proste kalkulacje i kontrolkę kalendarza. Przykład przedstawia formularz zamówienia na materiały biurowe, składający się z tabeli, której kolumny to nazwa pozycji, ilość sztuk, cena oraz wartość (rysunek 1). Dane pobierane są z zewnętrznego pliku „zamowienie.xml” o następującej strukturze:

<towar>

<nazwa>Notes samoprzylepny żółty 102x76</nazwa>

<ilosc>100</ilosc>

<cena>2.40</cena>

<wartosc/>

</towar>

<towar>

<nazwa>Długopis Bic</nazwa>

<ilosc>50</ilosc>

<cena>0.90</cena>

<wartosc/>

</towar>

<towar>

<nazwa>Zszywacz Eagle 930B</nazwa>

<ilosc>3</ilosc>

<cena>9.50</cena>

<wartosc/>

</towar>

<towar>

<nazwa>Koszulka A4 Esselte 100szt. w folii </nazwa>

<ilosc>10</ilosc>

<cena>7.29</cena>

<wartosc/>

</towar>

<wartoscZamowienia/>

<data/>

</zamowienie>

W formularzu nazwa i cena są wartościami wyświetlanymi jako etykiety - tylko do odczytu, natomiast ilość sztuk jest wyświetlana w okienku edycyjnym i może być zmieniana przez użytkownika (osobę zamawiającą). Po każdej zmianie ilości sztuk formularz przelicza całkowitą wartość zamówienia oraz wartości cząstkowe dla poszczególnych pozycji. Poniższy kod przedstawia formularz umieszczony w dokumencie XHTML, aby nie zaciemniać idei funkcjonowania formularza pominięte zostały znaczniki formatujące wygląd poszczególnych elementów.

1 <?xml version="1.0" encoding="utf-8"?>

2 <html xmlns="http://www.w3.org/1999/xhtml"

3 xmlns:xforms="http://www.w3.org/2002/xforms"

4 xmlns:xs="http://www.w3.org/2001/XMLSchema">

5 <head>

6 <title>Formularz zamówienia</title>

7 <xforms:model>

8 <xforms:instance src="zamowienie.xml"/>

9 <xforms:bind ref="towar/wartosc" type="float"/>

10 <xforms:bind nodeset="data" type="xs:date" />

11 <xforms:bind calculate="(../ilosc)\*(../cena)" nodeset="towar/wartosc" />

12 <xforms:bind calculate="sum(../towar/wartosc)" nodeset="wartoscZamowienia" />

13 </xforms:model>

14 </head>

15 <body>

16 <h1>Zamówienie na materiały biurowe

17 <xforms:input ref="data"><xforms:label> z dnia: </xforms:label> </xforms:input>

18 </h1>

19 <xforms:repeat nodeset="towar">

20 <xforms:output ref="nazwa" />

21 <xforms:input ref="ilosc" />

22 <xforms:output ref="cena" />zł

23 <xforms:output ref="wartosc" incremental="true"/> zł

24 </xforms:repeat>

25 <xforms:output ref="wartoscZamowienia" incremental="true">

26 <xforms:label>Ogółem: </xforms:label>

27 </xforms:output>

28 </body>

29 </html>

Formularz składa się z modelu dokumentu (wiersze od 7 do 13), gdzie deklaruje się strukturę dokumentu XML, który jest tworzony za pomocą formularza oraz umieszcza się deklaracje typów danych, formuły obliczeń i ścieżkę do pliku z danymi inicjującymi (linia 8).

Linia 17 wyświetla kontrolkę kalendarza

Wiersze 19 i 24 stanowią instrukcję pętli powtarzającej polecenia wyświetlania kontrolek formularza dla wybranych węzłów dokumentu XML (w tym przypadku dla każdego zamawianego towaru).

Linie 25 do 27 powodują wyświetlenie sumy ogólnej zamówienia.

**XLink**

* XLink (XML Linking Language) jest specyfikacją W3C służącą do tworzenia łączy URI w dokumentach XML-a.
* XLink jest odpowiednikiem elementu a w HTML (i w XHTML), jest jednak od niego dużo bardziej rozwinięty. XLink nie tylko używa się do tworzenia odnośników ale również łączenia dokumentu z innym.
* XLink umożliwia wstawianie do dokumentów XML elementów tworzących i opisujących łącza między zasobami.

