41

PROPOZYCJE i MATERIAŁY

Grzegorz Zieliński

XML
ROZSZERZALNY JĘZYK ZNAKOWAŃ

WYDAWNICTWO SBP

41

XML

ROZSZERZALNY JĘZYK ZNAKOWAŃ

POLISH LIBRARIANS ASSOCIATION

CONTRIBUTIONS AND MATERIALS

Grzegorz Zieliński

XML
EXTENSIBLE MARKUP LANGUAGE

WYDAWNICTWO SBP

WARSAW 2000

STOWARZYSZENIE BIBLIOTEKARZY POLSKICH

PROPOZYCJE I MATERIAŁY

Grzegorz Zieliński

XML
ROZSZERZALNY JĘZYK ZNAKOWAŃ

WYDAWNICTWO SBP

WARSZAWA 2000

Książka dofinansowana z funduszu środków pozabudżetowych Instytutu Informacji Naukowej i Studiów Bibliologicznych Uniwersytetu Warszawskiego

Recenzent
Mieczysław MURASZKIEWICZ

Projekt graficzny okładki i strony tytułowej
Wydawnictwo SBP

Redaktor tomu
Janusz NOWICKI

Redakcja techniczna i korekta
Anna LIS

ISBN 83-87629-50-2

CIP - Biblioteka Narodowa
Zieliński Grzegorz
XML - rozszerzalny język znakowań / Grzegorz Zieliński. - Warszawa : Wydaw. SBP, 2000. -
(Propozycje i Materiały / Stowarzyszenie Bibliotekarzy Polskich ; 41)

Wydawnictwo SBP, Warszawa 2000, Wyd. I. Ark. wyd. 3,25. Ark. druk. 6,25
Łamanie: Urszula LASOCKA
Druk i oprawa: Zakład Poligraficzny PRIMUM, Kozerki 17^
05-825 Grodzisk Mazowiecki, tel. 724-18-76

Od Autora

Niniejsza książka powstała na bazie pracy magisterskiej napisanej w Instytucie Informacji Naukowej i Studiów Bibliologicznych Uniwersytetu Warszawskiego w roku 2000. W tym miejscu chciałbym podziękować dyrektorowi Instytutu profesorowi Marcinowi Drzewieckiemu za umożliwienie napisania tej pracy i inicjatywę opublikowania jej drukiem.

Szczególne wyrazy wdzięczności kieruję także pod adresem mojego promotora, profesora Mieczysława Muraszkiewicza, bez którego wszechstronnej pomocy i otwartości na każdą propozycję zgłaszaną przez studentów nigdy nie podjąłbym się podobnego wyzwania.

WSTĘP

Internet zmienił świat. Nie ulega wątpliwości, że wkroczył przebojem do codziennego życia, stając się medium równie ważnym, co telewizja czy radio. Rozproszone po całym świecie komputery podłączone do sieci gromadzą niezmierzone zasoby informacji, często niemożliwych do uzyskania w tradycyjnych źródłach, takich jak książka czy czasopismo. Miliony ludzi codziennie przeczesują Web w poszukiwaniu najróżniejszych danych: od przepisów kulinarnych i rozkładów jazdy pociągów, przez prognozę pogody, oferty pracy, kursy walut, recenzje płyt i książek, aż po poważne rozprawy naukowe i wyniki najnowszych badań. To wszystko jest dostępne „w zasięgu" zaledwie kilku kliknięć. Ponadto łatwo jak nigdy dotąd, zamienić się z pasywnego odbiorcy serwowanych informacji w aktywnego nadawcę dowolnego komunikatu. Wystarcza do tego choćby minimalna znajomość stosunkowo prostego języka HTML pozwalającego tworzyć strony WWW, bądź też dowolne oprogramowanie generujące kod HTML niejako w ukryciu przed autorem.

Skoro jest to takie proste a sieć rozwija się dynamicznie i wydaje się spełniać swoje funkcje informacyjne znakomicie, skąd w takim razie wzięło się obserwowane w ostatnich trzech latach wielkie poruszenie wokół języka XML (Rozszerzalnego Języka Znakowań)? Wystarczy wziąć do ręki jakąkolwiek prasę fachową, aby natknąć się na sformułowania typu „lingua franca do wymiany danych" bądź „język, który zrewolucjonizuje Internet". Mówi się, żeby nie naprawiać tego, co nie jest zepsute. Czy więc wprowadzenie kolejnego standardu do Internetu nie spowoduje niepotrzebnego zamieszania?

Okazuje się, że nie, a niezwykła popularność rodzącej się technologii nie jest przesadzona. Potwierdza to żywe zainteresowanie XML-em ze strony komercyjnych uczestników rynku IT, oraz duża aktywność publicystyczna dotycząca tego języka. Po dokładniejszej analizie okazuje się, że sieć dnia dzisiejszego ma wiele luk, z którymi istniejące narzędzia nie mogą sobie poradzić. Uczynienie Internetu platformą naprawdę efektywnej wymiany danych wymaga nowych rozwiązań, którym naprzeciw wydaje się wychodzić właśnie Rozszerzalny Język Znakowań.

Niniejsza praca podejmuje próbę scharakteryzowania XML-a zarówno od strony teoretycznej, jak również towarzyszy jej element praktyczny, w postaci licznych przykładów ilustrujących omawiane zagadnienia.

W jej pierwszej części omówiono genezę języka XML i jego protoplastów, a także starano się wyjaśnić pewne nieporozumienia dotyczące roli, jaka z założenia miała mu przypaść. Ponadto wskazano pewne słabe punkty Internetu funkcjonującego zgodnie z dotychczasowymi standardami.

Drugi rozdział jest charakterystyką języka omawiającą jego kluczowe cechy, obszary, w których znajduje on zastosowanie zastępując dotychczasowe rozwią-

zania, oraz implikacje wynikające z wprowadzenia go do sieci jako obowiązującego standardu.

Część trzecia opiera się na specyfikacji XML-a i opisuje zasady tworzenia dokumentów zgodnych z tym formatem.

Zadaniem kolejnego rozdziału jest pokazanie w praktyczny sposób tworzenia dokumentów XML, czemu ma służyć przedstawienie pięciu konkretnych, zróżnicowanych i w pełni funkcjonalnych przykładów.

Wreszcie część piąta omawia pewne dodatkowe standardy, które są niezbędne w celu pełnego zastosowania XML-a w środowisku WWW. Dotyczą one sposobu prezentacji dokumentów XML oraz tworzenia połączeń hipertekstowych.

Rozdział 1

FILOGENEZA XML-A

1.1. SGML (STANDARD GENERALIZED MARKUP LANGUAGE)

Po raz pierwszy pomysł stworzenia uniwersalnego, otwartego standardu wymiany i przetwarzania ustrukturyzowanych dokumentów pojawił się na spotkaniu, które odbyło się we wrześniu 1967 roku w Canadian Government Printing Office[1]. W swoim wystąpieniu William Tunnicliffe, prezes GCA (Graphic Communications Association), zasugerował oddzielenie treści dokumentu od informacji formatujących. Również w tamtym okresie Stanley Rice, który zajmował się opracowaniem graficznym książek, zaproponował odpowiedni zestaw znaczników. Dzięki wysiłkom GCA powstał komitet GenCode, który zdefiniował sposób znakowania ukierunkowany na hierarchię dokumentu. Był to pewnego rodzaju kod, mający na celu umożliwienie korzystania z archiwów danych zawartych w dokumentach pochodzących od różnych producentów.

Kolejną taką próbą była idea Generalized Markup Language (GML) rozwijana przez IBM, której podstawą było założenie stworzenia narzędzia umożliwiającego „produkowanie" dokumentów najróżniejszych typów – od podręczników przez notatki i artykuły prasowe do oficjalnych kontraktów i dokumentacji technicznej – przy użyciu tych samych plików źródłowych. Autorami tego języka byli Charles Goldfarb, Edward Mosher oraz Raymond Lorie, którzy w 1969 roku opracowali pierwszą jego wersję (akronim GML pochodzi także od nazwisk twórców). GML był rezultatem projektu zintegrowanego systemu informacyjnego dla kancelarii prawniczych i pozwalał na edytowanie, formatowanie oraz przeszukiwanie zbiorów danych

[1] Historia powstania języka SGML za: R. Lewandowski: *Standard Generalized Markup Language – SGML*. http://rafal.clpz.poznan.pl/Sgml/

tekstowych. Autorzy projektu w znacznej mierze opierali się na wcześniejszych propozycjach Tunnicliffe'a i Rice'a. Tu po raz pierwszy pojawiły się znane do dzisiaj konstrukcje zwane tagami. Język ten cechowały stosunkowo proste reguły składniowe, a także liczne ułatwienia dla tworzących dokumenty (jak np. możliwość opuszczania najbardziej oczywistych tagów). Wprawdzie znacznie upraszczało to sam proces pisania a także mogło poprawić czytelność tak „skróconych" dokumentów, pociągało to za sobą konieczność opracowywania osobnych kompilatorów dla każdego ich typu w celu dostarczenia do aplikacji przetwarzającej wejściowej informacji w odpowiednim formacie. GML zyskał znaczną akceptację, szczególnie w IBM, dla którego został stworzony. Ocenia się, że 90% dokumentów elektronicznych IBM zostało stworzonych za pomocą GML-a[2]. Jednak lawinowo rosnąca liczba rodzajów dokumentów wymagających elektronicznego przetwarzania wymogła konieczność stworzenia bardziej uniwersalnego narzędzia.

Pierwszym krokiem ku standaryzacji było powołanie do życia w 1978 r. Komitetu Języków Komputerowych dla Przetwarzania Tekstu (Computer Languages for the Processing of Text Commitee) przy Amerykańskim Narodowym Instytucie Standaryzacyjnym (American National Standards Institute). Charles Goldfarb pracował w tym komitecie nad opracowaniem standardowego języka opisu tekstu bazującego na GML-u. Dzięki pomocy komitetu GenCode w 1980 roku powstała pierwsza wersja robocza (*ang. working draft*) standardu SGML (Standard Generalized Markup Language). W 1983 r. szósta z roboczych wersji została zarejestrowana przez GCA jako standard przemysłowy. Projekt zyskał również autoryzację Międzynarodowej Organizacji Standaryzacyjnej (ISO). W październiku 1985 r. ukazała się międzynarodowa wersja robocza, a w roku 1986 SGML przyjęto jako ogólnoświatowy standard pod numerem ISO 8879:1986[3], definiujący metody reprezentacji tekstu w postaci elektronicznej, niezależnej od platformy sprzętowej i systemu operacyjnego.

Język ten miały z założenia charakteryzować trzy cechy[4]:

– Formalność – umożliwienie sprawdzenia poprawności składniowej dokumentów przed ich przetworzeniem w procesie tzw. walidacji.

– Strukturalność – możliwość kompleksowej obsługi dokumentów o skomplikowanej strukturze.

– Rozszerzalność – pozwalająca na manipulowanie wielkimi repozytoriami (składnicami) informacji.

SGML jest metajęzykiem, który określa zasady tworzenia języków opisujących klasy dokumentów. Pozwala on na oznaczanie tekstu w sposób

---

[2] R. Księżyk, J. Staszelis: *Na początku było słowo...* http://www.fuw.edu.pl/~ksiezyk/sgml4gw.html.

[3] International Organization for Standardization: *ISO 8879:1986. Information processing. Text and Office Systems. Standard Generalized Markup Language (SGML)*. Geneva, ISO 1986.

[4] A. Rifkin: *A look at XML*, http://www.webdeveloper.com/xml/xml a_look_at xml.html.

ogólny (ang. *generic markup*), zwane również oznaczaniem opisowym (ang. *descriptive markup*)[5]. Zawarte w tekście znaczniki niosą informację znaczeniową i, w przeciwieństwie do oznaczania proceduralnego (ang. *procedural markup*), nie informują systemu o wyglądzie dokumentu, rodzaju użytych czcionek, rozmieszczeniu elementów tekstu na stronie. SGML opisuje nie tylko zawartość dokumentu (tekst, ilustracje itp.), lecz również jego strukturę logiczną (słowo, akapit, rozdział, sekcja). Pozwala także na tworzenie dokumentów składających się z wielu plików, zachowując ich integralność. Znaczniki są obiektami opisującym tekst. Na początku i na końcu każdego z nich znajdują się znaki (ograniczniki), które pozwalają na wyraźne oddzielenie obiektów opisujących tekst od zasadniczej treści dokumentu.

Standard ten stał się ważnym składnikiem wielu systemów przetwarzania dokumentów[6]. Jako jedni z pierwszych wdrożyli go amerykańscy wydawcy (Association of American Publishers), by usprawnić elektroniczną wymianę rękopisów między autorami, redaktorami i wydawnictwami[7]. W 1988 Departament Obrony USA uczynił SGML standardem (jako tzw. CALS) dokumentacji technicznej sprzętu wojskowego dostarczanego armii amerykańskiej, podobnie później uczyniła grupa producentów półprzewodników (Hitachi, Intel, National Semiconductor, Philips Semiconductors, Texas Instruments) tworząc standard PICS, oraz Air Transport Association z rozwiązaniem ATA-2100 przeznaczonym dla przemysłu lotniczego. SGML znalazł także zastosowanie w środowiskach akademickich, gdzie powstały bazy tekstów klasyki literatury światowej otagowanych w SGML-u zgodnie z Text Encoding Initiative (TEI).

Obecnie największym projektem w Polsce jest wdrożenie systemu SGML-owego w Wydawnictwie Naukowym PWN[8]. PWN przenosi swoje publikacje encyklopedyczne i słowniki na formę elektroniczną z użyciem SGML-a. Ma to ułatwić wykorzystanie i uaktualnianie istniejących materiałów oraz równoległe publikowanie ich wersji elektronicznych. Warto nadmienić, że 14 IX 1998 Normalizacyjna Komisja Problemowa (NKP) nr 242 ds. Informacji i Dokumentacji podjęła uchwałę o przystąpieniu do tłumaczeniu na język polski normy ISO 8879 SGML. Prace nad polską wersją SGML rozpoczęły się na początku 1999 roku[9].

SGML nie jest rozwiązaniem wolnym od wad – szczególnie, jeśli patrzymy na ten język pod kątem WWW. Przede wszystkim SGML jest standardem drogim. Dodanie jego obsługi do jakiejkolwiek aplikacji (np. procesora

---

[5] R. Lewandowski: *Standard Generalized Markup Language – SGML*. http://rafal.clpz.poznan.pl/Sgml/.

[6] Eksperci czasem nazywają dokumenty SGML „produktami informacyjnymi" (*ang. information products*). T. Freter: *XML: Mastering information on the Web*. http://www.sun.com/980310/xml/.

[7] Przykłady za: R.Księżyk, J. Staszelis: *Na początku było słowo...* http://www.fuw.edu.pl/~ksiezyk/sgml4gw.html.

[8] Ibid.

[9] R. Księżyk: *Zdarzenia*. http://www.fuw.edu.pl/~ksiezyk/ml_events_pl.html.

tekstu czy też przeglądarki internetowej) – tu trzeba wiedzieć, że musi on być zaimplementowany w całości – zwiększa jej cenę dwu-, trzykrotnie[10]. Jego implementacja jest dość skomplikowana, gdyż z założenia służy on do rozwiązywania złożonych problemów, stąd jego użycie ogranicza się tylko do dużych systemów, których zasięg i ranga usprawiedliwia wysokie koszty wdrożenia. Dokumenty umieszczane w sieci rzadko wymagają aż tak potężnego narzędzia, więc producenci software'u internetowego jasno dają do zrozumienia, że nie mają zamiaru oferować wsparcia dla standardu SGML.

Ponadto brak przyjaznego oprogramowania do edycji, nieliczne i stosunkowo skomplikowane narzędzia do formatowania i przetwarzania tekstów SGML, oraz nieświadomość i niewiedza użytkowników – pamiętać należy, że jest to standard dość skomplikowany i jednocześnie restrykcyjny, więc tworzenie dokumentów jest dużo trudniejsze niż z wykorzystaniem HTML-a – powodują, że podstawowe cechy i przyczyny popularności Internetu tj. powszechność dostępu i łatwość publikowania, kłóciłoby się z wymaganiami stawianymi przez SGML.

1.2. HTML (HYPERTEXT MARKUP LANGUAGE)

Pod koniec lat osiemdziesiątych, pracujący w szwajcarskim laboratorium CERN (Collective of European High-Energy Physics Researchers), pomysłodawca idei World Wide Web, Tim Berners Lee używając przeglądarki i jednocześnie edytora NeXUS zaadoptował jeden ze schematów dokumentów zgodnych z SGML-em tworząc własny zestaw tagów i wzbogacając go jednym, rewolucyjnym dodatkiem: mechanizmem hiperłączy (linków). Tak powstał język HTML, będący niczym innym jak aplikacją SGML-a, przeznaczoną do publikowania dokumentów hipertekstowych w Internecie.

Kiedy w roku 1992 rodziła się pierwsza wersja HTML-a, niewielki zbiór znaczników, który oferowała, wydawał się wystarczający do opisania dokumentów mających znaleźć się w niezbyt rozwiniętej ówcześnie sieci a łatwość ich tworzenia oraz kontrola nad dalszym rozwojem języka poprzez zakładaną konieczność implementacji kolejnych, oficjalnych wersji schematów dokumentów zgodnych z SGML-em miały zapewnić harmonijny i przewidywalny rozwój standardu.

Społeczność internetowa jako oficjalny standard przyjęła dopiero opublikowaną w roku 1995 przez Internet Engineering Task Force (IETF) wersję HTML 2.0. Kolejne, rozszerzone specyfikacje tego języka opracowywało już

[10] S. Sol: *Introduction to XML for Web developers.* http://wdvl.internet.com/Authoring/Languages/XML/Tutorials/Intro/toc.html.

Konsorcjum WWW (World Wide Web Consortium), a ukazywały się one kolejno w 1996 (HTML 3.2), 1997 (HTML 4.0) i 1999 (HTML 4.01) roku.

HTML z założenia miał charakteryzować się:

– Prostotą – tworzenie dokumentów miało być maksymalnie ułatwione, a więc możliwe nie tylko dla wąskiej grupy programistów.

– Możliwością osadzania w dokumentach multimediów (grafiki, dźwięku, animacji itp.).

– Elastycznością umożliwiającą wspieranie mechanizmów hipertekstowych.

Przez wszystkie te lata HTML stał się najbardziej znanym markup językiem i odniósł największy sukces w dziedzinie elektronicznych publikacji. Zdefiniowany w nim zestaw tagów miał za zadanie kojarzyć reguły formatowania z odpowiednimi porcjami tekstu. Dokumenty, które zostały poddane adjustacji HTML (tzn. zawierające tekst wraz ze znacznikami), czytane są przez aplikacje przetwarzające (np. przeglądarki), które wiedzą jak wyświetlić zawartość zgodnie z określonymi wcześniej regułami. Jednak również i to rozwiązanie to nie jest wolne od wad:

1. Niemożność oddzielenia znaczenia danych od sposobu ich prezentacji

Pierwotnym założeniem twórców języka HTML było znakowanie opisujące znaczenie informacji zawartej w dokumencie bez dawania jakichkolwiek wskazówek, co do sposobu jej prezentacji[11]. Innymi słowy tytuł dokumentu, nagłówek, wyróżniony fragment tekstu czy adres autora miały znaleźć się odpowiednio wewnątrz elementów oznaczonych tagami TITLE, H1, EM i ADDRESS. Powodem takiego rozwiązania było skądinąd słuszne założenie, że przeglądarka „zna" dużo lepiej zarówno preferencje użytkownika, jak i środowisko, w którym działa, więc na podstawie tych informacji powinna sama zadecydować jak wyświetlić np. nagłówek pierwszego poziomu. Wśród odbiorców mogli się znaleźć tacy, którzy np. mieli kłopoty ze wzrokiem i musieli używać większych czcionek niż standardowe, czy też posiadacze przeglądarek działających wyłącznie w trybie tekstowym. Stąd podejmowano próby ograniczenia ingerencji autorów w wygląd tworzonych przez nich dokumentów.

Z czasem standard ten zaczął ewoluować w kierunku urozmaicenia sposobów prezentacji a wprowadzane znaczniki w żaden sposób nie określały zawartości elementu (np.: FONT, CENTER, BGCOLOR). Ponadto ostra konkurencja na rynku przeglądarek podsycana ciągłym dążeniem do tworzenia coraz efektowniejszych stron, w połączeniu z długotrwałym proce-

[11] L. M. Garshol: *Introduction to XML*.
http://www.stud.ifi.uio.no/~lmariusg/download/xml/xml_eng.html.

sem rozwijania standardu przez instytucje normalizacyjne spowodowała, że poszczególni producenci zaczęli na własną rękę dodawać do swoich produktów obsługę nowych znaczników, najczęściej w żaden sposób niekompatybilnych z konkurencją. Stąd po pewnym czasie HTML stał się językiem prezentacyjnym, którego poszczególne „dialekty" były zrozumiałe tylko przez konkretne produkty jednego dostawcy.

Sieć WWW stała się w pewnym sensie nieco bardziej wyszukanym faksem[12], przesyłającym dokumenty każdemu, kto o to poprosi. Co więcej strony prezentowane na Webie „są (...) tylko do odczytu i tylko przez człowieka. Dane, które są w nich zawarte nie mogą być wykorzystane przez inne aplikacje. Użytkownik musi ręcznie wywołać interesującą stronę i wyczytać z niej istotne informacje"[13]. Następujący fragment kodu HTML (tabela z kursem dolara):

```
<table>
<tr>
  <th>Waluta</th><th>Kupno</th><th>Sprzedaż</th><th>Data</th>
</tr>
<tr>
  <td>USD</td><td>4,20 zł</td><td>4,57 zł</td><td>21.01.2000</td>
</tr>
</table>
```

będzie zrozumiały dla odbiorcy dopiero po wyświetleniu go przez przeglądarkę. Natomiast w żaden sposób nie da się jej zmusić do zaprezentowania tejże tabeli w odpowiedzi na pytanie użytkownika: „jaki był kurs dolara w dniu 21 stycznia 2000 roku?", gdyż zastosowane znaczniki nie oddają sensu zawartych w nich informacji. Sytuacja zmieni się, gdy sieć zacznie korzystać z formatów oferujących odpowiednio przygotowane dane. Wtedy też w pełni efektywnie można będzie korzystać z aplikacji automatyzujących rutynowe działania jak np. pobieranie z sieci aktualnych kursów walut.