XLink – atrybuty:

* **type** – okresla typ łącza, możliwe wartości to: simple, extended, arc, locator, title.
* **href** – określa adres URI łącza
* **role** – określa funkcję docelowego zasobu w sposób czytelny dla komputera
* **arcrole** – określa funkcję łuku
* **title** – określa tytuł łącza
* **label** – określa etykietę dla typu **locator**
* **from** – określa poprzedni punkt łuku
* **to** – określa następny punkt łuku
* **show** – wskazuje miejsce wyświetlania wskazanego zasobu, może przyjmować wartości: new, replace, embed, other
* **actuate** – określa jak łącze ma być uruchomione, może przyjmować
* wartości: onLoad, onReguest, other

Łącza

**Łącza proste** to połączenie dwóch zasobów – zasobu lokalnego i zasobu zdalnego (**type=”simple”**). Łącza proste są odpowiednikiem elementu **a** z HTML-a.

**Łącza rozszerzone** wiążą wiele różnych zasobów, które mogą być kombinacją zasobów lokalnych i zdalnych (**type=”extended”**). Łącza rozszerzone nie są obsługiwane przez dostępne oprogramowanie.

**Łącze lokalne** określane jest przez element, który wraz z zawartością jest umieszczany wewnątrz łącza rozszerzonego, tworzy wskazanie konkretnego zasobu (**type=”resource”**)

**Łącze zdalne** określane jest przez element, który wraz z zawartością jest umieszczany wewnątrz łącza rozszerzonego, tworzy lokalizator (type=”locator”)

**Łuki**

Każda możliwa ścieżka między dwoma zasobami nazywana jest **łukiem.**

Inaczej określając, **łuk jest przejściem z jednego zasobu do drugiego zasobu w określonej kolejności (type=”arc”)**

Za pomocą atrybutu **from** określamy, skąd ma zostać wywołane połączenie. Za pomocą atrybutu **to** określamy, dokąd chcemy pójść.

**Pokazywanie i uruchamianie**

**show:**

 replace – zamienia nowym zasobem zawartość okna

 new – otwiera nowe okno z zawartością

 embed – włącza wskazany zasób do bieżącego zasobu

**actuate:**

 onRequest – łącze ma być uruchamiane na żądanie użytkownika

 onLoad – łącze ma być uruchamiane po załadowaniu dokumentu

Fragment dokumentu Xlink:

<znacznik xmlns:xlink="http://www.w3.org/1999/xlink"

xlink:type="simple"

xlink:show="replace"

xlink:href="http://www.przyklad.pl/przyklad.xml">

To jest bardzo prosty odnośnik

</znacznik>

Powyższy przykład w działaniu jest bardzo zbliżony do działania odnośnika w XHTML-u:

<a href="http://www.przyklad.pl/przyklad.xml">To jest bardzo prosty odnośnik</a>

**XInclude**

XInclude zdefiniowany jest w rekomendacji W3C

XML Inclusions (XInclude) to aplikacja XML służąca do wstawiania zawartości do pliku

XInclude można porównać z technologiami po stronie **serwera,** takimi jak:

 instrukcja include() z PHP

 instrukcja require() z PHP

XInclude można porównać z technologiami po stronie **klienta,** takimi jak: XLink, encjami zewnętrznymi z DTD, regułą @import z CSS, całą grupą znaczników z różnych aplikacji XML,

np. z elementami <include /> i <import /> z XSL.

**XInclude różni od XLink** (show="embed") tym, że ma określony model przetwarzania. W XInclude przetwarzanie odbywa się na

niższym poziomie i „gotowy” kod jest przekazywany na wyższy poziom.

**XInclude vs. DTD:** Największą wadą encji zewnętrznych jest to, że przy niepowodzeniu ładowania pliku (np. gdy pliku nie ma) parser musi zasygnalizować błąd krytyczny. XInclude natomiast daje możliwość zadeklarowania wartości, która zostanie

wyświetlona gdy wystąpi błąd przy ładowaniu pliku.

Element include – atrybuty:

**href** - wartością tego atrybutu jest adres URI lub IRI. Atrybut ten jest opcjonalny. Jego brak oznacza dokładnie to samo co href=""

**xpointer** - identyfikacja części źródła dokumentu do wstawienia

**parse** - informuje w jaki sposób wstawiać źródło, czy jako tekst (text) czy jako parsowany XML (xml). Gdy użyjemy text, nie możemy używać atrybutu xpointer. Jest opcjonalny,

domyślna wartość to xml

**encoding** - atrybut ten może być stosowany tylko dla parse="text", gdyż właśnie w plikach tekstowych czasami trudno określić kodowanie

**accept** i **accept-language** – wspomagają ustalenie zawartości. Wartości tych atrybutów są mapowane odpowiednio na nagłówki Accept i Accept-language z HTTP.

**Błędy źródła**

Błąd źródła to błąd, który odnosi się do nieudanej próby pobrania źródła z adresu URL. Błąd źródła powinien być rozwiązany przez

zachowanie rezerwy awaryjnej. Rezerwa awaryjna wiąże się z przetworzeniem zawartości **elementu fallback.** Gdy brakuje elementu fallback, wtedy błąd źródłowy prowadzi do błędu krytycznego.

Element **fallback** powinien być potomkiem elementu **include**.

**Element fallback nie posiada atrybutów. Zawartością elementu fallback powinna być treść, wyświetlana w przypadku błędu.** Może to być tylko sam tekst lub tekst z elementami z innych przestrzeni nazw (bez względu na wartość atrybutu parse).

**Błędy krytyczne**

Z błędem krytycznym związane są sytuacje, które uniemożliwiają przetwarzanie

Przykłady:

 atrybut parse ma inne wartości niż text i xml,

 w elemencie include znajduje się więcej niż jeden

element fallback

 gdy źródło ma niepoprawny XML (nie well-formed)

Przykład:

...

<xi:include parse=”xml”

href=”plik.xml”>

<xi:fallback>Brak pliku</xi:fallback>

</xi:include>

...

**XSLT**

XSLT (ang. XSL Transformations, Extensible Stylesheet Language Transformations, w wolnym tłumaczeniu Przekształcenia Rozszerzalnego Języka Arkuszy Stylów) – oparty na XML-u język przekształceń dokumentów XML. Pozwala na przetłumaczenie dokumentów z jednego formatu XML na dowolny inny format zgodny ze składnią XML-a (np. na stronę WWW XHTML, wzór matematyczny MathML lub dokument biurowy ODF), jak również na zwykły HTML i czysty tekst.

Dzięki dużej sile wyrazu, łatwości implementacji i powszechnemu stosowaniu XML-a jako standardu dla zapisu informacji, XSLT jest uniwersalnym narzędziem znajdującym zastosowanie w wielu rodzajach oprogramowania. Najbardziej popularne to generowanie stron WWW w serwisach internetowych oraz konwersja pomiędzy alternatywnymi formatami np. w pakietach biurowych.

XSLT jest rozwijany przez W3C jako część rodziny języków XSL (obok XPath i XSL-FO). Powstał pod wpływem języków funkcyjnych oraz języków opartych na dopasowywaniu wzorców (ang. pattern matching) jak awk. Jego bezpośrednim poprzednikiem jest DSSSL, odpowiednik XSLT dla SGML-a.

Najnowszą wersją rekomendacji XSLT jest 2.0, ale ze względu na jej powolne rozpowszechnienie (vide brak obsługi w Xalanie, czy "fabrycznie" na platformie .NET) rekomendacja 1.0 jest nadal powszechnie wykorzystywana.

XSLT przypomina języki funkcyjne - arkusze XSLT zbudowane są z reguł opisujących, w jaki sposób zamienić poszczególne elementy wejściowego XML-a. Warsztat programisty XSLT obejmuje m.in. instrukcje sterujące, możliwość definicji własnych funkcji (tzw. szablonów nazwanych), funkcje wbudowane realizujące na przykład sortowanie. Do znajdowania i wskazywania elementów źródłowego XML-a używany jest Xpath.

Arkusze XSLT są poprawnymi dokumentami XML. Elementem głównym jest xsl:stylesheet. Prefiks xsl: w używanych w tym dokumencie nazwach elementów oznacza jedynie, że należą one do przestrzeni nazw XSLT - tak więc xsl:stylesheet należy czytać jako element stylesheet z przestrzeni nazw XSLT.

**Algorytm transformacji**

Każdy procesor XSLT posługuje się przedstawionym poniżej w skrócie algorytmem.

1. Przygotowanie do transformacji:

* Parsowany jest arkusz XSLT oraz wejściowy XML oraz budowane są ich drzewa. Uwzględniony jest fakt, że arkusz XSLT może się składać z wielu plików (instrukcje xsl:include i xsl:import).
* Z dokumentów usuwane są nadmiarowe białe znaki.
* Do drzewa XSLT dołączane są standardowe reguły.

2. Transformacja:

* Tworzony jest główny element drzewa wyjściowego (root node).
* Główna część: przetwarzane są elementy drzewa wejściowego, począwszy od elementu głównego.
* Zwracane jest drzewo wyjściowe, w formacie określonym przez xsl:output.