Osadzenie sposobu prezentacji wśród danych implikuje konieczność modyfikacji całego kodu HTML w przypadku zmiany wyglądu dokumentu (np. przekształcenie danych z tabeli do postaci uporządkowanej listy). Wreszcie odszukanie jakiejkolwiek informacji w olbrzymim zbiorze nieustrukturyzowanych danych, jakimi są pliki HTML, najczęściej sprowadza się do napisania odpowiedniego skryptu przeszukującego dokumenty słowo po słowie, porównując je z zadanym wzorcem. Takie rozwiązanie jest nieefektywne przede wszystkim ze względu na ogrom danych do przeanalizowania, jak i brak jakiejkolwiek możliwości uwzględnienia kontekstu, w jakim to słowo się pojawi.

---

[12] J. Bosak, T. Bray: *XML and the Second-Generation Web*. http://www.sciam.com/1999/0599issue/0599bosak.html.

[13] R. Księżyk: *XML dla o(d)pornych – cz.1.: Jak skomputeryzować Internet?* „PC Kurier" 1999 nr 9 s. 63.

2. Brak rozszerzalności

HTML nie pozwala na tworzenie własnych tagów i atrybutów, aby lepiej sparametryzować i semantycznie uporządkować przedstawiane dane[14]. Dostarczane przez twórców specyfikacji zbiory predefiniowanych tagów pozwalają jedynie na opis nagłówka dokumentu (HEAD) i treści właściwej (BODY). Bardziej szczegółowe elementy konstrukcji dokumentu jak np. streszczenia, rozdziały czy bibliografia nie dają się w sposób bezpośredni, za pomocą znaczników HTML, wyrazić.

Tym bardziej trudno sobie wyobrazić, aby kolejne wersje standardu dostarczyły narzędzi pozwalających na precyzyjne opisanie elementów zawierających wzory chemiczne, równania matematyczne, notację muzyczną, sonety, analizy finansowe itp. Praktycznie każde środowisko, gałąź przemysłu, nauka czy zwykła grupa hobbystów używająca Internetu generuje specyficzne, znacznie zróżnicowane komunikaty, których próba ogarnięcia i unifikacji w jeden wspólny standard jest z góry skazana na niepowodzenie.

3. Słabość wewnętrznej struktury HTML-a

Przejawia się ona w dwóch aspektach:

– HTML nie pozwala na odwzorowywanie głębszych zależności strukturalnych jak schematy baz danych czy hierarchie zorientowane obiektowo[15]. Co więcej, bardzo łatwo jest stworzyć poprawny składniowo dokument HTML, który pod względem semantycznym nie ma sensu[16]. Związane to jest między innymi z zawartością sekcji BODY, która pozwala na występowanie dozwolonych w niej elementów w praktycznie dowolnej kolejności. Stąd brak formalnych przeciwwskazań, aby np. nagłówek trzeciego poziomu H3 zawierał w sobie kilka nagłówków pierwszego H1 i drugiego poziomu H2.

– Brak możliwości sprawdzenia przez aplikacje strukturalnej poprawności przetwarzanego kodu (tzw. parsowanie). Doprowadza to do paradoksalnej sytuacji, w której nieprawidłowy dokument HTML jest poprawnie wyświetlany przez przeglądarkę. Przykład[17]:

```
<HEAD>
<META KEYWORD="zły dokument HTML>
<BODY BGCOLOR="005MF">
```

---

[14] J. Bosak: *XML, Java, and the future of the Web.* http://sunsite.unc.edu/pub/suninfo/standards/xml/why/xmlapps.htm.

[15] Ibid.

[16] L. M. Garshol: *Introduction to XML.* http://www.stud.ifi.uio.no/~lmariusg/download/xml/xml_eng.html.

[17] E. Ladd, J. O'Donnell i inni: *HTML 4, DHTML, VRML, XML.* Warszawa, Wydawnictwo Lynx-SFT 1999 s. 374.

```
<H1>Nagłówek</H3>
<UL>
<LI>Pozycja pierwsza
</UL>
```

Błędy występujące w tym przykładzie:
– brak wymaganego elementu TITLE
– brak domknięcia (") wartości atrybutu KEYWORD
– wartość atrybutu BGCOLOR nie jest prawidłową wartością szesnastkową
– pomieszanie etykiet nagłówka H1 i H3
– brak domknięcia tagu LI

Przeglądarki są zbyt mało rygorystyczne, jeśli chodzi o nieprawidłowy kod i cechuje je swoiste „domyślanie się", o co chodziło autorowi dokumentu. Jednakże wobec nasilającego się trendu w kierunku automatycznego przetwarzania dokumentów sieciowych, koniecznością stanie się ścisłe przestrzeganie prawidłowej składni. W przeciwnym razie niemożliwe stanie się stworzenie oprogramowania dokonującego prawidłowej analizy gramatycznej dokumentów.

1.3. XML (EXTENSIBLE MARKUP LANGUAGE) A SGML I HTML

W 1996 roku rozgorzała dyskusja nad stworzeniem nowego języka znakowania, który dawałby możliwości i „rozszerzalność" SGML-a, będąc jednocześnie równie prostym jak HTML. W sierpniu w Seattle została zorganizowana przez GCA konferencja gromadząca ekspertów związanych z środowiskiem SGML, której celem była dyskusja nad wprowadzeniem tego standardu do Internetu. W jednym z najważniejszych wystąpień Jon Bosak z Sun Microsystems zwrócił uwagę dwa podstawowe problemy[18]:
– klasy aplikacji, którym HTML nie oferuje odpowiedniego formatu informacji
– cechy SGML-a utrudniające zaakceptowanie go jako powszechnej technologii informacyjnej.

Ten drugi aspekt w efekcie zamienił się w rozważania, jak zmodyfikować SGML-a, by odpowiadał on potrzebom sieci. Konkluzją było opracowanie listy cech „nieistotnych" dla środowiska WWW.

World Wide Web Consortium zdecydowało się powołać do życia grupę roboczą wybitnych specjalistów pod przewodnictwem Jona Bosaka z Sun

[18] T. Freter: *XML: Mastering information on the Web.* http://www.sun.com/980310/xml/.

Corporation dla zaadoptowania standardu SGML w Internecie. Grupa ta, nazwana SGML Editorial Review Board, we współpracy z SGML Working Group podjęła pracę nad tym zagadnieniem. Szybko okazało się, że prosta redefinicja SGML-a nie jest możliwa i należy stworzyć coś, wprawdzie silnie związanego z istniejącym standardem, ale jednak zupełnie nowego. Tak została powołana do życia XML Working Group, w której skład weszli zarówno przedstawiciele wielkich firm komputerowych (Sun, Microsoft, Hewlett-Packard, Netscape, Adobe, Fuji Xerox), jaki i sprzedawców związanych z rynkiem SGML i integratorów systemów (ArborText, Inso, SoftQuad, Grif, Texcel, Isogen), reprezentanci społeczności akademickiej (NCSA, TEI) oraz firmy wdrażające początkowe wersje projektu (DataChannel, Vignette). Praca została podzielona na trzy etapy[19]:

– Opracowanie składni nowego języka.

– Opracowanie nowego mechanizmu połączeń hipertekstowych opartego na tymże języku.

– Opracowanie narzędzia do jego prezentacji.

W pierwszej kolejności Bosak z zespołem zrobili to, co twórcy Javy zrobili z C++[20]. Wszystkie nieistotne, rzadko używane i nieprzydatne w sieci elementy charakterystyki SGML-a zostały odrzucone. To, co pozostało to spójny, zwarty mechanizm, który nazwano eXtensible Markup Language (XML[21]). Specyfikacja XML-a, napisana głównie przez Tima Braya i C.M. Sperberg-McQueena liczyła niewiele ponad 30 stron, co w porównaniu 500 stronami opisu technicznego SGML-a stanowiło olbrzymi postęp, tym bardziej, że wszystkie najbardziej użyteczne funkcje SGML-a zostały przez XML-a odziedziczone. Ocenia się, że XML posiada 80% mocy deskryptywnej SGML-a przy jedynie 20% jego złożoności[22].

Przez następne kilka lat język XML rozwijał się dalej przyciągając i przekonując do siebie zarówno pieniądze możnych sponsorów (Sun, Microsoft, IBM) jak i naukowców takich jak np. Peter Murray-Rust, który był jednym z głównych twórców CML (Chemical Markup Language), języka opartego na XML-u czy grupę matematyków pracujących nad MathML (Math Markup Language)[23]. W połowie 1997 roku rozpoczęły się prace nad The eXtensible Linking Language (XLL) – językiem do tworzenia połączeń hiperteksto-

[19] L. M. Garshol: *Introduction to XML*.
http://www.stud.ifi.uio.no/~lmariusg/download/xml/xml_eng.html.

[20] S. Sol: *Introduction to XML for Web developers*.
http://wdvl.internet.com/Authoring/Languages/XML/Tutorials/Intro/toc.html.

[21] Skrót ten został wymyślony przez Jamesa Clarka – szefa techniczego W3C XML Working Group.

[22] K. Sall: *XML: Structuring data for the Web*.
http://wdvl.com/Authoring/Languages/XML/Intro/.

[23] Bogatszą listę języków bazujących na XML-u oraz standardów z nim stowarzyszonych można znaleźć w następujących źródłach:
R. Księżyk: *XML dla o(d)pornych - cz.2.: Spójrzmy z góry na HTML!* „PC Kurier" 1999 nr 10 s. 68-70. A. Góralski: *XML - i co dalej?* „Netforum" 1999 nr 7/8 s. 30-32. A. Góralski: *XML - i co dalej?* „Netforum" 1999 nr 9 s. 34-37.

wych – a latem 1997 Microsoft ukończył pracę i zaczął lansować Channel Definition Format (CDF), który był jedną z pierwszych w pełni funkcjonalnych i działających aplikacji XML-a. Wreszcie w 1998 roku wraz z pojawieniem się rekomendacji (Version 1.0) zatwierdzonej przez W3C narodził się oficjalnie standard XML[24]. W chwili obecnej powstają kolejne wersje robocze specyfikacji XLL oraz XSL (eXtensible Stylesheet Language), który jest proponowanym językiem prezentacyjnym dla dokumentów XML[25].

XML pojawił się jako wielki następca, łączący najlepsze cechy SGML-a (rozszerzalność i kompletność) i HTML-a (prostota) – języków, które odniosły ogromny sukces, lecz które nie były pozbawione wad. Jednocześnie jego przewidywane „zastąpienie" poprzednich rozwiązań jest nie do końca uprawnionym rozumowaniem, co wynika z błędnego pojmowania relacji między tymi standardami.

Wychodząc od najbardziej ogólnego SGML-a zauważmy, że zarówno HTML jak i inne języki znakowań (markup języki), należą do zupełnie odrębnej warstwy. Często mówi się, że SGML jest metajęzykiem a nie konkretnym językiem. Jest to sformułowanie nie do końca ścisłe. Jon Bosak wyjaśnia: „SGML nie jest tak abstrakcyjny jak prawdziwy metajęzyk Backus/Naur Form używany do definiowania języków programowania. Niemniej jednak na-zwanie SGML-a metajęzykiem niesie w sobie część prawdy (...) jest to język do definiowania języków znakowań"[26]. Natomiast HTML jest aplikacją SGML-a, konkretnym markup językiem, ze ściśle określoną składnią znakowania oraz kryjącym się za nim znaczeniem. Innymi aplikacjami SGML-a są wspomniane już PICS, ATA-2100 czy TEI. Każdy z języków znakowań ma trzy wspólne cechy:

– Posiada zdefiniowany zestaw tagów wraz ich znaczeniem i regułami użycia (czyli gramatyką).

– Gramatyka i znaczniki są zdefiniowane zgodnie z SGML-em.

– Jest przeznaczony do pracy z określoną kategorią dokumentów lub danych.

XML z kolei nie jest aplikacją o profilu SGML, lecz należy do tej samej warstwy, co jego język-matka. Mimo, że uproszczony i pozbawiony części możliwości SGML-a, jest to nadal metajęzyk pozwalający na stworzenie nieskończonej liczby markup języków. „Gdyby SGML był np. standardem określającym wszystkie znaki stosowane we wszystkich alfabetach świata, to XML byłby standardem obejmującym np. tylko języki, w których piszemy od lewej do prawej strony. Nie byłby natomiast konkretnym językiem.(...) Stąd (...) tekst zapisany po arabsku jest zgodny z SGML, a nie jest zgodny z XML. Natomiast tekst angielski jest zgodny z XML, stąd jest on zgodny

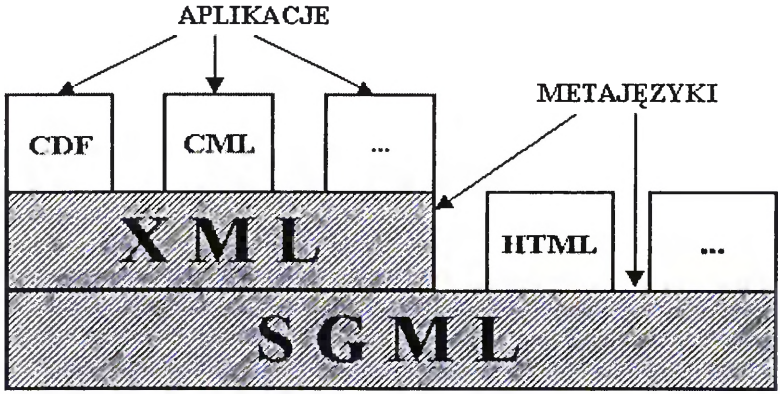
---

[24] World Wide Web Consortium: *Extensible Markup Language (XML) version 1.0. W3C recommendation*. http://www.w3.org/TR/1998/REC-xml-19980210.

[25] Patrz też: 5. Standardy stowarzyszone.

[26] J. Bosak: *Four Myths about XML*.
http://www.oasis-open.org/cover/bosak-4myths.html.

z SGML"[27]. Inaczej mówiąc XML zawiera się w SGML-u. Relacje pomiędzy poszczególnymi językami przedstawiono na rysunku 1[28].
Podsumowując można powiedzieć, że XML jest bardziej uproszczoną wersją SGML-a niż rozszerzonym HTML-em. Sami członkowie W3C XML WG określają go jako SGML Light a nie HTML++[29].



Rys. 1

---

[27] M. Jagodziński: *XML: nieprzewidziane konsekwencje*. http://www.portfolio.art.pl/xml.html.

[28] J. Marais: *An XML tutorial*. http://www.research.digital.com/SRC/personal/Johannes_Marais/talks/XMLTutorial/.

[29] L. M. Garshol: *Introduction to XML*. http://www.stud.ifi.uio.no/~lmariusg/download/xml/xml_eng.html.

Rozdział 2

CHARAKTERYSTYKA XML-A

2.1. ZAŁOŻENIA TWÓRCÓW

Zespół opracowujący język XML na początku pracy przyjął pewne istotne założenia[1], określające wymagania, jakie powinien spełniać ich finalny "produkt". Wyglądają one następująco[2]:

– **XML powinien być w prosty sposób użyteczny w Internecie**. Użytkownicy muszą móc oglądać dokumenty XML równie łatwo i szybko jak dokumenty HTML. W praktyce oznacza to próbę wymuszenia na producentach przeglądarek zapewnienia obsługi tego standardu bez konieczności zmiany dotychczasowych przyzwyczajeń użytkowników[3].

– **XML powinien wspierać różnorodne aplikacje**. Oznacza to przyjazność i użyteczność XML-a dla różnorodnych aplikacji, umożliwiających tworzenie, przeglądanie czy analizę składniową i znaczeniową odpowiednich dokumentów. Wprawdzie poprzedni punkt kładzie nacisk na zastosowania w sieci, to nie ogranicza bynajmniej jego roli tylko do tego pola.

– **XML powinien być zgodny z SGML**. Jak wspomniano wcześniej, wielu specjalistów tworzących standard wywodzi się ze środowiska SGML, stąd chęć zapewnienia kompatybilności XML-a z istniejącymi standardami, przy rozwiązywaniu stosunkowo nowych problemów związanych z publikowaniem ustrukturyzowanych dokumentów w Internecie.

---

[1] Przedstawione one zostały w rozdziale „Origins and goals" specyfikacji XML. World Wide Web Consortium, *Extensible Markup Language (XML) version 1.0. W3C recommendation*. http://www.w3.org/TR/1998/REC-xml-19980210.

[2] Tłumaczenie za: M. Jagodziński: *XML: nieprzewidziane konsekwencje*. http://www.portfolio.art.pl/xml.html.

[3] Dwaj najważniejsi uczestnicy tego fragmentu rynku tj. Microsoft (producent Explorera) i firma Netscape zareagowali dosyć ochoczo, zapowiadając szybkie wdrożenie obsługi tego standardu do swoich produktów, a także biorąc aktywny udział w dalszych pracach rozwojowych.

– **Pisanie programów przetwarzających XML powinno być łatwe**. Wśród członków W3C mówiło się, że napisanie aplikacji przetwarzającej kod XML nie powinno zająć kompetentnemu absolwentowi wyższych studiów informatycznych więcej niż dwa tygodnie[4].

– **Liczba opcjonalnych możliwości XML powinna być sprowadzona do minimum, najlepiej do zera**. Każde skomplikowanie standardu, czy to spowodowane uwzględnianiem specyficznych okoliczności czy dodawaniem cech nie zawsze koniecznych w danym przypadku, musi odbić się późniejszymi problemami z kompatybilnością np. w przypadku współdzielenia dokumentów. Starano się tego uniknąć.

– **Dokumenty zapisane w XML powinny być przejrzyste i czytelne dla człowieka**. Nawet nie posiadając specjalnej przeglądarki czy edytora XML, powinno być możliwe odczytanie lub stworzenie pliku zgodnego z tym standardem przy użyciu dowolnego oprogramowania „rozumiejącego" kod ASCII. Sens i znaczenie takiego dokumentu musi być jasne na pierwszy rzut oka.

– **Specyfikacja XML może być stworzona w krótkim czasie**. Większość prac standaryzacyjnych jest bardzo czasochłonna, lecz w tym przypadku skutecznego rozwiązania potrzebowano „tu i teraz", stąd konieczność zintensyfikowania prac nad standardem.

– **Specyfikacja XML powinna być zwięzła i ścisła**. Praktycznie wymagało to konieczności definicji XML-a przy użyciu języka EBNF (Extended Backus Naur Form) i konieczności jego dostosowania do współczesnych narzędzi i technik programistycznych[5], co ściśle wiąże się z czwartym założeniem.

– **Tworzenie dokumentów w XML powinno być proste**. Wprawdzie z czasem powstaną z pewnością zaawansowane narzędzia w znacznym stopniu wspomagające proces tworzenia nawet bardzo rozbudowanych plików XML, to napisanie jakiegokolwiek dokumentu musi być możliwe przy użyciu najprostszego edytora kodu ASCII.

– **Zwięzłość w oznakowaniu XML-em ma znikome znaczenie**. Punkt, mający swe korzenie w SGML-u. Niektóre z języków na nim opartych, cechowała tendencja do ograniczania do minimum ręcznego wprowadzania znakowania i tekstu. Miało to przyśpieszyć i ułatwić tworzenie dokumentów. W związku z możliwościami współczesnych narzędzi edycyjnych (np. skróty klawiaturowe pozwalające wprowadzać całe bloki tekstu) zrezygnowano z wymagania zwięzłości w stosunku do XML-a.

---

[4] Walsh Norman: *What is XML?* http://www.gca.org/conf/xml/xml_what.htm.
[5] Ibid.

2.2. CECHY JĘZYKA

XML pozwala nie tylko na tworzenie stron WWW – umożliwia przede wszystkim definiowanie standardów określających, jakie informacje i w jakiej kolejności powinny pojawić się w dokumencie. Ponadto w połączeniu z innymi standardami daje możliwość odseparowania treści dokumentów od sposobu jej formatowania, co znacznie ułatwia wielokrotne wykorzystanie tejże treści przez różnorodne aplikacje, w dowolnych formach prezentacyjnych. XML świetnie nadaje się do wymiany informacji między różnymi platformami sprzętowymi i programowymi bez konieczności wielowarstwowej i wielokrotnej konwersji formatów. Jego charakterystyka czyni go na tyle elastycznym, by mógł być wygodnie wykorzystywany do różnorakich celów. Do najważniejszych cech XML-a należą:

Prostota – Przejawia się zarówno w łatwości używania przez twórców dokumentów jak i programistów piszących oprogramowanie interpretujące kod XML. Dokumenty XML dzięki zastosowaniu odpowiedniego otagowania semantycznego, są w równym stopniu czytelne zarówno dla człowieka jak i maszyny. Zbudowane z wykorzystaniem mechanizmów zagnieżdżania podstawowych struktur logicznych, wprawdzie mogą rozrastać się warstwa po warstwie poprzez rozbudowywanie kolejnych elementów, to jednak mechanizm kryjący się za tymi strukturami jest stosunkowo prosty do zrozumienia i wykorzystania w sieci.

Rozszerzalność – Przejawia się w dwóch aspektach:

– XML pozwala na swobodne dodawanie i tworzenie własnych elementów w dokumencie. W przeciwieństwie do HTML-a nie istnieje tu predefiniowany, sztywny zestaw możliwych do zastosowania tagów, lecz są one tworzone przez autora w zależności od jego potrzeb.