Każdy element drzewa wejściowego przetwarzany jest następująco:

* Znajdowany jest najlepiej pasujący szablon. Ze wszystkich szablonów pasujących do przetwarzanego elementu (każdy szablon nienazwany ma wzorzec - atrybut match) wybierany jest ten o najwyższym priorytecie (obliczonym na podstawie atrybutu priority, postaci wzorca oraz pozycji w dokumencie - elementy zaimportowane mają zawsze najniższy priorytet).
* Znaleziony szablon jest aplikowany. Elementy szablonu znajdujące się w przestrzeni nazw XSLT (zazwyczaj te z prefiksem xsl:) traktowane są jak instrukcje i odpowiednio interpretowane. Reszta jest zwyczajnie kopiowana do drzewa wynikowego.
* Jeśli w szablonie umieszczona jest instrukcja xsl:apply-templates, procesor przechodzi w tym miejscu do rekurencyjnego przetwarzania listy elementów wskazanych atrybutem select lub - jeśli go brak - wszystkich dzieci aktualnego elementu. Jeśli w szablonie brak jest instrukcji xsl:apply-templates, żadne z elementów aktualnego poddrzewa (dzieci i ich następniki) nie są w tym miejscu dopasowywane (przetwarzane). Należy jednak pamiętać, że mogą zostać przeznaczone do dopasowania (za pomocą instrukcji xsl:apply-templates) z innego szablonu.

**Przykład**

Powiedzmy, że procesor XSLT transformuje poniższy fragment dokumentu XML:

...

<cokolwiek>

<DzieckoCokolwiek skąd="skądkolwiek">bum</DzieckoCokolwiek>

po bum

</cokolwiek>

...

mając do dyspozycji następujące reguły:

<xsl:template match="cokolwiek/DzieckoCokolwiek[@skąd='skądkolwiek']" priority="-1">

<i>(szablon 1)</i>

</xsl:template>

<xsl:template match="cokolwiek/DzieckoCokolwiek">

<i>(szablon 2)</i>

</xsl:template>

<xsl:template match="DzieckoCokolwiek">

<i>(szablon 3)</i>

</xsl:template>

Produktem transformacji będzie

<i>(szablon 2)</i> po bum

Brak reguły dla cokolwiek, użyta więc będzie standardowa każąca przetwarzać wszystkie dzieci po kolei.

Do DzieckaCokolwiek pasują wszystkie 3 reguły, jednak pierwsza posiada nadany niski priorytet -1, trzecia zaś postać wzorca otrzyma priorytet -0,5 natomiast druga 0. Wybrana więc zostanie druga.

Wewnątrz niej brak instrukcji xsl:apply-templates, tekst bum zostaje więc pominięty.

Następnie przetwarzane jest drugie dziecko cokolwiek czyli tekst po bum. Brak dla niego reguły, stosowana więc jest standardowa.

**Zastosowanie w tworzeniu stron WWW**

XSLT jest szeroko stosowany przy tworzeniu stron WWW. Pozwala na oddzielenie warstwy informacji od warstwy prezentacji, które w HTML-u są mocno pomieszane. Procesory XSLT wbudowane są w nowoczesne przeglądarki WWW.

**Przykład**

Załóżmy, że mamy do przedstawienia jakąś książkę. Możemy informacje o niej zapisać w formacie XML:

<?xml version="1.0" encoding="UTF-8"?>

<Książka>

<id>

<Tytuł>Pobawmy się w webmasterów</Tytuł>

<isbn>123456789</isbn>

<Autor>Jeszcze nieznany</Autor>

</id>

<SpisTreści>

<R>Rozdział 1</R>

<R>Rozdział 2</R>

<R>Rozdział 3</R>

</SpisTreści>

</Książka>

Powyżej zawarte są wszystkie informacje, jakie posiadamy o tej książce. Jednak aby wyświetlić ją w przeglądarce potrzebne są jeszcze instrukcje, jak to zrobić. Do tego służy dokument XSLT:

<?xml version="1.0" encoding="UTF-8" ?>

<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns="http://www.w3.org/1999/xhtml">

<xsl:output method="xml" indent="yes"

doctype-public="-//W3C//DTD XHTML 1.1//EN"

doctype-system="http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"/>

<xsl:template match="Książka">

<html>

<head> <title>Książka ,,<xsl:value-of select="./id/Tytuł"/>''</title> </head>

<body>

<xsl:apply-templates/>

</body>

</html>

</xsl:template>

<xsl:template match="id">

<h3>Dane identyfikacyjne książki</h3>

Tytuł: <xsl:value-of select="Tytuł"/><br />

ISBN: <xsl:value-of select="isbn"/><br />

Autor: <xsl:value-of select="Autor"/><br />

</xsl:template>

<xsl:template match="SpisTreści">

<h3>Spis treści</h3>

<table>

<xsl:apply-templates/>

</table>

</xsl:template>

<xsl:template match="R">

<tr><td><xsl:value-of select="."/></td></tr>

</xsl:template>

</xsl:stylesheet>

Aby transformacja zadziałała, zakładając, że szablon znajduje się w pliku książka.xsl, w dokumencie XML dodajemy linijkę

<?xml version="1.0" encoding="UTF-8"?>

<?xml-stylesheet type="text/xsl" href="książka.xsl"?>

<Książka>

...

</Książka>