– XML jest silnie związany z innymi standardami, które stanowią uzupełnienie, bądź czasami są wręcz niezbędne do zastosowania go w określonym celu (jak np. XLL „rozszerzający" XML o możliwość stosowania połączeń hipertekstowych, czy XSL określający sposób wyświetlenia dokumentu). Często XML stosowany jest jako baza do tworzenia innych standardów.

W rzeczywistości definiowanie formatów poszczególnych typów dokumentów (tzw. DTD[6]) jest tym, co twórcy XML-a mieli na myśli nazywając go „rozszerzalnym" (ang. *extensible*). Jest to przecież meta-język zawierający zbiór reguł określających zasady tworzenia słowników i gramatyk, które z kolei muszą być przestrzegane przez odpowiednie dokumenty zgodne z danym językiem znakowania. Norman Walsh pisze: „Język znakowania to mechanizm służący do wyodrębniania struktur w dokumencie. Specyfikacja XML określa standardowy sposób dodawania znakowania do dokumentu"[7].

[6] Patrz też: 3.8 Dokumenty dobrze uformowane a dokumenty właściwe – Definicja Typu Dokumentu.

[7] N. Walsh: *What is XML?* http://www.gca.org/conf/xml/xml_what.htm.

Warto zauważyć, że w pewnym sensie sformułowanie „dokument XML" określa coś, co nie istnieje[8]. Wszystkie dokumenty, które używają składni XML-a w rzeczywistości są przetwarzane przez aplikacje korzystające ze zbioru tagów i reguł gramatycznych zdefiniowanych przez autora dla tego konkretnego dokumentu lub zbioru dokumentów. Dla uproszczenia w dalszej części pracy, sformułowanie to będzie nadal występowało w znaczeniu dokumentu stworzonego zgodnie z ogólnymi ograniczeniami narzucanymi przez specyfikację.

Strukturalność – XML pozwala zapisać dowolną informację, pod warunkiem, że można przypisać jej pewną strukturę. Z kolei struktura dokumentu może być rozbudowywana poprzez kolejne zagnieżdżanie do dowolnego poziomu szczegółowości. Dzięki tej własności trafiający np. do przeglądarki otagowany dokument XML-owy zawiera dane, które mogą podlegać dalszej obróbce. Ponieważ dokument ma formę drzewa zagnieżdżających się elementów, można do jego przetwarzania stosować podejście obiektowe[9]. Opracowana przez W3C specyfikacja DOM[10] (Document Object Model) definiuje interfejs dostępu do zawartości elementów XML-owych z poziomu kodu aplikacji. Programy oferujące wsparcie dla DOM (np. Internet Explorer 5.0) udostępniają metody dostępu do drzewa elementów i ich zawartości. Daje to możliwość ingerencji w strukturę drzewa a także budowę nowych dokumentów od początku.

Oddzielenie danych od sposobu prezentacji – Elementy języka XML nie określają jak ma być przedstawiana zawartość dokumentu. Przypomnijmy, że HTML jest językiem znakowania złożonym z relatywnie niewielkiego zbioru standardowych tagów, którym towarzyszą mniej lub bardziej standardowe zachowania, dotyczące najczęściej sposobu prezentacji danej konstrukcji. XML to nieskończenie wielki zbiór dowolnych tagów, z którymi nie wiążą się żadne standardowe reakcje – te muszą być albo wyrażone w terminach operacyjnych odpowiedniego programu lub skryptu albo deklaratywnie określone przez arkusz stylów[11]. Oczywiście w przypadku dokumentów przeznaczonych do publikacji będzie to zwykle arkusz CSS, lecz w przypadku innych zastosowań może to być coś tak różnorodnego komponenty JavaBeans lub wyspecjalizowane protokoły przemysłowe. W ten sposób elementy mogą opisywać znaczenie tego, co zawierają, a to pozwala na bardziej „inteligentną" obsługę przez programy analizatorów składni bądź inne programy przetwarzające.

---

[8] S. St. Laurent.: *Why XML?* http://webdevelopersjournal.com/articles/why_xml.html.

[9] R. Księżyk: *XML dla o(d)pornych – cz.2.: Spójrzmy z góry na HTML!* „PC Kurier" 1999 nr 10 s. 65-70.

[10] World Wide Web Consortium, *Document Object Model (DOM) level 1 specification. Version 1.0. W3C recommendation*. http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001.

[11] J. Bosak: *Four myths about XML*. http://www.oasis-open.org/cover/bosak-4myths.html.

Poprawność – Zanim dokument XML zostanie przetworzony, odpowiedni program zwany parserem dokonuje analizy składniowej w celu sprawdzenia jego zgodności z ogólnymi regułami narzucanymi przez specyfikację, jak również gramatyką zdefiniowaną w schemacie dokumentu (jeśli takowy występuje). Parser w trakcie analizy tworzy drzewo dokumentu reprezentujące jego strukturę logiczną oraz ewentualnie dokonuje oceny prawidłowości użycia poszczególnych elementów. Jeśli złamana została któraś z reguł, analizator przerywa działanie informując o błędzie. Nie ma tu miejsca na znane z HTML-a „domyślanie się", o co chodziło twórcy dokumentu. Takie restrykcyjne podejście daje pewność, że XML sam z siebie nie spowoduje błędu czy zawieszenia się aplikacji, gdyż jeśli przetwarzany dokument będzie niepoprawny, zostanie on po prostu zignorowany. Poza tym zmusza to autorów do wyzbycia się nawyków dowolności kodowania, jaka cechowała tworzenie stron WWW z użyciem HTML-a, który wespół z przeglądarkami, dopuszczał istnienie dokumentów jawnie łamiących reguły specyfikacji.

Otwartość – XML jest standardem otwartym, dostępnym bez ograniczeń w sieci. Członkowie W3C podczas jego opracowywania publikowali kolejne wersje robocze, umożliwiając na bieżąco śledzenie postępów ich pracy. Każdy mógł i nadal może zgłaszać swoje uwagi na temat XML-a. Zaimplementowanie obsługi tego rozwiązania w jakimkolwiek oprogramowaniu nie wiąże się z żadnymi ograniczeniami. Podejście takie Konsorcjum WWW tłumaczy w następujący sposób: „Dane krążące w sieci pomiędzy różnymi platformami sprzętowymi i programowymi, muszą być możliwe do odczytania i użycia przez je wszystkie. Informacja publiczna nie może być w żaden sposób ograniczana do jednego rozwiązania czy producenta, a kontrola nad formatem danych nie powinna spoczywać wyłącznie w jednych rękach. Równie ważną rzeczą jest, aby dane zgromadzone w Internecie były dalej przetwarzane i wykorzystywane jak najmniejszym kosztem wysiłku i czasu.[12]".

2.3. OBSZARY ZASTOSOWAŃ

Ustanowienie XML-a standardem w Internecie zunifikuje procesy przechowywania i przetwarzania danych umożliwiając pracę na odmiennych systemach i współdzielenie jej wyników. W założeniach twórców, XML ma stanowić rozwiązanie kilku istotnych problemów, z którymi dotychczasowe technologie nie mogły sobie poradzić. Mówi się o czterech obszarach, gdzie aplikacje wspierające XML-a znajdą zastosowanie[13]:

[12] World Wide Web Consortium's XML Special Interest Group: *Frequently asked questions about the Extensible Markup Language*. http://www.ucc.ie/xml/.

[13] Podział i przykłady za: J. Bosak: *XML, Java, and the future of the Web*. http://sunsite.unc.edu/pub/sun-info/standards/xml/why/XML-apps.htm.

1. Aplikacje wymagające klienta pośredniczącego w wymianie danych pomiędzy dwoma heterogenicznymi bazami danych. Za przykład niech posłuży amerykańska sieć agencji ochrony zdrowia, której system informacyjny jest punktem krytycznym, gdyż musi gromadzić różnorakie dane (najczęściej pierwotnie występujące w postaci drukowanej) z historiami chorób, rachunkami od różnych lekarzy, szpitali, aptek itp. Dodanie pacjenta do takiej agencji wiąże się z koniecznością konwersji tych informacji do postaci rekordów bazy danych. Szpitale proponują następujący schemat komunikacji z poszczególnymi agencjami:

– połączenie przez sieć ze stroną szpitala
– autoryzacja użytkownika poprzez hasło
– dostęp do danych pacjenta z użyciem przeglądarki
– wydruk otrzymanych danych
– „ręczny" wybór potrzebnych danych z wydruku

Użycie XML-a w pełni zautomatyzuje ten proces zastępując najmniej efektywne jego etapy w następujący sposób:

– dostęp do danych konkretnego pacjenta (reprezentowanego przez odpowiednią ikonę) z użyciem przeglądarki
– „przeciągnięcie" folderu do wewnętrznej aplikacji agencji
– automatyczne uzupełnienie bazy agencji

HTML nie jest odpowiednim formatem do wykorzystania w powyższym schemacie ponieważ:

– zbiór tagów jest zbyt ubogi, aby mógł oddać bogactwo różnych pól występujących w np.: historii choroby pacjenta
– nie można przedstawić złożonej struktury tych dokumentów
– nie posiada żadnego mechanizmu sprawdzenia strukturalnej poprawności danych przed ich importem do bazy

XML jest uniwersalnym formatem, który posłuży jako jeden format wyjściowy dla wszystkich systemów eksportujących a jednocześnie będzie ujednoliconym formatem wejściowym dla systemów importujących. Aplikacja ma tu spełniać rolę mediacyjną w wymianie danych pomiędzy różnymi systemami, czyli być oprogramowaniem typu middleware. XML można nazwać w tym wypadku językiem pośredniczącym (ang. *hub language*), zapewniającym właściwą komunikację pomiędzy heterogenicznymi bazami danych.

Duże nadzieje wiąże się z XML-em przy niwelowaniu zasadniczych różnic architektonicznych dzielących różne organizacje stosujące systemy EDI[14] (elektroniczna wymiana danych; *ang. Electronic Data Interchange*). Jedyną rzeczą, jaką musiałyby one uzgodnić to zbiór standardowych tagów wykorzystywanych w wymienianych dokumentach, oraz stworzyć kod służą-

[14] J. Muszyński: *Nowe projekty standardów*. „Net World" 1999 nr 6 s.71-72.

cy do ich transformacji do formatów zgodnych z poszczególnymi systemami back-end. Innymi słowy oczekuje się, że XML zrobi dla danych to, co Java uczyniła z programami tj. uniezależni je całkowicie zarówno od platformy sprzętowej jak i programowej[15]. Rozwiązanie takie wydaje się szczególnie odpowiadać rozwijającym się gwałtownie transakcjom online i nabierającemu coraz większego znaczenia e-biznesowi.

2. Odciążenie serwerów dokonywane przez skierowanie części zadań na stronę klienta. Przykładem jest wspomniane już rozwiązanie stosowane w przemyśle półprzewodnikowym (PICS). Ponieważ każda z firm z tej gałęzi posiada kilka terabajtów danych technicznych, opracowano specyfikację efektywnej ich wymiany. Ideą, którą wcielono w życie, było nie tylko sprawienie, że dokumentacja ta (silnie sparametryzowany strumień danych) mogła być odczytywana przez odpowiednie aplikacje i przedstawiana w „czytelnej" postaci, ale także wspomagać miała bezpośrednio procesy projektowania. To rozwiązanie zdawało się zapowiadać pojawienie się Javy: zbliżały się bowiem czasy, kiedy ze strony WWW inżynierowie mogli ściągać nie tylko dane techniczne np. o określonych wyspecjalizowanych obwodach, ale także aplety Javy, które pozwalają te obwody modelować.

Zastosowanie XML-a i w tym przypadku oferuje coś, czego nie mogły dać istniejące rozwiązania:

– rozszerzony zbiór tagów niedostępnych w HTML-u

– prezentacja danych niezależna od platformy

– długotrwałe procesy obliczeniowe (np. modelowanie obwodów), które szczególnie dotykały serwery zmuszane do wysiłku przez wielu użytkowników zadających im podobne, skomplikowane zadania, zmieniły się w krótką „rozmowę" z klientami przesyłając im odpowiednie dane (aplety Javy są właśnie aplikacjami wykonywanymi po stronie klienta).

Połączenie technologii XML z Javą pozwoli odciążyć serwery, zastępując zużyte i nieefektywne rozwiązanie bazujące na współpracy HTML-a ze skryptami CGI[16] (ang. *Common Gateway Interface*) zapewniające programową interakcję między końcowym użytkownikiem oglądającym stronę WWW a serwerem. W efekcie Internet będzie mniej zatłoczony, a więc szybszy. Przykład: Załóżmy, że poszukujemy informacji o lotach z Londynu do Nowego Yorku[17]. Otrzymana lista z pewnością będzie zawierała kilkadziesiąt pozycji. Aby ją zawęzić do np. konkretnej linii lotniczej, godzin odlotu czy ceny biletu należy wysłać kolejne żądanie do serwera, określając

[15] J. Bosak: *Four myths about XML*.
http://www.oasis-open.org/cover/bosak-4myths.html.

[16] Mechanizm pozwalający na uruchamianie programów pracujących w systemie operacyjnym serwera, przekazywanie im danych wejściowych oraz odbieranie wyników ich pracy. P. Mielecki: *Okienko na świat*. „Chip" 2000 nr 3 s. 169.

[17] J. Bosak, T. Bray: *XML and the Second-Generation Web*.
http://www.sciam.com/1999/0599issue/0599bosak.html.

kryteria selekcji i czekać na odpowiedź. Natomiast gdyby wspomniana lista była napisana w XML-u, to już za pierwszym razem mógłby jej towarzyszyć niewielki program Javy, pozwalający dowolnie manipulować i sortować otrzymane dane zupełnie niezależnie od serwera.

Daje to możliwość zastąpienia dotychczasowych, zwykle statycznych dokumentów HTML, stronami będącymi aplikacjami działającymi po stronie klienta. Todd Freter nazywa je „weblikacjami" (*ang. weblication*)[18]. Na konferencji poświęconej XML-owi, która odbyła się z Seattle 23-go marca 1998 roku Adam Bosworth z Microsoftu dokonał jednej z pierwszych demonstracji takiego rozwiązania. Była to internetowa aukcja dzieł sztuki, gdzie obserwatorzy mogli oglądać wybrane przez siebie eksponaty, włączać się do licytacji, zgłaszać kolejne kwoty obserwując jednocześnie działania innych licytujących. Podczas całej aukcji komunikacja z serwerem była ograniczona do minimum[19]. XML jest więc jednym z rozwiązań, które ma prowadzić do przekształcenia Internetu z kanału informacyjnego w uniwersalną platformę przetwarzającą dane[20].

3. Pozostawienie decyzji, co do sposobu prezentacji dokumentów końcowemu użytkownikowi. XML umożliwi przeglądanie dokumentów lub ich wybranych części w różnych formatach bez konieczności ściągania dodatkowych danych z serwera. Aplikacje wykorzystujące tą własność, będą mogły dynamicznie generować tzw. spisy zawartości (*ang. table of contents*) dokumentu. Już w chwili obecnej możliwe jest stworzenie obiektowej bazy danych i zaprezentowanie użytkownikowi spisu treści opisującego ten zbiór danych, które z kolei mogą być rozwinięte na jego życzenie (najczęściej poprzez kliknięcie) ukazując głębsze poziomy hierarchicznej struktury prezentowanego dokumentu. Niestety owo rozwijanie i zwijanie poszczególnych elementów ze spisu zawartości jest dosyć powolne, bowiem wymaga ciągłego kontaktu z serwerem.

Natomiast w środowisku XML spisy treści będą generowane dynamicznie i przesyłane do przeglądarki klienta, gdzie przy użyciu np. apletu Javy zostanie określony dokładny sposób wyświetlenia dokumentu a możliwe to będzie dzięki budowie każdego dokumentu z zagnieżdżonych elementów tworzących strukturę hierarchiczną.

Kluczową rolę będzie tu odgrywał także standard XSL[21], opisujący reguły stosowania arkuszy stylów w celu prezentacji dokumentów XML w różnych formach, w zależności od preferencji odbiorcy. Autorzy tekstów publikowanych w sieci nie będą już musieli dbać o wygląd swoich dokumentów – to

---

[18] T. Freter: *Beyond text and graphics: XML makes Web pages function like applications.* http://www.sun.com/980414/xml/.

[19] Przykład wraz z komentarzem dostępny pod adresem: http://www.microsoft.com/xml/parser/auction/overview.htm.

[20] T. Freter: *Beyond text and graphics: XML makes Web pages function like applications.* http://www.sun.com/980414/xml/.

[21] Patrz też: 5. Standardy stowarzyszone.

zadanie przypadnie teraz projektantom arkuszy. Niezależność warstwy prezentacyjnej od przedstawianych danych pozwoli także zrewolucjonizować pracę wydawców, dając im możliwość publikowania raz napisanego tekstu, zarówno w dowolnej formie elektronicznej, jak i drukowanej. Jon Bosak idzie jeszcze dalej, określając XML jako format publikacji niezależny od medium[22]: „XML i XSL są w stanie zastąpić wszystkie istniejące formaty wykorzystywane zarówno przez procesory tekstu jak i oprogramowanie DTP. W zamian dostaniemy jeden, międzynarodowy format o niemal nieograniczonym polu zastosowań, zarówno wśród publikacji drukowanych jak i prezentowanych w sieci, który będzie w pełni kompatybilny ze wszystkimi istniejącymi programami i platformami sprzętowymi". Nate Zelnick dodaje: „(...) dokumenty będą możliwe do odczytania nawet, jeśli przetwarzające je programy zmienią się zupełnie: Dokumenty mogą żyć wiecznie"[23].

4. Aplikacje, w których „inteligentny" agent stara się dostarczać informacje w zależności od potrzeb użytkownika. Załóżmy, że istnieje 500 programów WEBTV (telewizji internetowej) scharakteryzowanych w ujednolicony sposób (tematyka, język, wiek, dla którego są przeznaczone, data powstania itp.) Każdy z potencjalnych odbiorców telewizji także jest opisany konkretnymi parametrami (wykształcenie, wiek, hobby itp.). Zadaniem „inteligentnych" aplikacji będzie automatyczny dobór odpowiednich programów dla konkretnego odbiorcy. Na początku tworzenia takich aplikacji trzeba będzie z pewnością określać wiele parametrów jawnie, ale z czasem powinny pojawić się automaty, które same będą rozpoznawały preferencje użytkownika. Wprawdzie programy takie, to wciąż dość odległa perspektywa, ale podstawowe założenie, jakie musi być spełnione do jej realizacji tj. istnienie standardu opisu zarówno programów jak i odbiorców jest zapewnione przez XML-a.

Ponadto pewną dozą „inteligencji" może zostać wzbogacone przeszukiwanie sieci opartej na dokumentach oznakowanych XML-em. W chwili obecnej wyszukiwarki dostarczają w odpowiedzi na zapytania użytkowników wiele nierelewantnych dokumentów. Wynika to chociażby z nieodróżniania kontekstu, w jakim występuje informacja np.: jeśli szukamy wiadomości o nowym modelu Mercedesa to otrzymamy spis stron nie tylko o samochodach, ale też np. biografię kogoś, kto ma na imię Mercedes. Stosując tagi XML można uniknąć tego rodzaju problemów.

Podsumowując: XML oferując uniwersalny format danych, uzupełni listę otwartych standardów WWW, do której należą:

– TCP/IP (Transfer Control Protocol / Internet Protocol) – uniwersalny protokół komunikacyjny, wykorzystywany do łączenia z Internetem przez szeroką gamę urządzeń: od komputerów klasy mainframe, przez komputery osobiste, laptopy aż po telefony komórkowe.

[22] J. Bosak: *Four myths about XML*. http://www.oasis-open.org/cover/bosak-4myths.html.

[23] N. Zelnick: *Why XML?* http://webdeveloper.com/xml/xml_050198.html.

– HTML – uniwersalny język prezentacyjny pozwalający publikować dokumenty w sieci.

– Java – język programowania niezależny od platformy sprzętowej, odciążający serwery i pozwalający wykonywać programy po stronie klienta.

2.4. KONSEKWENCJE DLA SIECI

Internet drugiej generacji[24], oparty na XML-u będzie szybszy, przyjaźniejszy i efektywniejszy w działaniu. Z drugiej strony wszyscy biorący aktywny udział w jego tworzeniu mogą odczuć zwiększone wymagania wobec siebie. Na przykład projektanci stron WWW, dotychczas zaangażowani tylko w pisanie tekstów i łączenie ich z grafiką, będą musieli nabrać biegłości w konstruowaniu wielowarstwowych, powiązanych ze sobą systemów schematów dokumentów, struktur drzewiastych, rozbudowanych mechanizmów hiperłączy, metadanych i arkuszy stylów. Pociąga to za sobą konieczność zmiany dotychczasowych przyzwyczajeń i innego podejścia do zagadnienia publikowania w sieci[25]:

– dostarczanie informacji w sposób zgodny z XML-em.

– zamiana tradycyjnego podejścia do informacji na obiektowy, czyli właściwy XML-owi.

– dobór odpowiednich narzędzi do tworzenia, dostarczania i zarządzania informacją opartą na XML-u.

Rozpowszechnienie standardu w wystarczającym stopniu również będzie wymagało podjęcia odpowiednich wysiłków, które powinny skupiać się na:

– konwersji istniejących informacji do XML-a i innych ustrukturyzowanych formatów, które uczynią te informacje w pełni wartościowymi.

– rozwijaniu nowych struktur opartych na XML-u.

– zarządzaniu istniejącymi zbiorami informacji bazującymi na XML-u.

Oczywiście pojawienie się XML-a nie spowoduje całkowitej dewaluacji istniejących standardów. SGML będzie wciąż najbardziej kompleksowym i najpełniejszym rozwiązaniem w dużych, kosztownych systemach dokumentacyjnych i wielkich repozytoriach informacji. W tej chwili istnieją już narzędzia potrafiące „w locie" przeprowadzić konwersję z formatu zgodnego z SGML-em do XML-a lub HTML-a[26], co z kolei umożliwia publikację takiej informacji w sieci.

---

[24] J. Bosak, T. Bray: *XML and the Second-Generation Web.* http://www.sciam.com/1999/0599issue/0599bosak.html.

[25] T. Freter: *XML: Mastering information on the Web.* http://www.sun.com/980310/xml/.

[26] T. Freter: *Beyond text and graphics: XML makes Web pages function like applications.* http://www.sun.com/980414/xml/.

Wciąż nie wiadomo, jaka będzie w przyszłości rola HTML-a. Istnieją cztery warianty[27]:

– HTML albo XML. HTML jest wciąż najłatwiejszym sposobem do umieszczania krótkich, nieskomplikowanych dokumentów w sieci. Powszechność użycia i przeciętna znajomość wśród internautów przemawia za jego zachowaniem. Z drugiej strony wiele organizacji podejmuje działania mające na celu, jeśli nie jego wyeliminowanie, to co najmniej poważną redefinicję np.: producenci telefonów komórkowych opracowali uproszczony HTML (nazwany Compact HTML) pozbawiony własności istotnych tylko dla pełnoekranowych aplikacji, takich jak przeglądarki.

– HTML i XML. Poza wspomnianą już powszechnością i łatwością tworzenia dokumentów, HTML – mimo wszystkich swoich wad – stał się najpopularniejszym językiem wymiany myśli w sieci, głęboko wpisując się w jej krajobraz jako najlepszy środek komunikacji i jego supremacja będzie niezwykle trudna do przezwyciężenia nawet przez tak wszechstronny standard, jakim jest XML.

– XML w HTML-u. Propozycja ta zakłada dołączanie do dokumentów HTML pewnych obiektów zdefiniowanych w XML-u. Pomysł wpisywania tzw. wysp[28] XML w kod HTML pozwoli twórcom stron na wykorzystanie pewnych zaawansowanych własności tego języka niedostępnych w HTML-u, a poza tym będzie to stanowiło dobre pole do eksperymentów z nowym standardem w dobrze już znanym środowisku. To rozwiązanie zastosował Microsoft w pakiecie Office 2000, gdzie natywnym formatem dokumentów uczyniono HTML z „wyspami" opisującymi najbardziej newralgiczne informacje poszczególnych aplikacji wchodzących w skład pakietu.

– HTML jako XML. Idea tego pomysłu sprowadza się do redefinicji HTML-a do postaci aplikacji XML. Oznaczałoby to konieczność tworzenia co najmniej dobrze uformowanych[29] dokumentów HTML. Wprawdzie XML nie jest kompatybilny „wstecz" z istniejącymi dokumentami hipertekstowymi, to są pewne narzędzia pozwalające na prostą konwersję tych dokumentów do XML-a pod warunkiem, że są one w pełni zgodne co najmniej z wersją 4.0 specyfikacji W3C[30]. Niniejsze rozwiązanie cieszy się największym poparciem W3C, które pod koniec stycznia 2000 nadało status rekomendacji specyfikacji języka XHTML[31] (Extenisble HTML), będącego pomostem łączącym dominujący obecnie w sieci HTML z korzyściami, jakie daje XML. XHTML został utworzony poprzez przepisanie HTML-a

---

[27] T. Freter: *XML: It's the future of HTML*. http://www.sun.com/980602/xml/.

[28] Ibid.

[29] Patrz też: 3.8 Dokumenty dobrze uformowane a dokumenty właściwe – Definicja Typu Dokumentu.

[30] A. Rifkin: *A look at XML*. http://www.webdeveloper.com/xml/xml_a look_at_xml.html.

[31] World Wide Web Consortium: *XHTML™ 1.0: the Extensible HyperText Markup Language – a reformulation of HTML 4 in XML 1.0. W3C recommendation.* http://www.w3.org/TR/2000/REC-xhtml1-20000126.

4.0 w formie aplikacji XML, co pozwala używać do jego przetwarzania oprogramowania już istniejącego, jak i korzystać z „programowej niezależności" nowego formatu. Dalsze prace mają się skupić na modularyzacji języka w celu wyodrębnienia jego podzbiorów, które będą wykorzystywane przez różne urządzenia odbiorcze (palmtopy, telefony komórkowe itp.).

Ogólne prognozy, co do roli XML-a w zrewolucjonizowanym Internecie przedstawia Gartner Group w raporcie Strategic Analysis Report z września 1998[32]: „ (...) XML i związane z nim standardy są kluczowymi technologiami pozwalającymi na to, żeby dokumenty stały się interaktywnymi przekaźnikami informacji między ludźmi i maszynami (...). W roku 2000, przetwarzalność dokumentów, możliwa dzięki ich otagowaniu, będzie najpoważniejszą tendencją w wydawnictwach korporacyjnych (prawdopodobieństwo 0,8), pozwalając przedsiębiorstwom na zamienianie dokumentów będących statycznymi pojemnikami danych w potężne aplikacje (...). Użycie standardów XML rozszerzy otagowanie dokumentów i w pierwszej połowie roku 2000, przewyższy użycie HTML w zastosowaniach wydawniczych (prawdopodobieństwo 0,9)".

---

[32] R. Księżyk: *XML, czyli jak posprzątać Internet.* http://www.fuw.edu.pl/~ksiezyk/a_xml1_pl.html.

Rozdział 3

DOKUMENTY XML – ZASADY TWORZENIA

Mimo tego, że XML umożliwia stosowanie dowolnych tagów to oczywiście istnieją pewne ograniczenia w tworzeniu prawidłowych dokumentów XML, które narzuca opisywana dalej specyfikacja. Jak już wspomniano, XML jest raczej narzędziem do generowania języków znakowań niż konkretnym markup językiem, dlatego więc specyfikacja nie podaje zbioru predefiniowanych, konkretnych tagów, lecz określa metodologię ich tworzenia. Właśnie owe znaczniki w połączeniu z właściwą zawartością stanowią dokument XML[1].

3.1. STRUKTURA DOKUMENTU

Aby dokument został właściwie przetworzony musi on być **dobrze uformowany** (ang. ***well-formed***) tj. nie może łamać żadnej z zasad określonych przez specyfikację, a więc mieć odpowiednią fizyczną i logiczną strukturę. Stworzenie dokumentu dobrze uformowanego jest minimalnym wymaganiem, które musi być spełnione, aby był on zrozumiały dla komputera nie tracąc nic ze swojej czytelności dla człowieka[2].

---

[1] Termin „dokument" w tym znaczeniu może być nieco mylący, gdyż znakowanie XML wprawdzie może być zawarte w konkretnym pliku, lecz równie dobrze może przybierać formę przesyłanego strumienia danych, zbioru rekordów bazy danych, czy też dynamicznie generowanych komunikatów między współpracującymi aplikacjami. Uściślając, mówiąc dokument XML należy myśleć o obiekcie z danymi (*ang. data object*).

[2] Patrz też: 2.1 Założenia twórców (punkt 6).

Każdy dokument ma swoją fizyczną i logiczną strukturę. Fizycznie dokument jest zbudowany ze zbioru encji[3] (jednostek przechowujących pewną zawartość) o niepowtarzalnych nazwach, z których każda encja podlegająca parsowaniu[4] spełnia wymogi dokumentu dobrze uformowanego.

Logicznie dokument składa się z:

- deklaracji
- elementów
- komentarzy
- odniesień do znaków (*ang. character references*)
- instrukcji przetwarzania (*ang. processing instructions*)

3.2. DANE A ZNAKOWANIE

Na tekst dokumentu XML składają się **dane** (ang. ***character data***) i **znakowanie** (*ang.* ***markup***). Przez dane rozumiemy najczęściej dane znakowe (litery, cyfry, interpunkcja, itp.) czyli wszystko to, co definiują standardy Unicode[5] i ISO 10646[6] z wyjątkiem znaków specjalnych oraz dane binarne, natomiast markupem są wszelkie elementy znakowania XML tj. tagi, komentarze, instrukcje przetwarzania itd. Inaczej mówiąc tekst, który nie stanowi markupu uznawany jest za dane stanowiące zawartość dokumentu XML. Znaki specjalne to znak and (**&**), oraz lewy ostry nawias (**<**), które mogą wystąpić w literalnej formie tylko jako elementy znakowania, w komentarzu, instrukcji przetwarzania, sekcji CDATA lub jako dosłowna wartość wewnętrznej encji. Analogicznie prawy ostry nawias (**>**) nie może występować w łańcuchu znakowym „**]]>**" gdy ten nie oznacza końca sekcji CDATA. Wszystkie ww. znaki oraz znaki pojedynczego (**'**) i podwójnego cudzysłowu (**„**) występują w postaci predefiniowanych encji i mogą być wywoływane w dwojaki sposób (tab. 1). Oznacza to, że np. chcąc aby przeglądarka wyświetliła dane, które mają postać „<Data>" należy zapisać je jako „***<Data>***".

---

[3] Słowo „encja" (ang. entity) pochodzi od francuskiego „entité", którego z kolei teutońskim odpowiednikiem jest „thing" (rzecz). Terminem synonimicznym do encji jest termin „obiekt". M. Bryan: *An introduction to the Extensible Markup Language (XML)*. http://www.personal.u-net.com/~sgml/xmlintro.htm. Patrz też: 3.16 Encje.

[4] Encja, której zawartość jest tekstem stanowiącym integralną część dokumentu XML.

[5] Unicode Consortium: *The Unicode Standard. Version 2.0*. Reading Addison-Wesley Developers Press 1996.

[6] International Organization for Standardization: *ISO/IEC 10646-1993. Information technology. Universal Multiple-Octet Coded Character Set (UCS). Part 1: Architecture and Basic Multilingual Plane*, Geneva, ISO 1993.

Tabela 1

Znak	Odniesienie do znaku	
>	>	>
<	<	<
&	&	&
‘	'	'
”	"	"

3.3. SEKCJE CDATA

Są to specjalne bloki, w których wszystkie tagi i odniesienia do encji są traktowane jak inne dane znakowe, a co za tym idzie ignorowane przez procesor XML. Składnia:

<![CDATA[zawartość sekcji]]>

Konstrukcja ta została wprowadzona jako udogodnienie w przypadku stosowania dużych bloków tekstu zawierających znaki specjalne, które mają być traktowane jak zwykły tekst. Pozwala to uniknąć ciągłego używania odniesień do encji.

Łańcuch znakowy "]]>" w sekcji CDATA oznaczający jej koniec, jako jedyny jej element jest rozpoznawany jako markup, stąd znaki (<) i (&) mogą tu występować w swej literalnej postaci. Sekcje CDATA nie mogą być zagnieżdżane.

Przykład 3.3a:

```
<EXAMPLE>
    <![CDATA[ <DOCUMENT>
    <Name> Grzegorz Zieliński </NAME>
    </DOCUMENT> ]]>
</EXAMPLE>
```

Gdy parser XML napotka fragment kodu jak przedstawiony wyżej, to nie zakomunikuje błędu wynikającego z niezgodności wielkości liter

w otwierającym i zamykającym tagu <NAME>, co więcej w ogóle nie potraktuje zawartości sekcji CDATA jako tagi.

3.4. KOMENTARZE

O ile sekcje CDATA należą do danych znakowych i jako takie są traktowane przez aplikację XML, to komentarze są w całości ignorowane w procesie przetwarzania.

Składnia jest identyczna jak w HTML-u:

<!--To jest komentarz... -->

Ograniczenia jakie występują w stosowaniu komentarzy:
– nie wolno używać w komentarzach łańcuchów "-" ani "--"
– nie wolno używać komentarzy wewnątrz tagów
– nie wolno używać wewnątrz deklaracji encji
– nie wolno używać przed deklaracją XML
– nie mogą być zagnieżdżane

3.5. INSTRUKCJE PRZETWARZANIA

Instrukcja przetwarzania to zbiór informacji przeznaczonych dla aplikacji XML. Nie są one parsowane, lecz trafiają bezpośrednio do aplikacji, a ta z kolei może je przesłać dalej, bądź przetworzyć sama. Wszystkie instrukcje mają format:

<?NAZWA_APLIKACJI_DOCELOWEJ INSTRUKCJE_DLA_APLIKACJI?>

Przykład:

<?JAVA_OBJECT JAR_FILE = "/java/myjar.jar"?>

Jako nazwa aplikacji docelowej nie może wystąpić w jakiejkolwiek formie łańcuch znaków „xml"[7] jako zarezerwowany dla kolejnych wersji specyfikacji.

[7] Zasada zabraniająca używania łańcucha znakowego „xml" dotyczy tworzenia nazw wszystkich elementów składowych znakowania XML.

3.6. DEKLARACJA XML

Element ten pochodzi bezpośrednio z SGML-a i jest przykładem instrukcji przetwarzania. Dokument XML może, choć nie musi, rozpoczynać się od deklaracji typu, oznaczającej zgodność ze specyfikacją konkretnej aplikacji XML. Choć jest to jedynie element zalecany, to jego brak może spowodować błędną identyfikację dokumentu, a co za tym idzie problemy interpretacyjne. Deklaracja ta umożliwia aplikacjom przetwarzającym (przeglądarki, edytory, procesory tekstu itp.) na określenie sposobu przetwarzania i definiuje dokument jako XML. Przykład 3.6a:

```
<?xml version="1.0" encoding="ISO-8859-2" standalone="yes"?>
<document>
Ten dokument nie ma DTD i zawiera dane znakowe zakodowane
zgodnie z normą ISO
</document>
```

W deklaracji zawierają się informacje o:

– języku znakowania (tu: xml).

– wersji języka (1.0).

– zastosowanego sposobu kodowania (ISO-8859-2); brak tego parametru oznacza kodowanie zgodne z UTF-8.

– obecności zewnętrznej względem dokumentu części lub całości definicji znakowania tzw. DTD[8] (parametr standalone); użyta tu wartość „yes" oznacza, że takowej nie ma.

3.7. ELEMENTY (TAGI I ATRYBUTY)

Elementy są podstawowymi jednostkami logicznymi tworzącymi dokument XML. Składają się z **tagów: otwierającego i zamykającego** oraz wszystkiego, co jest pomiędzy nimi (zawartość elementu). Tagiem jest wszystko znajdujące się pomiędzy znakiem lewego (<) a prawego ostrego nawiasu (>), a co nie jest komentarzem ani sekcją CDATA. Każdy element ma swoją unikatową nazwę, która określa jego **typ** (tzw. **identyfikator ogólny** ang. **generic identifier**), a także może posiadać atrybuty o zdefiniowanych nazwach i wartościach. Elementy mogą być wielopoziomowo zagnieżdżane, co oznacza, że jeden element może być zawartością innego. Schemat:

[8] Patrz też: 3.8. Dokumenty dobrze uformowane a dokumenty właściwe – Definicja typu dokumentu.

<TAG atrybut1="wartość" atrybutN= "wartość"> zawartość elementu </TAG>

Przykład 3.7a:

<gracz nr="3" pozycja="obrońca">Paolo Maldini</gracz>

W każdym dokumencie XML musi występować dokładnie jeden element nazywany **elementem podstawowym** (ang. ***root element*** *lub* ***document element***), w którym zawierają się wszystkie pozostałe elementy. Jest on niepowtarzalny w danym dokumencie, synonimiczny z jego typem i zwykle opisuje funkcję lub charakter dokumentu[9]. Umieszczany jest on za deklaracją typu dokumentu.

Reguły, jakich należy przestrzegać przy tworzeniu elementów:

– Nazwy tagów i atrybutów nie mogą zaczynać się od łańcucha znaków „xml".

– XML, w przeciwieństwie np. do HTML-a, jest językiem rozróżniającym wielkie litery od małych (ang. ***case sensitive***) stąd np.: <NAME> i <name> to dwa zupełnie różne tagi.

– Nazwa tagu może zaczynać się od litery, znaku podkreślenia (_), lub dwukropka (:)[10], po których występuje kombinacja liter, cyfr, kropek (.), dwukropków (:), podkreśleń (_), myślników (-) ale nie tzw. białych miejsc[11] (ang. ***white space***); specyfikacja nie określa maksymalnej długości nazw, lecz pewne ograniczenia mogą nieść w sobie poszczególne aplikacje XML.

– Każdemu tagowi otwierającemu musi odpowiadać tag zamykający z powtórzoną nazwą poprzedzoną znakiem (/) – patrz schemat wyżej; w szczególnych wypadkach przy tzw. pustych elementach (ang. ***empty element***), czyli elementach bez zawartości domknięcie tagu można zawrzeć w tagu otwierającym dodając znak (/) po nazwie i ewentualnych parametrach np. ***<IMAGE file="landscape.gif"/>.***

– Konieczność prawidłowego zagnieżdżania; element podstawowy nie może zawierać się w żadnym innym elemencie a każdy inny element jest poprawnie zagnieżdżony, jeśli zarówno tag otwierający jak i zamykający należą do zawartości tego samego elementu; formalnie: dla każdego elementu C, który nie jest elementem podstawowym, istnieje element P, taki że C należy do zawartości P, ale nie należy do zawartości innego elementu, który zawiera się w P. P nazywamy wtedy **rodzicem** (ang. ***parent***) C a C **potomkiem** (ang. ***child***) P.

---

[9] Ta zasada nie jest wymagana, ale taka jest konwencja.

[10] Używanie dwukropka na początku nazwy tagu wprawdzie jest dozwolone, ale w pewnych wypadkach może to powodować problemy podczas przetwarzania i raczej należy tego unikać. Patrz też: 5.1. Przestrzenie nazw XML.

[11] Należą do nich znaki spacji, powrotu karetki, przesuwu o wiersz i tabulacji. Patrz też: 3.14. Atrybuty specjalne.

– Atrybuty nie mogą powtarzać się w obrębie jednego tagu a ich wartość musi być umieszczona między znakami pojedynczego (') lub podwójnego cudzysłowu (") występujących po znaku równości (=).

– Nazw atrybutów tyczą te same reguły co nazw tagów, natomiast wartości[12] mogą zawierać białe miejsca, znaki interpunkcyjne lub odwołania do encji (z wyjątkiem odwołań do encji zewnętrznych względem dokumentu) pod warunkiem, że w tejże nie występuje znak (<) w swej literalnej postaci.

3.8. DOKUMENTY DOBRZE UFORMOWANE A DOKUMENTY WŁAŚCIWE – DEFINICJA TYPU DOKUMENTU

Przedstawione do tej pory zasady tworzenia dokumentów XML odnoszą się do dokumentów dobrze uformowanych, czyli podstawowych tworów tego języka, które są zgodne z narzucanymi przez niego regułami składniowymi. Dokument zgodny z XML musi być well-formed w momencie gdy jest przetwarzany. Jeśli nie jest, nie jest zgodny z XML. Aby lepiej zrozumieć, czym jest dokument well-formed posłużmy się analogią, w której XML jest pew-nym alfabetem zaś dokumenty XML są utworami zapisanymi zgodnie z regułami tego alfabetu[13]. Dokumenty te mogą być przetworzone przez programy obsługujące XML, jeśli reguły te są spełnione. Gdy z kolei chcemy, aby dokument był przydatny do pewnego, ściśle określonego celu musimy określić zbiór precyzyjniejszych reguł, które musi spełnić dokument. Powiedzmy że XML definiuje alfabet polski – każdy tekst napisany po polsku będzie wtedy dokumentem dobrze uformowanym. Natomiast jeśli chcemy pisać np.: faktury, zeznania podatkowe czy sonety musimy narzucić więcej reguł na strukturę dokumentu.

Robi się to tworząc **właściwe dokumenty** (ang. ***valid document***) XML, których logiczna struktura i sposób przechowywania danych jest dokładnie określony stworzonym specjalnie dla danego rodzaju utworów sposobem znakowania opisanym w **deklaracji typu dokumentu** (ang. ***document type declaration***) a nazywanym **definicją typu dokumentu** (ang. ***document type definition***) lub w skrócie **DTD**.

DTD definiuje gramatykę i słownictwo języka znakowania i jest sercem idei wszystkich markup systemów. Można powiedzieć, że jest to zestaw

[12] Uwaga: wszystkie wartości atrybutów są domyślnie traktowane jako dane typu string, stąd konieczność ich konwersji na wartości typu numerycznego poza środowiskiem XML, w przypadku gdy chcemy wykonywać na nich operacje arytmetyczne.

[13] M. Jagodziński, *XML: nieprzewidziane konsekwencje*. http://www.portfolio.art.pl/xml.html.

reguł, którym podlega dokument, a które oprogramowanie musi odczytać przed przetworzeniem dokumentu. Z zasady reguły te odnoszą się do nazwy, zawartości każdego elementu oraz kontekstu, w jakim może lub musi występować. Inaczej rzecz ujmując, DTD zawiera to wszystko, co powinien wiedzieć parser aby prawidłowo zinterpretować dobrze uformowany dokument XML. Złożoność DTD jest różna w zależności od stopnia złożoności samego dokumentu.

Obecność definicji typu dokumentu i ścisłe przestrzeganie zawartych w niej reguł jest niezbędnym warunkiem, jaki musi spełniać właściwy dokument XML. Każdy valid dokument, będący oczywiście dobrze uformowanym dokumentem XML, jest jednocześnie także well-formed dokumentem SGML. Zastosowanie DTD daje dostęp pewnych zaawansowanych funkcji języka XML, takich jak mechanizmy łączenia, czy encje, co w znaczny sposób zwiększa skalę zastosowań takich dokumentów, a fakt, że funkcjonują one w ustrukturyzowanym środowisku sprawia, że ich pisanie, przetwarzanie, magazynowanie i sposoby prezentacji są znacznie ułatwione.

3.9. UMIEJSCOWIENIE DTD

Deklaracja typu dokumentu, należąca wraz z deklaracją XML do tzw. prologu dokumentu, zawiera sama w sobie lub wskazuje na umiejscowienie DTD. Inaczej mówiąc, definicja znakowania złożona z deklaracji: typów elementów, atrybutów, encji i notacji[14] , bowiem tym w rzeczywistości jest DTD, może zawierać się w wewnętrznym bądź zewnętrznym względem dokumentu podzbiorze lub w obydwu tych miejscach. Schemat dokumentu dla DTD umieszczonego w wewnętrznym podzbiorze (3.9a):

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE nazwa_elementu_podstawowego
[..deklaracje poszczególnych elementów... ]>
        <nazwa_elementu_podstawowego>
                (...tu główna część dokumentu ...)
        </nazwa_elementu_podstawowego>
```

W przypadku gdy DTD jest zawarte w zewnętrznym podzbiorze schemat wygląda następująco (3.9b):

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE nazwa_elementu_podstawowego
 SYSTEM "adres_pliku_z_dtd">
```

[14] Patrz rozdział: 3.17. Deklaracje notacji.

```
<nazwa_elementu_podstawowego>
        (... główna część dokumentu ...)
</nazwa_elementu_podstawowego>
```

Należy zwrócić tu uwagę na wartość parametru standalone („no") oraz obecność słowa SYSTEM (podobnie jak każde inne słowo kluczowe występujące w DTD musi być napisane wielkimi literami). Adres pliku może być podany w postaci bezwzględnej (pełny URL), jak i relatywnej w stosunku do danego pliku. Wygodną cechą rozwiązania z zewnętrznym DTD jest możliwość współdzielenia pliku z definicją znakowania przez wiele dokumentów o tej samej strukturze, co może być pomocne w procesach komunikacji, usprawniając obieg informacji między np. organizacjami działającymi na tym samym polu. Poza tym unika się konieczności wielokrotnego powtarzania definicji markupu, a fakt zgromadzenia wszystkich standardów w jednym miejscu umożliwia ich łatwiejszą i szybszą aktualizację.

Istnieje również możliwość, aby dokument korzystał z markupu podzielonego między zewnętrzny a wewnętrzny podzbiór. Schemat (3.9c):

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE nazwa_elementu_podstawowego
SYSTEM "adres_pliku_z_dtd"
[ ...deklaracje elementów... ]>
   <nazwa_elementu_podstawowego>
   (... główna część dokumentu ...)
   </nazwa_elementu_podstawowego>
```

Innym sposobem korzystania z zewnętrznego DTD jest zastosowanie słowa kluczowego PUBLIC, które pozwala korzystać głównie z opracowanych wcześniej, najczęściej przez wielkie światowe organizacje, definicji znakowań. Schemat:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE nazwa_elementu_podstawowego PUBLIC "nazwa_DTD"
                                        "adres_pliku_z_dtd" >
    <nazwa_elementu_podstawowego>
    ... główna część dokumentu ...
    </nazwa_elementu_podstawowego>
```

gdzie "nazwa_DTD" dla standardów ISO zaczyna się od łańcucha "ISO", dla zatwierdzonych przez ISO innych standardów od znaku plus (+) a dla niezatwierdzonych standardów pierwszy znak to myślnik (-). Dalej pojawiają dane o autorze DTD, typie definiowanego dokumentu i określenie języka zgodne z ISO 639[15]. Wszystkie te informacje są oddzielone znakiem podwójnego ukośnika (//).

[15] International Organization for Standardization: *ISO 639:1988. Code for the representation of names of languages*, Geneva, ISO 1988.

3.10. DEKLARACJE TYPÓW ELEMENTÓW

Aby można było mówić o właściwym dokumencie XML, wszystkie jego elementy muszą być zdefiniowane w DTD, co ma jednoznacznie określić ich nazwę, zawartość (w tym ewentualnych potomków) oraz możliwy zestaw towarzyszących im atrybutów. Żaden element nie może być zadeklarowany więcej niż raz. Schemat:

<!ELEMENT nazwa_elementu zawartość_dokumentu>

Przykład: załóżmy, że chcemy stworzyć prosty dokument z adresami kontaktowymi do naszych znajomych (3.10a):

```
<?xml version="1.0" encoding="ISO-8859-2" standalone="yes"?>
<!DOCTYPE CONTACTS [
<!ELEMENT CONTACTS (CONTACT)>
<!ELEMENT CONTACT (NAME, EMAIL)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT EMAIL (#PCDATA)>]>
<CONTACTS>
<CONTACT>
      <NAME> Jan Kowalski </NAME>
      <EMAIL> kowalski@firma.com.pl </EMAIL>
</CONTACT>
</CONTACTS>
```

Zdefiniowano tu element podstawowy CONTACTS, który zawiera element CONTACT nazywany jego potomkiem. Dalej CONTACT jest rodzicem elementów NAME i EMAIL, których zawartością z kolei są parsowane dane znakowe (oznaczane jako **#PCDATA**). Dla wyrażenia, że jakiś element może mieć dowolnego potomka (w tym także zawierać dane #PCDATA) używane jest słowo kluczowe ANY: ***<!ELEMENT NAZWA_ELEMENTU ANY>***. Należy tu zauważyć, że w praktyce deklaracja taka ma sens praktycznie tylko w przypadku elementu podstawowego, gdyż stanowi on niejako element nadrzędny wobec innych i sam nie może być niczyim potomkiem, bowiem łamałoby to zasady tworzenia dokumentów dobrze uformowanych.

Używając odpowiednich wyrażeń możemy bardzo dokładnie zdefiniować relacje pomiędzy elementami a ich potomkami. Mogą one wyglądać w następujący sposób: element X jest valid jeśli zawiera jednego bądź więcej potomków Y lub jednego potomka Z. Dokonuje się tego wykorzystując **model zawartości dokumentu** (ang. ***element content model***) używający zbioru specjalnych wskaźników (Tab. 2) opisujących zachowanie elementów:

Tabela 2

Operator	Znaczenie
+	jedno lub więcej wystąpień
*	zero lub więcej wystąpień
?	zero lub jedno wystąpienie
()	grupuje wyrażenia
|	logiczne OR
,	logiczne AND

Nawiązując do wcześniejszego przykładu, jeśli zmienilibyśmy deklarację elementu CONTACT na następującą: ***<!ELEMENT CONTACT (NAME, EMAIL+)>*** to element EMAIL byłby elementem wymaganym i powtarzalnym, co oznacza, że musiałby pojawić się co najmniej raz w zawartości CONTACT. Inaczej mówiąc każda osoba mogłaby mieć więcej niż jeden adres poczty elektronicznej (3.10b).

Z kolei deklaracja **<!ELEMENT CONTACT (NAME, EMAIL*)>** określa EMAIL jako element opcjonalny I powtarzalny, czyli każda osoba może równie dobrze mieć kilka adresów jak i żadnego (3.10c).

Załóżmy teraz, że w opisie każdej osoby może pojawić się maksymalnie jeden element EMAIL lub, w przypadku gdy osoba nie posiada własnej skrzynki pocztowej, element ten nie występuje. Zapisuje się to następująco: **<!ELEMENT CONTACT (NAME, EMAIL?)>** (3.10d).

Brak jakiegokolwiek wskaźnika przy elemencie zdefiniowanym w DTD oznacza, że element ten jest wymagany i musi wystąpić w dokumencie dokładnie raz.

Potomkowie elementów są zgrupowani wewnątrz pary nawiasów okrągłych „(„ ")". Chcąc wymusić występowanie poszczególnych elementów w dokumencie w ściśle określonej kolejności, należy w obrębie DTD oddzielić je od siebie znakiem przecinka (,). Stąd w poprzednich przykładach element EMAIL będzie zawsze poprzedzony elementem NAME.

Grupę elementów, z których w dokumencie musi pojawić się jeden i tylko jeden, definiujemy posługując się znakiem pionowej kreski (|). Wracając do przykładu załóżmy, że do każdego kontaktu oprócz nazwiska potrzebujemy jednego adresu pocztowego lub numeru telefonu (powiedzmy, że zawiera się on w elemencie PHONE). Do DTD należy dodać definicję PHONE:

<!ELEMENT PHONE (#PCDATA)>

oraz zmienić deklarację CONTACT na:

<!ELEMENT CONTACT (NAME, (EMAIL | PHONE))>

Teraz w dokumencie może wystąpić albo element EMAIL albo PHONE (3.10d). Ponieważ wewnątrz grupowania można używać wyłącznie jednego łącznika – (,) albo (|) – dlatego należy stosować zagnieżdżanie nawiasów. Poniższy przykład jest błędną deklaracją:

<!ELEMENT CONTACT (NAME, EMAIL | PHONE)>

Wszystkie wskaźniki wystąpień tj. plus (+), gwiazdka (*) oraz znak zapytania (?) mogą odnosić się nie tylko do pojedynczych, ale także zgrupowanych elementów. Np.:

<!ELEMENT CONTACT (NAME, EMAIL)+>

Tu w ramach jednego elementu CONTACT, co najmniej raz lub wielokrotnie wystąpi element EMAIL, za każdym razem poprzedzony elementem NAME.

Szczególnym przypadkiem elementu jest taki, w którego zawartości mogą występować zarówno parsowane dane znakowe jak i potomkowie (ang. ***mixed content***). Deklaracja wygląda wtedy następująco (3.10e):

<!ELEMENT nazwa_elementu (#PCDATA | nazwa_potomka1 | nazwa_potomkaN)>

W tym przypadku nie ma możliwości określenia liczby wystąpień poszczególnych potomków (3.10e).

Elementy puste deklaruje stosując słowo kluczowe EMPTY wg schematu:

<!ELEMENT nazwa_elementu EMPTY>

3.11. DEKLARACJE LISTY ATRYBUTÓW

Atrybuty są stosowane do łączenia par nazwa – wartość z elementami wchodzącymi w skład dokumentu. Występują one wyłącznie w tagach otwierających i tagach pustych elementów. W przypadku dokumentów właściwych XML deklaruje się je według schematu:

<!ATTLIST nazwa_elementu nazwa_atrybutu1 typ wartość_domyślna nazwa_atrybutuN typ wartość_domyślna >

Nazwa_elementu wskazuje na element, z którym skojarzone są atrybuty. Gdy dla danego typu elementu występuje więcej niż jedna deklaracja listy atrybutów traktowane są one łącznie, natomiast jeśli w kilku z nich znajdu-

ją się różne, sprzeczne definicje poszczególnych atrybutów, ta która występuje jako pierwsza jest wiążąca (pozostałe są ignorowane). Zaleca się, aby jednemu elementowi odpowiadała tylko jedna lista atrybutów.

3.12. DEFINICJE WARTOŚĆI DOMYŚLNYCH

Określają wartość atrybutu, która powinna być użyta, gdy nie zrobi tego autor dokumentu (ang. ***default value***) oraz mówią o tym, kiedy dany atrybut powinien być obecny. Występują one w trzech formach:
#REQUIRED – oznacza, że dany atrybut musi zawsze towarzyszyć danemu elementowi i chociaż DTD nie określa jego wartości to w samym dokumencie musi on występować i mieć konkretną wartość, która nie łamie żadnych zasad leksykalnych charakterystycznych dla zadeklarowanego typu atrybutu.
#IMPLIED – oznacza atrybut opcjonalny (może, lecz nie musi wystąpić)
#FIXED – stosowane w przypadku, gdy chce się z góry narzucić konkretną wartość atrybutowi. Podaje się wg następującego schematu:

<!ATTLIST nazwa_elementu nazwa_atrybutu typ #FIXED „wartość”>

Ponadto istnieje możliwość nadania atrybutowi bezpośrednio pewnej wartości, która będzie użyta w przypadku, gdy autor nie dokona w treści dokumentu jej redefinicji:

<ATTLIST ELEMENT atrybut_a „159”>

Przykłady zastosowań są podane w następnej części poświęconej typom atrybutów.

3.13. TYPY ATRYBUTÓW

Istnieje dziesięć rodzajów atrybutów, które można zaliczyć do trzech grup:

– **atrybut znakowy** (ang. ***string type***): CDATA.
– **atrybuty identyfikacyjne** (ang. ***tokenized type***): ID, IDREF, IDREFS, NMTOKEN, NMTOKENS, ENTITY, ENTITIES.
– **atrybuty wyliczeniowe** (ang. ***enumerated type***): NOTATION, wyliczenie.

CDATA – Jego wartością może być dowolny łańcuch znakowy, oczywiście nie zawierający opisanych wcześniej znaków specjalnych (&), (<) i (”). Przykład (3.13.a):

```
<!ELEMENT DIALOG (#PCDATA)>
<!ATTLIST DIALOG OSOBA CDATA>
(...)
<DIALOG OSOBA="Kowalski"> To mówi Kowalski.</DIALOG>
```

ID – Stanowi unikatowy identyfikator dla atrybutu, który określa element w kontekście dokumentu. Spełnia podobną rolę, co wewnętrzne linki w HTML-u (oznaczane znakiem hash (#)). Używany głównie przez programy i języki skryptowe przetwarzające dokument. ID musi być poprawną nazwą XML. Każdy element może mieć maksymalnie jeden atrybut ID. Musi on występować razem z #IMPLIED lub #REQUIRED. Patrz przykład poniżej.

IDREF i IDREFS – Pozwala, aby wartość jednego atrybutu była elementem w innym miejscu dokumentu, o ile wartość IDREF jest wartością ID elementu, do którego się odnosi. Przykład (3.13b):

```
<!ELEMENT CONTACT (NAME, EMAIL)>
<!ATTLIST CONTACT CONTACT_NUM ID #REQUIRED>
<!ATTLIST CONTACT SUPERIOR IDREF #IMPLIED>
(...)
<CONTACT CONTACT_NUM = "2">
<NAME>Jan Kowalski</NAME>
<EMAIL>kowalski@firma.com</EMAIL>
</CONTACT>
<CONTACT CONTACT_NUM = "1" SUPERIOR = "2">
<NAME>Ewa Nowak</NAME>
<EMAIL>nowak@firma.com</EMAIL>
</CONTACT>
```

NMTOKEN i NMTOKENS – Użyteczne głównie dla aplikacji przetwarzających. Stosowane są do określania właściwych nazw (*ang. valid names*). Można ich używać, gdy wiąże się jakiś komponent (np. klasę Javy lub algorytm bezpieczeństwa) z danym elementem. Przykład:

```
(...)
<!ATTLIST DATA AUTHORIZED_USERS NMTOKENS #IMPLIED>
(...)
<DATA SECURITY="ON"
AUTHORIZED_USERS = "Nowak Kowalski Piekarska">
(...chronione dane...)
</DATA>
```

ENTITY i ENTITIES – jako wartości przyjmują nazwy zadeklarowanych w DTD nieparsowanych encji. Patrz przykład poniżej.

NOTATION – typ ten pozwala atrybutom na przyjmowanie wartości zadeklarowanej w deklaracji notacji wraz z skojarzonym identyfikatorem SYSTEM lub PUBLIC, które określają sposób interpretacji elementu, do którego należy atrybut. Najczęściej używany do kojarzenia określonych typów plików z odpowiednimi aplikacjami.

```
<!ELEMENT MOVIE EMPTY>
<!ATTLIST MOVIE Source ENTITY #REQUIRED>
<!ATTLIST MOVIE Player  NOTATION (mp3) "mp3">
<!ENTITY StairwayToHeaven SYSTEM "../mp3/songs/Stairway.mp3">
<!NOTATION mp3  SYSTEM "winamp.exe">
(...)
<MOVIE Source = "&StairwayToHeaven;" Player = "mp3"/>
```

WYLICZENIE – lista z określonymi, wymienionymi elementami do wyboru, z których jeden, dowolny może być wartością atrybutu (3.13c).

```
<!ELEMENT SONG (#PCDATA)>
<!ATTLIST SONG Source ( CD | LP | MC | VIDEO | MP3) "CD">
(...)
<SONG Source="MP3">Stalrway to heaven</SONG>
<SONG>Child in time</SONG>
```

Warto zauważyć, że w powyzszym przykładzie, dla drugiego wystąpienia elementu „SONG" parametr „Source" przyjmie wartość „CD", gdyż właśnie ta wartość została zdefiniowana jako domyślna w definicji ATTLIST.

3.14. ATRYBUTY SPECJALNE

XML dysponuje dwoma specjalnymi atrybutami, które dotyczą odpowiednio a) obsługi „białych miejsc" (***xml:space***) b) identyfikacji języka dokumentu (***xml:lang***).

– Wspomniane już wcześniej „białe miejsca" są często stosowane przez autorów dla polepszenia czytelności dokumentu, oddzielenia znakowania od zawartości itp. Niektóre z nich mogą stanowić integralną część zawartości dokumentu (gdy mamy do czynienia np. z zapisem utworu poetyckiego) i jako takie nie powinny być zmieniane przez aplikację przetwarzającą[16]. Aby wymusić zachowanie wszystkich istniejących „białych miejsc" w danym elemencie należy posłużyć się atrybutem xml:space np. w następujący sposób:

```
<!ATTLIST Poem xml:space (default|preserve) „preserve">
```

[16] Domyślnie każde „białe miejsce" lub ich sekwencja jest zamieniana na znak pojedynczej spacji.

Wybrana wartość „preserve" nakazuje zachować wszystkie „białe miejsca"; wybranie wartości „default" oznaczałoby zastosowanie wszystkich domyślnych procedur w przypadku wystąpienia tych znaków.

– Czasami pożyteczną rzeczą jest określenie języka używanego w zawartości elementu czy też wartości atrybutu. Np.:

<QUOTATION xml:lang="en"> This is an English text. </QUOTATION>

Wartością parametru „xml:lang" najczęściej jest dwuliterowy kod języka zdefiniowany przez normę ISO 639.

3.15. SEKCJE WARUNKOWE

Sekcjami warunkowymi (ang. ***conditional sections***) są wydzielone fragmenty DTD znajdujące się w zewnętrznym podzbiorze, które przy użyciu odpowiednich wyrażeń można z definicji typu dokumentu wykluczyć bądź do niej włączyć. Przykłady:

<![INCLUDE [<!ELEMENT Phone (#PCDATA)>]]>
<![IGNORE [<!ELEMENT EMAIL (#PCDATA>]]>

Słowo kluczowe INCLUDE oznacza włączenie zawartości sekcji do DTD a słowo IGNORE wykluczenie. Sekcje mogą być zagnieżdżane, z tym, że obecność INCLUDE wewnątrz sekcji oznaczanej jako IGNORE będzie zinterpretowane jako nakaz wykluczenia z DTD obydwu sekcji.

3.16. ENCJE

Encja jest podstawowym, atomowym elementem fizycznej struktury dokumentu, którą możemy zdefiniować jako obiekt przechowujący pewną zawartość, z którego składa się dany materialny bądź niematerialny system[17] (w naszym przypadku dokument XML). Każdy dokument zawiera jedną encję zwaną **encją dokumentu** (*ang.* ***document entity***), która stanowi korzeń drzewa encji, może obejmować cały dokument i stanowi punkt wyjścia dla przetwarzającego procesora XML. Jako jedyna nie posiada ona własnej nazwy.

Poszczególne encje składające się na dokument mogą być parsowane (ich zawartością jest tekst stanowiący integralną część dokumentu) lub

[17] H. Wayne: *Bazy danych nie tylko dla ludzi biznesu*. Wydawnictwa Naukowo-Techniczne, Warszawa 1994 s. 47.

nie parsowane (mogące zawierać tekst lub dane nie będące XML-em) zawsze skojarzone poprzez nazwę z odpowiednią notacją. Wywołanie takiej encji (patrz paragraf poniżej) może wystąpić jedynie jako wartość atrybutu typu ENTITY lub ENTITIES odpowiadającego mu elementu.

Oprócz przedstawionych w rozdziale 3.2 „Dane a znakowanie" encji zastępujących znaki specjalne, wyróżniamy jeszcze dwa typy encji: 1) **ogólne** (*ang.* ***general entities***) 2) **parametryczne** (*ang.* ***parameter entities***), z których każda składa się z deklaracji (w obrębie DTD) i **odniesienia do encji** (*ang.* ***entity reference***).

1. Używane są w zawartości dokumentu. Deklaracja i odniesienie wyglądają w następujący sposób:

<!ENTITY Nazwa_encji "...zawartość encji...">
(...)
&Nazwa_encji;

Encja ogólna nie może być wywoływana w deklaracji DOCTYPE, ale za to odniesienie do niej może wystąpić jako wartość atrybutu elementu. Przykład (3.16.a):

<!ELEMENT NAME (#PCDATA)>
<!ATTLIST NAME Status CDATA #REQUIRED>
<!ELEMENT EMAIL (#PCDATA)>
<!ENTITY ADDRESS "@firma.com.pl">
<!ENTITY student "student, rok 5, zaoczny">
(...)
<NAME Status="&student;"> Jan Kowalski </NAME>
<EMAIL>kowalski&ADDRESS;</EMAIL>

2. Encje parametryczne używane są wyłącznie w DTD i są encjami parsowanymi. Składnia:

<!ENTITY % Nazwa_encji "...zawartość encji..">
(...)
%Nazwa_encji;

Przykład (3.16b):

<!ENTITY % TAG_NAMES "NAME | EMAIL | PHONE">
<!ELEMENT PERSONAL_CONTACT (%TAG_NAMES; | BIRTHDAY)>
<!ELEMENT BUSINESS_CONTACT
(%TAG_NAMES;| COMPANY_NAME)>

Reguły dotyczące tworzenia encji:

– Encja musi być zadeklarowana, żeby można się było do niej odwołać.

– Wprawdzie encje mogą wywoływać inne encje, ale nie może wystąpić zjawisko sprzężenia zwrotnego tj. jeśli encja A wywołuje encję B, to B w żaden sposób (pośrednio lub bezpośrednio) nie może wywołać A.
– Tekst, który stanowi zawartość parsowanej encji musi być poprawnie uformowanym XML-em.

Podobnie jak w przypadku DTD, encje także mogą zawierać się w wewnętrznym bądź zewnętrznym względem dokumentu zbiorze. W takim przypadku procesor XML, korzystając z odniesienia do encji, włączy do dokumentu zawartość określonego pliku. Dostęp do zewnętrznych encji uzyskuje się analogicznie jak w przypadku zewnętrznych DTD tzn. za pomocą identyfikatorów SYSTEM i PUBLIC. Np.:

```
<!ENTITY license_agreement SYSTEM "http://www.mydomain.com/
license.xml">
<!ENTITY open-hatch PUBLIC
"-//Textuality//TEXT Standard open-hatch boilerplate//EN
"http://www.textuality.com/boilerplate/OpenHatch.xml">
```

W przypadku odwoływania się do zewnętrznej nieparsowanej encji należy użyć słowa kluczowego NDATA i nazwy odpowiadającej jej notacji. Przykład:

```
<!ENTITY logo SYSTEM "../images/logo.gif" NDATA gif >
```

Zewnętrzne parsowane encje mogą rozpoczynać się od **deklaracji tekstu** (*ang.* ***text declaration***), podobnej do deklaracji XML, zawierającej opcjonalnie informacje o wersji języka znakowania i obowiązkowo o zastosowanym sposobie kodowania np.:

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

Wywołania zewnętrznych encji nie mogą wystąpić jako wartości atrybutów elementów.

3.17. DEKLARACJE NOTACJI

Notacje (*ang.* ***notations***) są stosowane do określania formatów nieparsowanych encji, formatów elementów z atrybutem NOTATION lub aplikacji, do której skierowana jest instrukcja przetwarzająca. Deklaracja notacji (zawarta w DTD) określa jej nazwę, która będzie stosowana w encjach, deklaracjach typów atrybutów i specyfikacjach atrybutów oraz zewnętrzny identyfikator pozwalający procesorowi XML zlokalizować aplikację pomocniczą, będącą w stanie przetworzyć dane zawarte w notacji. Przykłady:

```
<!ATTLIST MOVIE Player  NOTATION (mp3 | cd) "mp3">
<!NOTATION mp3  SYSTEM "winamp.exe">
<!NOTATION cd  SYSTEM "cdplayer.exe">
(...)
<!ENTITY logo SYSTEM "../images/logo.gif" NDATA gif >
<!NOTATION gif SYSTEM "./Program Files/Adobe/photoshp.exe">
```

Rozdział 4

PRZYKŁADY DOKUMENTÓW

4.1. E-MAIL

Poniżej przedstawiono przykład dokumentu XML ze zdefiniowaną w wewnętrznym podzbiorze DTD strukturą listu elektronicznego. Składa się on z elementu podstawowego email, którego bezpośrednimi potomkami są sekcje: nagłówkowa (*head*) i treściowa (*body*). W nagłówku obowiązkowo należy przedstawić nadawcę listu (wewnątrz atrybutu *from*) poprzez podanie adresu poczty elektronicznej, opcjonalnie poprzedzonego imieniem i/lub nazwiskiem. Uwzględniono również możliwość istnienia nadawcy o wielu imionach lub nazwiskach. Adresatami listu może być kilku odbiorców (w elemencie *to*), z których każdy jest opisywany w sposób analogiczny do nadawcy. Ponadto należy podać temat listu (*subject*) oraz określić datę wysłania (atrybut *date*) i znaczenie danej przesyłki w trzystopniowej skali (atrybut *importance*, wartość domyślna „średnia").

Sama treść listu (wewnątrz *body*) składa się z co najmniej jednego akapitu (*p*) i podpisu nadawcy (*signature*). Można poprzedzić to przywitaniem z grzecznościową formułą (*salutation*) oraz dołączyć do listu pliki (pusty element *attach*) o określonych nazwach i typie przesyłki (atrybuty *name* i *encoding*).

Przykładowy, krótki e-mail zbudowany według powyższego schematu znajduje się za DTD. Całość listingu można zachować w pliku np. email.xml i otworzyć w IE 5.0.

```
<?xml version="1.0" encoding="ISO-8859-2" standalone="yes"?>
<!DOCTYPE  email [
```

```
<!ELEMENT  email       (head, body)>
<!ELEMENT  head        (from, to, subject)>
<!ATTLIST  head        importance (high | medium | low) "medium"
                       date CDATA #REQUIRED>
<!ELEMENT  from        (person?, address)>
<!ELEMENT  to          (person?, address)+>
<!ELEMENT  person      (firstname*, lastname*)>
<!ELEMENT  firstname   (#PCDATA)>
<!ELEMENT  lastname    (#PCDATA)>
<!ELEMENT  address     (#PCDATA)>
<!ELEMENT  subject     (#PCDATA)>
<!ELEMENT  body        (salutation?, p+, signature, attach*)>
<!ELEMENT  salutation  (#PCDATA)>
<!ELEMENT  p           (#PCDATA)>
<!ELEMENT  signature   (#PCDATA)>
<!ELEMENT  attach      EMPTY>
<!ATTLIST  attach      encoding (mime|binhex) "mime"
                       name CDATA #REQUIRED>]>

<email>
<head importance="high" date="12.03.2000">
      <from>
      <person ><firstname>Jan</firstname>
      <lastname>Kowalski</lastname></person >
      <address>jkowalski@free.com.
      pl</address>
      </from>

      <to>
      <person >
      <firstname>Tomasz</firstname>
      <lastname>Nowak</lastname></person >
      <address>tnowak@firma1.com.pl</address>
      <address>marcin@firma2.com.pl</address>
      </to>

   <subject>Przypomnienie</subject>
</head>

<body>
   <salutation>Szanowny Panie!</salutation>
   <p>Przypominam o jutrzejszym zebraniu o godz 14:30 w sali kon-
   ferencyjnej.</p>
<p>Przesyłam jednocześnie obiecane materiały prezentacyjne.</p>
```

```
<signature>JK</signature>
<attach encoding="mime" name="slajdy1.ppt"/>
<attach encoding="mime" name="slajdy2.ppt"/>
</body>
</email>
```

4.2. DRZEWO GENEALOGICZNE

Przedstawiony poniżej listing[1] opisuje drzewo genealogiczne (niepełne) pewnej fikcyjnej rodziny założonej przez Zofię Nowak i Stanisława Kowalskiego, którzy zawarli związek małżeński w dniu szesnastym czerwca 1923 roku. Para ta miała piątkę dzieci: Marię, Wiktora, Paulinę, Julię i Roberta. Maria jako pierwsza opuściła dom wychodząc za mąż za Grzegorz Rudzyńskiego, któremu urodziła Tomasza i Macieja. Z kolei Wiktor Kowalski poślubił Helenę Mieczyńską i wychował córkę Katarzynę.

```
<?xml version="1.0" encoding="windows-1250"?>
<!DOCTYPE FAMILYTREE [
<!ELEMENT FAMILYTREE (PERSON | FAMILY)*>
<!ELEMENT PERSON (NAME*, BORN?, DIED?, SPOUSE?)>
<!ATTLIST       PERSON  ID ID #REQUIRED
                FATHER  CDATA  #IMPLIED
                MOTHER  CDATA  #IMPLIED>
<!ELEMENT  NAME (#PCDATA)>
<!ELEMENT BORN (#PCDATA)>
<!ELEMENT DIED  (#PCDATA)>
<!ELEMENTSPOUSE EMPTY>
<!ATTLIST       SPOUSE IDREF IDREFS #REQUIRED>
<!ELEMENT FAMILY (HUSBAND, WIFE, MARRIAGE?, CHILD*) >
<!ATTLIST       FAMILY ID ID #REQUIRED>
<!ELEMENT HUSBAND EMPTY>
<!ATTLIST       HUSBAND IDREF IDREF #REQUIRED>
<!ELEMENT WIFE EMPTY>
<!ATTLIST       WIFE IDREF IDREF #REQUIRED>
<!ELEMENT MARRIAGE (#PCDATA)>
<!ELEMENT CHILD EMPTY>
<!ATTLIST       CHILD IDREF IDREF #REQUIRED>]>

<FAMILYTREE>
```

[1] Przykład jest oparty na kodzie zamieszczonym w: E. R. Harold: *XML bible*. IDG Books Worldwide 1999. Pewne elementy zostały dodane lub zmienione przez autora niniejszej pracy.

```
<PERSON ID="p1">
        <NAME>Zofia Nowak</NAME>
        <BORN>11 luty 1900</BORN>
        <DIED>12 kwiecień 1965</DIED>
        <SPOUSE IDREF="p2"/>
</PERSON>
<PERSON ID="p2">
        <NAME>Stanisław Kowalski</NAME>
        <SPOUSE IDREF="p1"/>
</PERSON>
<PERSON ID="p3" FATHER="p2" MOTHER="p1">
        <NAME>Maria Kowalska</NAME>
        <BORN>29 wrzesień 1924</BORN>
        <DIED>29 wrzesiń 1975</DIED>
        <SPOUSE IDREF="p4"/>
</PERSON>
<PERSON ID="p4" FATHER="p2" MOTHER="p1">
        <NAME>Grzegorz Rudzyński</NAME>
        <SPOUSE IDREF="p3"/>
</PERSON>
<PERSON ID="p7">
        <NAME>Tomasz Rudzyński</NAME>
        <BORN>1 maj 1950</BORN>
</PERSON>
<PERSON ID="p5" FATHER="p2" MOTHER="p1">
        <NAME>Wiktor Kowalski</NAME>
        <BORN>15 grudzień 1930</BORN>
        <SPOUSE IDREF="p6"/>
</PERSON>
<PERSON ID="p6">
        <NAME>Helena Mieczyńska</NAME>
        <SPOUSE IDREF="p5"/>
        </PERSON>
<PERSON ID="p8" FATHER="p2" MOTHER="p1">
        <NAME>Paulina Kowalska</NAME>
        <BORN>15 grudzień 1930</BORN>
        <DIED>16 grudzień 1930</DIED>
</PERSON>
<PERSON ID="p9">
        <NAME>Maciej Rudzyński</NAME>
        <BORN>23 styczeń 1954</BORN>
</PERSON>
<PERSON ID="p10" FATHER="p2" MOTHER="p1">
        <NAME>Julia Kowalska</NAME>
```

```
        <BORN>18 marzec 1934</BORN>
    </PERSON>
    <PERSON ID="p11">
        <NAME>Katarzyna Kowalska</NAME>
        <BORN>12 czerwiec 1960</BORN>
    </PERSON>
    <PERSON ID="p12" FATHER="p2" MOTHER="p1">
        <NAME>Robert Kowalski</NAME>
        <BORN>28 marzec 1938</BORN>
    </PERSON>

    <FAMILY ID="f1">
        <HUSBAND IDREF="p2"/>
        <WIFE IDREF="p1"/>
        <MARRIAGE>16 czerwiec 1923</MARRIAGE>
        <CHILD IDREF="p3"/>
        <CHILD IDREF="p6"/>
        <CHILD IDREF="p8"/>
        <CHILD IDREF="p10"/>
        <CHILD IDREF="p12"/>
    </FAMILY>
    <FAMILY ID="f2">
        <HUSBAND IDREF="p4"/>
        <WIFE IDREF="p3"/>
        <MARRIAGE>12 kwiecień 1948</MARRIAGE>
        <CHILD IDREF="p7"/>
        <CHILD IDREF="p9"/>
    </FAMILY>
    <FAMILY ID="f3">
        <HUSBAND IDREF="p5"/>
        <WIFE IDREF="p6"/>
        <CHILD IDREF="p11"/>
    </FAMILY>

</FAMILYTREE>
```

Warto zwrócić uwagę, że każda osoba (element person) pojawiająca się w drzewie genealogicznym posiada własny, unikatowy atrybut ID będący później podstawą do wyrażania wszelkich relacji (rodzic-dziecko, dziecko-matka, mąż-żona itd.), czego z kolei dokonuje się używając atrybutu typu IDREF[2], stanowiącego odwołanie do odpowiedniego, zdefiniowanego wcześniej identyfikatora. Zauważmy ponadto, że element SPOUSE, określający relację „jest żoną/mężem" jako jedyny zawiera atrybut typu

[2] Patrz też: 3.13. Typy atrybutów.

IDREFS, co pozwala na odwołanie do więcej niż jednej osoby poprzez jej ID. Taka definicja jest niezbędna by prawidłowo opisać osobę, która więcej niż raz wstępowała w związek małżeński.

Wprawdzie przykład ten opisuje dosyć ubogo każdą osobę, pozwalając jedynie na podanie jej nazwiska, daty urodzenia, śmierci i ślubu, to jednak jego struktura jest na tyle elastyczna, że łatwo go wzbogacić dodatkowymi informacjami jak wykształcenie, zawód czy miejsce zamieszkania.

4.3. DYSKOGRAFIA

Załóżmy, że chcemy stworzyć dokumenty, w których będziemy zamieszczali dyskografie swoich ulubionych wykonawców muzycznych. Plik DTD (np. discography.dtd) może wyglądać w następujący sposób:

```
<!ELEMENT  Discography   (Artist | Group)>
<!ELEMENT  Artist        (Pseudo?, Last_Name+, First_Name+,
                         Birth_Date?,
                         Nationality*, Instrument+, Album+)>
<!ELEMENT  Pseudo        (#PCDATA)>
<!ELEMENT  Last_Name     (#PCDATA)>
<!ELEMENT  First_Name    (#PCDATA)>
<!ELEMENT  Birth_Date    (#PCDATA)>
<!ELEMENT  Nationality   (#PCDATA)>
<!ELEMENT  Instrument    (#PCDATA)>
<!ELEMENT   Album        (Cover, Title, Date, Rec_Comp*,
                         Musicians?, Songs)>
<!ELEMENT  Cover         EMPTY>
<!ATTLIST  Cover         source CDATA #REQUIRED
                         type NOTATION (GIF | JPEG) "JPEG">
<!NOTATION GIF           SYSTEM "c:\windows\system\gif.dll">
<!NOTATION JPEG          SYSTEM "c:\windows\system\jpeg.dll">
<!ENTITY   adres         "http://www.server/images/">
<!ELEMENT  Title         (#PCDATA)>
<!ELEMENT  Date          (#PCDATA)>
<!ELEMENT  Rec_Comp      (#PCDATA)>
<!ELEMENT  Musicians     (Musician+)>
<!ELEMENT  Musician      (Last_Name, First_Name,
                         Instrument+)>
<!ELEMENT  Songs         (Song+)>
<!ATTLIST  Songs         number CDATA #REQUIRED
                         total_time  CDATA #REQUIRED >
<!ELEMENT  Song          (#PCDATA)>
```

```
<!ATTLIST   Song time     CDATA #REQUIRED>
<!ELEMENT   Group         (Name, Begin, Finish?, Nationality,
                          Creators, Album+)>
<!ELEMENT   Name          (#PCDATA)>
<!ELEMENT   Begin         (#PCDATA)>
<!ELEMENT   Finish        (#PCDATA)>
<!ELEMENT   Creators      (Musician+)>
```

Pierwszą rzeczą, na jaką trzeba się zdecydować przy tworzeniu plików zgodnych z powyższą definicją jest to, czy będzie to dyskografia zespołu (element *Group*) czy pojedynczego wykonawcy (*Artist*). W tym drugim przypadku na początku podajemy ogólne in-formacje o ewentualnym pseudonimie, nazwisku i imieniu (bądź nazwiskach i imionach) artysty, dacie jego urodzenia (element *Birth_Date* jest opcjonalny) narodowości i instrumentach, na jakich gra.

Dalej następują opisy poszczególnych płyt, na które składają się:

– zachowana w pliku o formacie GIF lub JPEG okładka płyty (tu założono, że w zależności od typu pliku jego przetwarzaniem zajmie się fikcyjna biblioteka DLL, której adres podany jest w deklaracjach NOTATION[3])

– tytuł, data wydania i wytwórnia

– opcjonalnie muzycy biorący udział w nagraniu płyty (ich nazwiska, imiona i instrumenty których używali)

– zbiór tytułów umieszczonych na płycie nagrań wraz z czasem trwania całego albumu i ilością piosenek oraz czasami poszczególnych nagrań.

Poniżej przedstawiono przykład pliku zgodnego z omawianym DTD z opisem dwóch płyt Petera Hammilla.

```
<?xml version='1.0'?>
<!DOCTYPE Discography SYSTEM "discography.dtd">
<Discography>
<Artist>
   <Last_Name>Hammill</Last_Name>
   <First_Name>Peter</First_Name>
   <Birth_Date>1948</Birth_Date>
   <Nationality>eng</Nationality>
   <Instrument>vocals</Instrument>
   <Instrument>guitar</Instrument>
   <Instrument>keyboard</Instrument>
   <Album>
      <Cover source="&adres;over.gif" type="GIF" />
      <Title>Over</Title>
      <Date>1977</Date>
```

[3] Patrz też: 3.17. Deklaracje notacji.

```
        <Rec_Comp>Charisma Records Ltd.</Rec_Comp>
        <Musicians>
                <Musician>
                <Last_Name>Evans</Last_Name>
                <First_Name>Guy</First_Name>
                <Instrument>drums</Instrument>
                </Musician>
                <Musician>
                <Last_Name>Potter</Last_Name>
                <First_Name>Nick</First_Name>
                <Instrument>bass</Instrument>
                </Musician>
                <Musician>
<Last_Name>Smith</Last_Name>
<First_Name>Graham</First_Name>
<Instrument>violin</Instrument>
                </Musician>
        </Musicians>
        <Songs number="8" total_time="44:18">
                <Song time="5:12">Crying Wolf</Song>
                <Song time="4:13">Autumn</Song>
                <Song time="8:42">Time Heals</Song>
                <Song time="5:33">Alice (Letting Go)</Song>
                <Song time="6:57">This Side Of Looking Glass</Song>
                <Song time="4:44">Betrayed</Song>
                <Song time="3:55">(On Tuesdays She Used To Do)
                  Yoga</Song>
                <Song time="7:11">Lost And Found</Song>
        </Songs>
    </Album>

    <Album>
        <Cover source="&adres;incamera.gif" type="GIF" />
        <Title>In camera</Title>
        <Date>1974</Date>
        <Rec_Comp>Charisma Records Ltd.</Rec_Comp>
        <Rec_Comp>Virgin Records Ltd.</Rec_Comp>
        <Songs number="7" total_time="47:39">
                <Song time="3:42">Ferret And Fetherbird</Song>
                <Song time="5:45">(No More) The Submariner</Song>
                <Song time="4:18">Tapeworm</Song>
                <Song time="3:36">Again</Song>
                <Song time="6:42">Faint-heart And The Sermon</Song>
                <Song time="6:01">The Comet, The Course, The Tail
                </Song>
```

```
            <Song time="17:26">Gog Magog (In Bromine
            Chambers)</Song>
        </Songs>
    </Album>

</Artist>
</Discography>
```

W przykładzie tym założono również, że wszystkie okładki płyt są przechowywane pod adresem http://www.serwer/images/, stąd deklaracja encji[4] „adres" pozwalająca unikać wielokrotnego wpisywania tego samego ciągu liter.

Sporządzenie dyskografii dla zespołu różni się jedynie pierwszymi kilkoma elementami, którymi są nazwa grupy, początek (element *Begin*) i koniec (*Finish*) działalności, narodowość i opis jej założycieli (element *Creators*) dokonywany analogicznie jak muzyków biorących udział w nagrywaniu płyty pojedynczego wykonawcy.

4.4. PRACA MAGISTERSKA

W niniejszym podrozdziale zamieszczono schemat, na którego podstawie można opisać pracę magisterską. Wprawdzie specyfika poszczególnych dokumentów tego rodzaju może czasami nie pasować do przedstawionej tu propozycji ze względu na szeroki zakres, jakiego mogą one dotyczyć (inną strukturę z pewnością będzie miała np. praca będąca bibliografią czy praca inżynierska), to jednak wydaje się, że dużej części prac magisterskich proponowany schemat może odpowiadać.

```
<!ELEMENT MASTERS_THESIS  (GENERAL, TITLE, CONTENTS,
                           PREFACE, CHAPTER+,
                           CONCLUSIONS,BIBLIOGRAPHY,
                           APPENDIX*)>
<!ELEMENT GENERAL   (UNIVERSITY, FACULTY, DEPARTMENT?,
                     AUTHOR, SUPERVISOR, YEAR,
                     NOTE?)>
<!ELEMENT UNIVERSITY      (#PCDATA)>
<!ELEMENT FACULTY         (#PCDATA)>
<!ELEMENT DEPARTMENT      (#PCDATA)>
<!ELEMENT SUPERVISOR      (#PCDATA)>
```

[4] Patrz też: 3.16. Encje.

```
<!ELEMENT YEAR          (#PCDATA)>
<!ELEMENT NOTE          (#PCDATA)>
<!ELEMENT TITLE         (#PCDATA)>
<!ELEMENT CONTENTS      (P+)>
<!ELEMENT PREFACE       (P+)>
<!ELEMENT CHAPTER       NR, TITLE, INTRO?, SUBSECTION+)>
<!ELEMENT INTRO         (P+)>
<!ELEMENT P             (#PCDATA | FOOTNOTE | ACCENT)*>
<!ELEMENT FOOTNOTE      (#PCDATA)>
<!ATTLIST   FOOTNOTE    NR CDATA #REQUIRED>
<!ELEMENT ACCENT        (#PCDATA)>
<!ELEMENT NR            (#PCDATA)>
<!ELEMENT SUBSECTION    NR?, TITLE?, (P*, (IMAGE|EXAMPLE)?)+)>
<!ELEMENT IMAGE         EMPTY>
<!ATTLIST  IMAGE        NAME CDATA #REQUIRED
                        FILE CDATA #REQUIRED>
<!ELEMENT EXAMPLE       (#PCDATA)>
<!ELEMENT CONCLUSIONS   (P+)>
<!ELEMENT BIBLIOGRAPHY  (INT_REF | PRINT_REF)+>
<!ELEMENT INT_REF       (AUTHOR*, TITLE?, ADDRESS)>
<!ELEMENT PRINT_REF     (AUTHOR*, TITLE, ISSUE_DATA)>
<!ELEMENT AUTHOR        (#PCDATA)>
<!ELEMENT ADDRESS       (#PCDATA)>
<!ELEMENT ISSUE_DATA    (#PCDATA)>
<!ELEMENT APPENDIX      (P | IMAGE | EXAMPLE)+>
<!ATTLIST  APPENDIX     TITLE CDATA #IMPLIED>
```

Przedstawione DTD zakłada, że każdą pracę magisterską będzie rozpoczynał krótki blok informacji o charakterze ogólnym (GENERAL), gdzie zostaną zamieszczone dane o uczelni (UNIVERSITY), wydziale (FACULTY) i instytucie (DEPARTMENT), gdzie złożono pracę, jej autorze (AUTHOR), promotorze (SUPERVISOR), roku napisania (YEAR) i otrzymanej ocenie (NOTE). Każdy dokument zgodny z niniejszym schematem musi mieć tytuł, oraz zawierać wstęp, rozdziały, zakończenie, bibliografię i ewentualnie dodatki (APPENDIX). Najważniejszym elementem są oczywiście kolejne (ponumerowane i zatytułowane) rozdziały rozbite na mniejsze podrozdziały (SUBSECTION). Właściwa treść pracy zawiera się w dopiero w poszczególnych akapitach (P), które z kolei mogą być w praktycznie dowolny sposób przeplatane ilustracjami (wykresy, schematy itp.) oraz przykładami (EXAMPLE). Ten ostatni element jest wygodnym rozwiązaniem chociażby w przypadku konieczności umieszczenia w pracy fragmentów kodów programów (choćby listingów XML!) jak również może on służyć do oznaczania cytatów w pracy literaturoznawczej.

W akapity mogą być wplatane ponumerowane przypisy (FOOTNOTE) a szczególnie ważne stwierdzenia, terminy czy hasła są wyróżnione w ele-

mencie ACCENT. Bibliografia może zawierać zarówno źródła internetowe (INT_REF) jak i pozycje drukowane (PRINT_REF). Ostatnim elementem jest opcjonalny dodatek, którego zawartość jest trudna do przewidzenia stąd, uwzględniono możliwość umieszczenia w nim zarówno tekstu, ilustracji jak i różnych przykładów.

4.5. DRAMATY SZEKSPIRA

W tej części przedstawiony został schemat dokumentu (play.dtd) pozwalający zapisać sztuki Williama Szekspira w postaci plików XML[5].

```
<!ELEMENT PLAY          (TITLE, FM, PERSONAE,
                         SCNDESCR, PLAYSUBT, INDUCT?,
                         PROLOGUE?, ACT+, EPILOGUE?)>
<!ELEMENT TITLE         (#PCDATA)>
<!ELEMENT FM            (P+)>
<!ELEMENT P             (#PCDATA)>
<!ELEMENT PERSONAE      (TITLE, (PERSONA | PGROUP)+)>
<!ELEMENT PGROUP        (PERSONA+, GRPDESCR)>
<!ELEMENT PERSONA       (#PCDATA)>
<!ELEMENT GRPDESCR      (#PCDATA)>
<!ELEMENT SCNDESCR      (#PCDATA)>
<!ELEMENT PLAYSUBT      (#PCDATA)>
<!ELEMENT INDUCT        (TITLE, SUBTITLE*,
                         (SCENE+| (SPEECH | STAGEDIR |
                         SUBHEAD)+))>
<!ELEMENT ACT           (TITLE, SUBTITLE*, PROLOGUE?,
                         SCENE+, EPILOGUE?)>
<!ELEMENT SCENE         (TITLE, SUBTITLE*,
                         (SPEECH | STAGEDIR |
                         SUBHEAD)+)>
<!ELEMENT PROLOGUE      (TITLE, SUBTITLE*, (STAGEDIR |
                         SPEECH)+)>
<!ELEMENT EPILOGUE      (TITLE, SUBTITLE*, (STAGEDIR |
                         SPEECH)+)>
<!ELEMENT SPEECH        (SPEAKER+, (LINE | STAGEDIR |
                         SUBHEAD)+)>
<!ELEMENT SPEAKER       (#PCDATA)>
<!ELEMENT LINE          (#PCDATA | STAGEDIR)*>
<!ELEMENT STAGEDIR      (#PCDATA)>
```

[5] Kod stworzony przez Jona Bosaca dostępny jest wraz z kolekcją dzieł Szekspira pod adresem http://www.oasis-open.org/cover/bosakShakespeare200.html.

```
<!ELEMENT SUBTITLE      (#PCDATA)>
<!ELEMENT SUBHEAD       (#PCDATA)>
```

Każdą sztukę rozpoczyna element podstawowy *PLAY*. Następnie przedstawiany jest tytuł (element TITLE) i uwagi autora dotyczące praw autorskich (paragrafy *P* wewnątrz elementu *FM*). Dalej następuje przedstawienie wszystkich postaci występujących w sztuce (PERSONAE) bądź pojedynczo (PERSONA), bądź jako grupa postaci (PGROUP) wraz z określeniem ich roli (GRPDESCR). Później opisywane jest miejsce akcji (SCNDESCR), podawany jest tytuł sztuki (PLAYSUBT) i następują kolejne akty (ACT). Każdy akt składa się z tytułu (ewentualnie podtytułów i prologu) oraz poszczególnych scen (SCENE) i może kończyć się epilogiem (EPILOGUE). Na każdą scenę składają się wypowiedzi (SPEECH) poszczególnych postaci (SPEAKER), którym może towarzyszyć pewien tytuł (SUBHEAD) np. nazwa śpiewanej pieśni. W każdej chwili mogą wystąpić didaskalia (STAGEDIR). Z kolei każda wypowiedź podzielona jest na wersy (LINE).

Pierwszy akt może poprzedzać kilkuscenowe wprowadzenie (strukturalnie zbudowane podobnie jak akt), bądź prolog złożony z wypowiedzi różnych osób przeplatanych didaskaliami. Struktura epilogu jest identyczna jak prologu.

Takie ustrukturyzowane przedstawienie utworu dramatycznego pozwala praktycznie na zapisanie dowolnej sztuki Szekspira bez ingerencji w oryginalny tekst. Poniżej przedstawiono niewielki fragment komedii „Wszystko dobre, co się dobrze kończy".

```
<?xml version="1.0"?>
<!DOCTYPE PLAY SYSTEM "play.dtd">
<PLAY>
   <TITLE>All's Well That Ends Well</TITLE>

   <FM>
   <P>The XML markup in this version is Copyright &#169; 1999 Jon
Bosak.
   This work may freely be distributed on condition that it not be
modified or altered in any way.</P>
   </FM>

<PERSONAE>
   <TITLE>Dramatis Personae</TITLE>
      <PERSONA>KING OF FRANCE</PERSONA>
      <PERSONA>DUKE OF FLORENCE</PERSONA>
      <PERSONA>BERTRAM, Count of Rousillon.</PERSONA>
      <PERSONA>LAFEU, an old lord.</PERSONA>
      <PERSONA>PAROLLES, a follower of Bertram.</PERSONA>
   <PGROUP>
      <PERSONA>Steward</PERSONA>
      <PERSONA>Clown</PERSONA>
```

```
    <GRPDESCR>servants to the Countess of Rousillon.</GRPDESCR>
    </PGROUP>
       <PERSONA>A Page. </PERSONA>
       <PERSONA>COUNTESS OF ROUSILLON, mother to Bertram.
       </PERSONA>
       <PERSONA>HELENA, a gentlewoman protected by the Countess.
       </PERSONA>
       <PERSONA>An old Widow of Florence. </PERSONA>
       <PERSONA>DIANA, daughter to the Widow.</PERSONA>
    <PGROUP>
       <PERSONA>VIOLENTA</PERSONA>
       <PERSONA>MARIANA</PERSONA>
       <GRPDESCR>neighbours and friends to the Widow.</GRPDESCR>
    </PGROUP>
       <PERSONA>Lords, Officers, Soldiers, &amp;amp;c.,
       French and Florentine.</PERSONA>
    </PERSONAE>

<SCNDESCR>SCENE  Rousillon; Paris; Florence; Marseilles.
</SCNDESCR>

<PLAYSUBT>ALL'S WELL THAT ENDS WELL</PLAYSUBT>

<ACT>
    <TITLE>ACT I</TITLE>
    <SCENE><TITLE>SCENE I. Rousillon. The COUNT's palace.</TITLE>
    <STAGEDIR>Enter BERTRAM, the COUNTESS of Rousillon,
     HELENA, and LAFEU, all in black</STAGEDIR>

<SPEECH>
<SPEAKER>COUNTESS</SPEAKER>
<LINE>In delivering my son from me, I bury a second husband.
</LINE>
</SPEECH>

<SPEECH>
<SPEAKER>BERTRAM</SPEAKER>
<LINE>And I in going, madam, weep o'er my father's death</LINE>
<LINE>anew: but I must attend his majesty's command, to</LINE>
<LINE>whom I am now in ward, evermore in subjection.</LINE>
</SPEECH>
(...)
</ACT>
(...)
</PLAY>
```

Rozdział 5

STANDARDY STOWARZYSZONE

W poprzednich częściach pracy skupiono się na pokazaniu tych cech XML-a, które czynią go niezwykle użytecznym rozwiązaniem przeznaczonym do strukturalizacji różnego rodzaju danych. Charakterystyka ta wywodziła się wprost z konstrukcji samego języka i w zasadzie nie wykraczała poza ramy określone przez specyfikację. Jednak mówiąc o użyciu tego standardu w Internecie należy omówić pewne dodatkowe narzędzia umożliwiające wzbogacanie dokumentów w cechy niezbędne do ich publikacji w sieci. Są nimi pewne towarzyszące standardy, najczęściej zdefiniowane w XML-u, pozwalające tworzyć połączenia hipertekstowe, oraz opisywać wygląd poszczególnych dokumentów.

5.1. PRZESTRZENIE NAZW XML

Będąca przedmiotem niniejszej części pracy specyfikacja dotycząca tzw. **przestrzeni nazw**[1] (*ang.* ***namespaces)*** w XML-u nie jest wprawdzie integralną częścią tego języka rozszerzającą obszar jego zastosowań, ale stanowi niezbędną podstawę do użycia omawianych w dalszej części standardów XLink i XSL.

Każdy dokument XML można przedstawić w postaci drzewa wzajemnie zagnieżdżających się elementów o określonych typach (nazwach) i towarzyszących im atrybutach. Ponieważ nie istnieje predefiniowany zbiór tagów, nazwy elementów są tworzone praktycznie dowolnie przez autorów poszczególnych dokumentów. James Clark nazywa je **nazwami lokalnymi**[2]

[1] World Wide Web Consortium: *Namespaces in XML. W3C recommendation.* http://www.w3.org/TR/1999/REC-xml-names-19990114.

[2] J. Clark: *XML namespaces*. http://www.jclark.com/xml/xmlns.htm.

(*ang.* ***local names***), ponieważ zakres ich zastosowania i kontekst, w jakim mają sens, nie wykracza poza dany dokument. Łatwo wyobrazić sobie sytuację, w której np. element TITLE raz może opisywać tytuł książki a w innym rozwiązaniu odnosić się do tytułu naukowego autora (np. prof.). W efekcie może to prowadzić do konfliktów nazw, szczególnie w przypadku tworzenia dokumentów korzystających z elementów i atrybutów pochodzących z różnych schematów DTD, a ponadto znacznie utrudnia pisanie oprogramowania przetwarzającego dokumenty XML.

Rozwiązaniem tego problemu jest wzbogacenie modelu danych poprzez określenie kontekstu używanych nazw elementów i atrybutów, którego dokonuje się kojarząc nazwę z odpowiednim adresem URL. W ten sposób można używać zarówno elementu TITLE należącego do przestrzeni nazw związanej z opisem książki, jak i elementu o tej samej nazwie oznaczającego stopień naukowy, lecz powiązanego z innym adresem URL. Kombinacja nazwy z adresem określana jest jako **nazwa uniwersalna**[3] (*ang.* ***universal name***). Przykład deklaracji przestrzeni nazw *book*, do której należy element TITLE:

```
<book:TITLE xmlns:book="http://www.books.com/xml/">
```

W przykładzie powyższym deklarowana jest przestrzeń nazw book (xmlns:book), którą wiąże się z konkretnym adresem URL[4] (np.: http://www.books.com/xml/), a następnie element TITLE przypisuje się do zadeklarowanej przestrzeni (book:TITLE). Formalnie rzecz ujmując, dokonuje się tu mapowanie lokalnej nazwy elementu pochodzącego z drzewa XML do drzewa, gdzie staje się ona nazwą uniwersalną. Proces ten wymaga stosowania prefiksów. Jeśli nazwa elementu, bądź atrybutu zawiera symbol dwukropka, to część nazwy poprzedzająca ten znak traktowana jest jako prefiks odnoszący się do odpowiedniego adresu przestrzeni nazw, natomiast nazwa występująca za dwukropkiem uznawana jest za nazwę lokalną.

W analogiczny sposób mechanizm ten działa dla atrybutów:

```
<NAME  staff:SECTION ="IT"
 xmlns:staff =" http://www.company.com/">
 Jan Kowalski
</NAME>
```

W przypadku zadeklarowania domyślnej przestrzeni nazw na początku dokumentu, wszystkie występujące w nim tagi bez prefiksów są traktowane jako należące do tej przestrzeni.

---

[3] ibid.

[4] Jedyną rolą adresów URL jest umożliwienie aplikacjom prawidłowe rozpoznanie kontekstu nazwy. Specyfikacja nie określa natomiast, co ma się kryć za tymi adresami. Praktycznym powodem, który skłonił W3C do użycia URL jako identyfikatorów jest powszechność ich użycia w Internecie oraz unikatowość opisywanych domen. T. Bray: *XML namespaces by example*. http://www.xml.com/xml/pub/1999/01/namespaces.html.

5.2. XLINK

Opracowywana specyfikacja XML Linking Language (XLink)[5] określa sposób tworzenia i opisywania połączeń pomiędzy zasobami[6] w dokumentach XML. W odróżnieniu od mechanizmów znanych z HTML-a daje ona możliwość łączenia więcej niż dwóch elementów, opisywania połączeń metadanymi oraz tworzenia baz danych linków zupełnie oddzielonych od łączonych zasobów.

Podobnie jak w większości innych standardów, także i tu obowiązującą technologią adresowania zasobów jest URL. Użycie elementów i atrybutów XLinka wymaga deklaracji przestrzeni nazw XLink, którą definiuje się używając adresu http://www.w3.org/1999/xlink. Przykład:

```
<nazwa_elementu
xmlns:xlink="http://www.w3.org/1999/xlink">
(...)
</nazwa_elementu>
```

W HTML-u hiperłącze mogło być opisane jedynie tagiem <A>. XML znosi to ograniczenie, gdyż tu każdy element może **być elementem łączącym** (*ang.linking element*). Stanowi o tym dodanie specjalnego atrybutu **xlink:type**, który może przybierać następujące wartości:

1) **simple,**
2) **extended,**
3) **locator,**
4) **arc,**
5) **resource,**
6) **title.**

XLink oferuje dwa typy linków: **proste** (*ang. **simple link***) i **rozszerzone** (*ang. **extended link***). Link prosty łączy zawsze dokładnie dwa zasoby: jeden lokalny i jeden odległy. Zasób określa się jako lokalny, gdy pewien fragment jego zawartości, bądź też całość stanowi element łączący. Natomiast zasobem odległym jest zasób znajdujący się poza linkiem, do którego ten jednoznacznie prowadzi. Połączenie, w którym występuje co najmniej jeden zasób lokalny nazywane jest **linkiem wplatanym** (*ang. **inline link***). Każdy link prosty jest linkiem inline.

---

[5] Bazą do napisania niniejszego podrozdziału była najnowsza wersja robocza z 21 lutego 2000 roku. World Wide Web Consortium: *XML Linking Language (XLink). W3C working draft.* http://www.w3.org/TR/2000/WD-xlink-20000221.

[6] Zasób jest zdefiniowany jako adresowalna jednostka informacji (np.: plik, grafika, dokument, program itp.).

Połączenia proste podobne są do linków znanych z HTML-a, choć można je opisać w dużo dokładniejszy sposób, z użyciem kilku parametrów niedostępnych w tamtym języku. Przykład linku prostego wskazującego na dołączony plik graficzny:

```
<IMAGE xmlns:xlink="http://www.w3.org/1999/xlink"
   xlink:type="simple" xlink:href="http://www.serwer/images/picture.gif"
   xlink:actuate="onLoad" xlink:show="embed"/>
```

Zauważmy, że każdy atrybut elementu IMAGE rozpoczyna się od prefiksu xlink, po którym występuje jego nazwa. Oznacza to, że wszystkie te atrybuty należą do przestrzeni nazw XLink, której adres jest wskazany wartością atrybutu xmlns:xlink. Deklaracja ta powinna wystąpić albo w elemencie podstawowym, bądź też przy poszczególnych elementach korzystających z prefiksu xlink. Powyższy przykład w DTD wyglądałby następująco:

```
<!ELEMENT IMAGE EMPTY>
<!ATTLIST IMAGE  xmlns:xlink  CDATA  #FIXED
"http://www.w3.org/1999/xlink"
              xlink:type    (simple)  #FIXED "simple"
              xlink:href     CDATA  #REQUIRED
              xlink:show    (new | replace | embed | undefined)
             #FIXED "onLoad"
              xlink:actuate (onLoad | onRequest | undefined)  #FIXED
               "embed">
(...)
<IMAGE xlink:href="http://www.serwer/images/picture.gif"/>
```

W przedstawionych przykładach pierwszy atrybut xlink:type definiuje połączenie jako link prosty. Z kolei wartością atrybutu xlink:href jest w tym wypadku bezwzględny adres URL (wartość np.„picture.gif" wskazywałaby na plik znajdujący się w tym samym katalogu, co dokument zawierający link).

Kolejne dwa atrybuty określają sposób połączenia bieżącego dokumentu z odległym zasobem. xlink:show opisuje jak należy wyświetlić jego zawartość. Możliwe są tu cztery opcje:

1) **replace** – zastępuje zawartość okna czy też ramki, w której wyświetlany jest bieżący dokument zawartością docelowego zasobu (najczęściej tak wygląda domyślne zachowanie przeglądarek dla linków HTML).
2) **new** – otwiera nowe okno i wyświetla docelowy dokument.
3) **embed** – wstawia zawartość dołączanego dokumentu do bieżącego okna.
4) **undefined** – w tym wypadku zachowanie zależy od aplikacji przetwarzającej dokument.

Następny atrybut, **xlink:actuate** opisuje w jakich warunkach następuje aktywacja połączenia a wartości, jakie może on przyjmować to:

1) **onRequest** – aktywacja tylko na wyraźne żądanie użytkownika (np. kliknięcie elementu).
2) **onLoad** – aktywacja w momencie, gdy element łączący zostaje załadowany.
3) **undefined** – zachowanie zależne od aplikacji przetwarzającej.

Opisanie elementu atrybutami Xlinka, a więc uczynienie go hiperłączem nie wyklucza możliwości dodania do niego innych atrybutów (w prezentowanym przykładzie mogą to być parametry określające np. wielkość czy format obrazka). Ponadto element łączący może mieć własnych potomków, którzy także są elementami łączącymi.

Prezentowane do tej pory linki proste łączą zawsze dwa elementy i są połączeniami jednokierunkowymi. Bardziej zaawansowanymi mechanizmami są połączenia rozszerzone, składające się ze zbioru zasobów zarówno lokalnych (stanowiących część elementu rozszerzonego połączenia) jak i odległych oraz połączeń między nimi. Połączenia te mogą być wielokierunkowe, co oznacza, że każdy zasób wchodzący w skład takiego połączenia może być zarówno celem, do którego prowadzi link, jak i źródłem, z którego link jest aktywowany. Może również jednocześnie stanowić jedno i drugie.

Oto przykład linku wskazującego na pewien serwis oraz jego różne lokacje:

```
<WEBSITE xmlns:xlink="http://www.w3.org/1999/xlink"
         xlink:type="extended" xlink:title="Serwis WWW ">
<NAME    xlink:type="resource" xlink:role="source">Mój
         serwis </NAME>
<HOMESITE xlink:type="locator"
         xlink:href="http://www.warszawa /gz/index"
         xlink:role="default"/>
<MIRROR  xlink:type="locator" xlink:title="Mój serwis
         w Krakowie"
         xlink:role="kr"      xlink:href="http://www.
                              krakow/gz/index "/>
<MIRROR  xlink:type="locator" xlink:title="Mój serwis w Gdańsku"
         xlink:role="gd" xlink:href="http://www.gdansk/gz/
         index"/>
</WEBSITE>
```

Struktura przedstawionego linku jest następująca: element WEBSITE definiowany jest jako połączenie rozszerzone (xlink:type="extended") oraz określane są jego dwa parametry tj. „title" i „role". Pierwszy z nich jest stosowany w celu krótkiego scharakteryzowania w sposób czytelny dla człowieka funkcji, jaką pełni zasób w danym połączeniu. Nie jest on wymagany, często przyjmuje wartość elementu TITLE dołączanego zasobu. Atry-

but "role" także opisuje funkcje zasobu w danym połączeniu, lecz jest przeznaczony dla aplikacji przetwarzającej dany dokument.

Link zawiera cztery zasoby: jeden lokalny (tekst „Mój serwis") oraz trzy odległe (na serwerze domowym w Warszawie i dwóch mirrorach w Krakowie i Gdańsku). Jak widać, lokalizacji zasobów lokalnych dokonuje się określając ich typ jako **„resource"**, natomiast zasoby odległe istniejące poza elementem łączącym (najczęściej w innym dokumencie) opisywane są terminem **„locator"**. Każdy z zasobów także może być opisany semantycznymi atrybutami „title" i „role". W opisywanym przypadku mamy do czynienia z linkiem inline, ponieważ jeden z jego zasobów jest zasobem lokalnym. Gdyby żaden z elementów rozszerzonego połączenia nie był określony jako typ „resource" mówilibyśmy wtedy o połączeniu **autonomicznym** (*ang.* ***out-of-line link***).

Połączenia out-of-line cechuje fizyczne odseparowanie od zasobów, które łączą. Umożliwia to przechowywanie ich w osobnych dokumentach nazywanych bazami połączeń (*ang.* ***linkbase***). Rozwiązanie takie pozwala na łączenie się z zasobami bez konieczności ingerencji w kod źródłowy zarówno dokumentu łączącego jak i dołączanego.

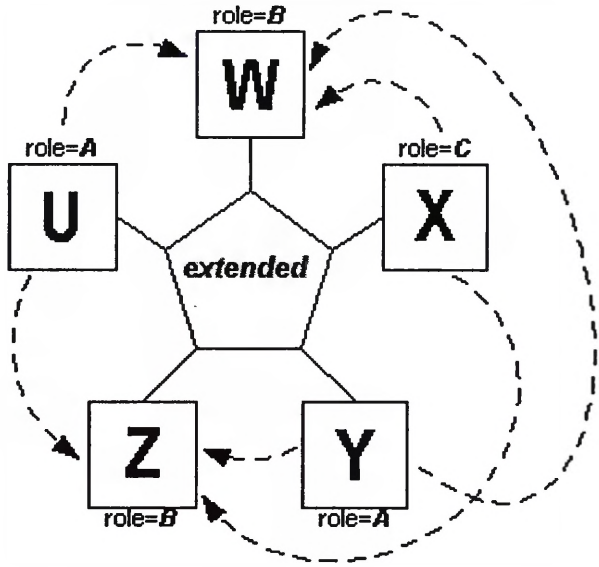
W terminologii informatycznej rozszerzony link nazywany jest „grafem zorientowanym[7] , w którym każdy związek pomiędzy parą wierzchołków reprezentuje kolejne połączenie między poszczególnymi zasobami"[8]. Rozszerzone połączenia pozwalają, aby jeden zasób był zarówno celem, do którego link prowadzi jak i źródłem, z którego wychodzi. Pamiętając, że link jest grafem, zauważmy, że w poprzednim przykładzie brakowało zdefiniowanych reguł, które opisywałyby drogi pomiędzy poszczególnymi wierzchołkami (*ang.* ***traversal***), czyli inaczej mówiąc zasoby tam występujące były połączone bez określonego kierunku i kolejności.

Załóżmy, że chcemy przedstawić wszystkie połączenia między trzema zasobami (A, B, C) rozszerzonego linku. W wyniku otrzymujemy zbiór par: A-A, A-B, A-C, B-A, B-B, B-C, C-A, C-B, C-C. Każde z tych połączeń może charakteryzować się innymi własnościami, co więcej niektóre z nich mogą być w danym przypadku zabronione. Opis tworzenia połączeń zawiera się w elemencie z atrybutem xlink:type o wartości **„arc"** (łuk). Najczęściej towarzyszą mu atrybuty „actuate" i „show" oznaczające to samo, co w połączeniach prostych. Ponadto każdy łuk opisany jest dwoma dodatkowymi parametrami tj. **xlink:from i xlink:to**. Wartości tych atrybutów odpowiadają wartościom atrybutów „role" elementów „locator" i „resource", z których połączenie jest inicjowane (*from*) i dokąd ono prowadzi (*to*). Jeśli istnieje jeden lub więcej zasobów, których wartością atrybutów xlink:role jest np. a (powiedzmy, że połączenie ma prowadzić z a (xlink:from="a") do b (xlink:to="b")), to łuk prowadzi z każdego zasobu o tej wartości do każde-

[7] Graf, w którym kolejność elementów tworzących pary wierzchołków jest określona.
[8] E. R. Harold: *XML bible*. IDG Books Worldwide 1999.

go zasobu, którego atrybut xlink:role jest równy "b". Na diagramie poniżej (rys. 2) przedstawiono schemat połączeń między pięcioma zasobami (U, W, X, Y, Z) rozszerzonego linku, w którym poszczególne łuki zostały zdefiniowane w następujący sposób:

```
<!-- go  - element z parametrem typu "arc"-->
<go xlink:from="A" xlink:to="B" />
<go xlink:from="C" xlink:to="B"/>
```



Rys. 2

Wracając do przykładu z serwisem www umieszczonym na różnych serwerach, można go tak zmodyfikować, aby zróżnicować zachowanie połączeń w zależności od dowolnej, wybranej cechy np.:

```
<WEBSITE xmlns:xlink="http://www.w3.org/1999/xlink"
         xlink:type="extended" xlink:title="Serwis WWW ">
  <NAME xlink:type="resource" xlink:role="source">Mój serwis</
   NAME>
  <HOMESITE xlink:type="locator"
            xlink:href="http://www.warszawa/gz/index" xlink:role"hp"/>
<MIRROR xlink:type="locator" xlink:title="Mój serwis w Krakowie"
    xlink:role="mirror"  xlink:href="http://www.krakow/gz/index"/>
```

```
<MIRROR xlink:type="locator" xlink:title="Mój serwis w Gdańsku"
    xlink:role="mirror" xlink:href="http://www.gdansk/gz/index"/>
<CONNECTION xlink:type="arc" xlink:from="source"
        xlink:to="mirror" xlink:show="replace"
        xlink:actuate="onRequest" xlink:title="Mirror Page"/>
<CONNECTION xlink:type="arc" xlink:from="source"
        xlink:to="hp" xlink:show="replace"
        xlink:actuate="onRequest" xlink:title="Home Page"/>
</WEBSITE>
```

W tym przykładzie wartość „mirror" parametru „role" jest współdzielona przez dwa zasoby tj. strony na serwerach w Krakowie i Gdańsku. Elementy łączące typu „arc" (CONNECTION) definiują dwa typy połączeń: jedno z mirrorami, któremu nadano tytuł „Mirror Page" i drugie ze stroną domową (poprzez parametr role o wartości „hp") nazwane „Home Page".

Jak widać, wartości zarówno parametru „role" jak i „source" mogą się powtarzać w różnych elementach. W przypadku braku jawnie zdefiniowanego źródła i kierunku połączenia (tj. braku atrybutów „from" i „to") zakładaną, domyślną procedurą jest rozpatrzenie wszystkich zadeklarowanych w linku wartości „role".

Oto tabela (tab. 3) podsumowująca jakie atrybuty (wiersze) mogą (wartość O) lub muszą (wartość +) występować z poszczególnymi typami elementów XLinka (kolumny).

Tabela 3

	simple	extended	locator	arc	resource	title
type	+	+	+	+	+	+
href	0		+			
role	0	0	0	0	0	
title	0	0	0	0	0	
show	0			0		
actuate	0			0		
from				0		
to				0		

Należy tu dodać, że w chwili obecnej nie istnieje żadna przeglądarka WWW, która oferowałaby wsparcie dla standardu Xlink.

5.3. XPOINTER

O ile standard XLink określał sposób łączenia ze sobą wielu zasobów, to XML Pointer Language (XPointer)[9] opisuje schemat adresowania poszczególnych części dokumentów XML. XPointer jest zbudowany na bazie specyfikacji XPath[10], która modeluje dokument XML jako strukturę drzewiastą złożoną z zagnieżdżających się **węzłów**[11] *(ang. **node**)*. XPath dostarcza zestawu ogólnych wyrażeń, pozwalających na identyfikację dowolnego węzła lub zbioru węzłów w strukturze dokumentu XML. Dokonuje się tego wykorzystując kombinacje konstrukcji zwanych osiami[12] (*ang. **axes***) i **predykatami**[13] (*ang. **predicates***). W skrócie można powiedzieć, że osie dokonują wstępnego wyboru węzłów w jakiś sposób powiązanych z węzłem wejściowym (np. wybierając jego potomków, rodzeństwo, atrybuty itp.), natomiast predykaty dokonują ich ostatecznej selekcji.

Wskaźniki XPointer stanowią abstrakcyjny język, dający dostęp do dowolnego miejsca w dokumencie, nawet jeśli autor nie zaznaczył go specjalnym identyfikatorem (w HTML-u ominięcie sekwencji „#nazwa" było niemożliwe). Wybranie odpowiedniego elementu może odbywać się z użyciem jego nazwy, typu, kolejności występowania czy relacji w stosunku do innych elementów. Poza tym możliwe jest wskazanie nie tylko na pojedynczy punkt, ale i na cały zbiór czy też zakres elementów.

Podstawową konstrukcją używaną przez XPointera są **ścieżki dostępu** (*ang. **location paths***) złożone z kolejnych **kroków dostępu** (*ang. **location steps***), określających miejsce docelowe, do którego odnosi się wskaźnik. Rozpatrywane są one w odniesieniu do węzła kontekstowego (ang. context node), którym jest znane miejsce w dokumencie (najczęściej początek dokumentu bądź inny krok dostępu). Schematycznie konstrukcja kroku dostępu wygląda następująco:

oś::węzeł_rozważany[predykat]

Oś określa kierunek poszukiwań (np. szukaj tylko wśród potomków danego węzła kontekstowego). Węzeł rozważany mówi, które węzły występujące wzdłuż danej osi wybierać (najczęściej jest to nazwa konkretnego

---

[9] Rozdział oparty na najnowszej wersji roboczej standardu z dnia 6-go grudnia 1999 roku. World Wide Web Consortium: *XML Pointer Language (XPointer)*. W3C working draft. http://www.w3.org/TR/1999/WD-xptr-19991206.

[10] World Wide Web Consortium: *XML Path Language (XPath) version 1.0. W3C recommendation*. http://www.w3.org/TR/1999/REC-xpath-19991116.

[11] Węzeł – obiekt, stanowiący kompletny zbiór informacji o określonym elemencie składowym (element, atrybut, instrukcja przetwarzania, komentarz, pełny blok parsowanych danych znakowych) dokumentu XML.

[12] Oś – zarezerwowana nazwa określająca sekwencję wyodrębnionych danych.

[13] Predykat – wyrażenie służące do odfiltrowania określonych zasobów według żądanych kryteriów.

elementu). Opcjonalny predykat jest wyrażeniem boolowskim, testującym każdy węzeł w wynikowym zbiorze. Poszczególne kroki dostępu złożone na ścieżkę dostępu oddzielane są znakiem slasha (/).

Przykład (5.3a). Załóżmy, że mamy poprawnie uformowany dokument XML opisujący uczestników konferencji na temat zastosowań języka XML.

```
<?xml version="1.0" encoding="windows-1250"?>
<CONFERENCE>
   <TOPIC>Obszary zastosowań XML-a</TOPIC>
   <PARTICIPANTS NR="3">
            <PARTICIPANT NATIONALITY="pl" ID="1">
                  <NAME>Jan Kowalski</NAME>
                   <COMPANY>Firma A</COMPANY>
             </PARTICIPANT>
             <PARTICIPANT NATIONALITY="pl" ID="2">
                   <NAME>Tomasz Nowak</NAME>
                   <COMPANY>Firma B</COMPANY>
      </PARTICIPANT>
     <PARTICIPANT NATIONALITY="eng" ID="3">
                  <NAME>John Smith</NAME>
                  <COMPANY>C Company</COMPANY>
      </PARTICIPANT>
   </PARTICIPANTS>
</CONFERENCE>
```

Wskazanie elementu będącego tematem konferencji może wyglądać następująco:

xpointer(/child::CONFERENCE/child::TOPIC)

gdzie: „/child::CONFERENCE" – oznacza wybór wszystkich potomków **węzła podstawowego**[14] (*ang.* ***root node***) o nazwie CONFERENCE a „child:: TOPIC" to selekcja wszystkich potomków elementu CONFERENCE (węzeł kontekstowy) o nazwie TOPIC.

XPointer używa 12 rodzajów osi oznaczających:

1. **ancestor** – wybór wszystkich rodziców węzła kontekstowego, rodziców rodziców węzła kontekstowego itd. aż po węzeł podstawowy,
2. **ancestor-or-self** – wybór węzła kontekstowego i wszystkich węzłów, które go pośrednio lub bezpośrednio zawierają,

[14] Węzeł podstawowy nie jest tożsamy z elementem podstawowym. Jest to abstrakcyjna konstrukcja obejmująca cały dokument XML, łącznie z deklaracją XML, DTD, komentarzami, instrukcjami przetwarzającymi występującymi przed i wewnątrz elementu podstawowego. Węzeł ten w skrócie oznaczany jest znakiem slash (/).

3. **attribute** – wybór atrybutów węzła kontekstowego np.: ***PARTICIPANT/attribute::NATIONALITY*** oznacza wybór atrybutu NATIONALITY elementu PARTICIPANT. „attribute::" można zastąpić znakiem „@",

4. **child** – wybór bezpośrednich potomków węzła kontekstowego. W wypadku braku jawnie zadeklarowanej osi, XPointer uznaje tą oś za domyślną,

5. **descendant** – wybór wszystkich, bezpośrednich i pośrednich potomków węzła kontekstowego. Skrócona forma zapisu tej osi to „//" np.: **//NAME** wskazuje bezpośrednio na wszystkie elementy NAME w dokumencie,

6. **descendant-or-self** – wybór węzła kontekstowego i jego wszystkich bezpośrednich i pośrednich potomków,

7. **following** – wybór wszystkich elementów, występujących kolejno po elemencie kontekstowym bez względu na ich hierarchię. Przykład: **//TOPIC/following::*[position()=3]** oznacza: znajdź wszystkie elementy TOPIC (w naszym przykładzie jeden), następnie biorąc pod uwagę wszystkie kolejne elementy (gwiazdka oznacza „dowolny element") znajdź ten, który jest trzeci w kolejności (predykat „position()=3"). Wynikiem jest węzeł z nazwiskiem Jana Kowalskiego,

8. **following-sibling** – wybór wszystkich węzłów występujących po węźle kontekstowym mających tego samego bezpośredniego rodzica, co węzeł kontekstowy,

9. **parent** – wybór bezpośredniego rodzica węzła kontekstowego. W skrócie oznaczany „..",

10. **preceding** – wybór wszystkich węzłów (bez względu na hierarchię) występujących przed węzłem kontekstowym,

11. **preceding-sibling** – wybór wszystkich węzłów przed węzłem kontekstowym, mających tego samego rodzica,

12. **self** – wybór węzła kontekstowego.

Węzły rozważane, oprócz podania nazwy elementu można wybierać także w inny sposób:

1. * – wybór wszystkich węzłów elementowych (bez atrybutów, komentarzy itd.),

2. **node()** – wybór wszystkich węzłów,

3. **text()** – wybór węzłów tekstowych tj. parsowanych danych znakowych, stanowiących zawartość dowolnego elementu,

4. **comment()** – wybór węzłów z komentarzami,

5. **processing-instruction()** – wybór węzłów z instrukcjami przetwarzania,

6. **point()** – wybór określonego miejsca w blokach tekstu, który może być np. sekcją CDATA, komentarzem czy zawartością pewnego elementu. Każdy punkt, na który można wskazać zawiera się pomiędzy dwoma węzłami lub dwoma znakami należącymi do zbioru parsowanych danych. Wybranie punktu np. przed pierwszą literą nazwiska Kowalski w zamieszczonym wyżej przykładzie wygląda następująco:

//NAME[position()=1]/child::text()/point()[position()=4]

Pierwszy punkt (oznaczany jako zero) występuje przed J, drugi między J a A, trzeci między A i N, czwarty między N a znakiem spacji i wreszcie szukany przez nas między spacją a K.

7. **range()** – XPointer dostarcza narzędzi do wyboru określonego zakresu z dokumentu. Wymaga to określenia ścieżek dostępu do punktu początkowego i końcowego zakresu i połączenia ich słowem kluczowym „**to**". Przykład:

xpointer(//PARTICIPANT[position()=1] to //PARTICIPANT[position()=3])

Sama funkcja range(), którą wywołuje się z parametrem będącym **zbiorem lokacji**[15] (*ang.* ***location-set***), zwraca wartość będącą zakresem odpowiadającym każdej z lokacji. Inaczej mówiąc dokonuje konwersji każdej lokacji na odpowiedni zakres. Funkcja **range-inside()** zwraca z kolei zawartość podanych lokacji (np.: jeśli lokacja jest elementem, to wynikiem działania tej funkcji jest zawartość elementu, lecz nie on sam). Funkcjami, które w wyniku dają punkty bezpośrednio poprzedzające i następujące za wskazanymi lokacjami są **start-point()** i **end-point()**.

Wskazania na punkty w odpowiednim miejscu tekstu umożliwia funkcja **string-range()**. Przykład:

xpointer(string-range(//NAME, "Kowalski", 0, 8)

Oznacza to odnalezienie w zawartości wszystkich elementów NAME łańcucha znaków Kowalski, ustawienie punktu przed literą K i wybranie następujących po nim 8 znaków.

Jak już wspomniano, predykaty pozwalają dokonywać dokładniejszych wyborów wśród węzłów rozważanych wzdłuż danej osi. Wyrażenia te ujęte są w parę nawiasów kwadratowych „[]". Najczęściej spotykaną funkcją predykatową w wyrażeniach XPointer jest „**position()**", która zwraca indeks konkretnego węzła w odniesieniu do ich listy. Pozwala to wybierać pierwszy, drugi i kolejne indeksowane węzły a także używać różnych operatorów relacyjnych jak: „<", „>", „=", „!=", „<=", „>=". Przykład:

xpointer(/descendant::PARTICIPANT[position()=2]/following-sibling::*)

Wyrażenie to wskazuje na węzły z nazwiskiem Jana Nowaka i Firmą B. Kolejny przykład pozwala wybrać węzły z nazwiskami wszystkich Polaków uczestniczących w konferencji:

xpointer(//PARTICIPANT[@NATIONALITY="pl"]/NAME)

XPointer pozwala na używanie czterech funkcji zwracających zbiór węzłów:

[15] Uporządkowana lista węzłów dokumentu, punktów i/lub zakresów używanych przez wyrażenia XPointer.

id() – Funkcja zwraca ten element z dokumentu, który ma określoną wartość atrybutu ID. Zauważmy tu, że wynikiem jest nie tylko sam wskazany element, ale także jego wszyscy jego potomkowie[16]. Przykładowe wywołanie tej funkcji może wyglądać następująco:

http://www.serwer.pl/gz/conference.xml#xpointer(id("2"))

Fraza „xpointer(id(„2"))" może być zastąpiona przez podanie bezpośrednio po znaku „#" wartości atrybutu ID. Jedyną wadą funkcji id() jest konieczność odpowiedniego znakowania (występowanie atrybutu ID) w dołączanym dokumencie.

root() – Funkcja ta zwraca wartość węzła podstawowego wskazywanego dokumentu. Zapis „root()" może być zastąpiony znakiem pojedynczego slasha (/). Przykład:

http://www.serwer.pl/gz/conference.xml#xpointer(/)

here() – Funkcja ta jest najczęściej stosowana w wewnętrznych linkach tj. takich, które wskazują z jednego miejsca dokumentu do innej lokacji w tym samym dokumencie. Jej użycie jest niezbędne w wypadku wskazania na „element następny po bieżącym" i „rodzica elementu bieżącego". Załóżmy, że dokument zawiera pokaz kolejnych slajdów. Deklaracje:

here()/following::SLIDE[position()=1]
here()/preceding::SLIDE[position()=1]

wskazują kolejno na następny i poprzedni slajd.

origin() – Funkcja ta działa podobnie jak here(), z tym, że używana jest w połączeniach autonomicznych, gdzie sam link nie jest obecny w dokumencie źródłowym. Wskazuje ona na element w dokumencie źródłowym, z którego użytkownik aktywował link.

Standard XPointer, podobnie jak przedstawiany poprzedni XLink, także nie doczekał się jeszcze implementacji w przeglądarkach internetowych[17].

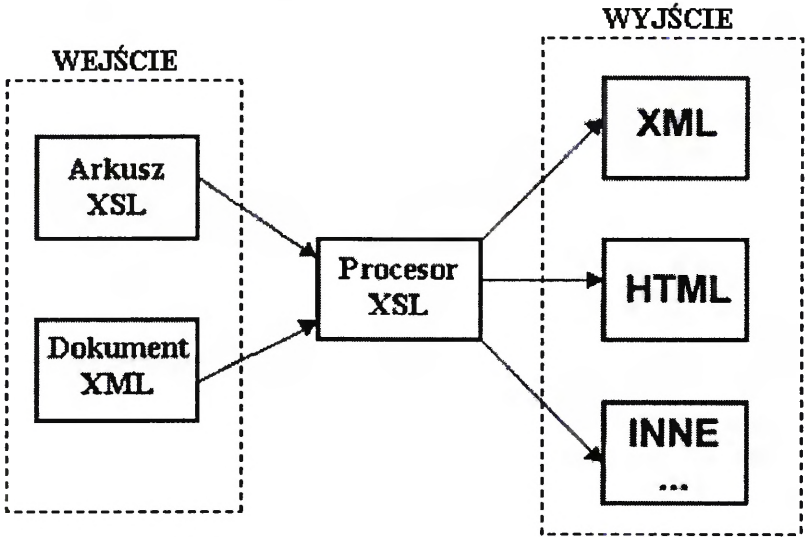
---

[16] Kiedy określa się cel w strukturze logicznej dokumentu używając narzędzi XPointera, istnieje niebezpieczeństwo wybrania połączenia do jego części, która wyrwana z kontekstu nie będzie miała sensu. Inaczej mówiąc, dołączany zbiór węzłów nie będzie poprawnie uformowanym XML-em. Niestety, część niestandardowych dokumentów XML nie dekomponuje się dobrze i w chwili obecnej nie istnieje żadne rozwiązanie tego problemu.

[17] Zbiór narzędzi, które w mniejszym lub większym stopniu radzą sobie ze standardami XLink i XPointer można znaleźć m. in. pod adresem http://www.xmlsoftware.com.

5.4. XSL

XML jest językiem służącym wyłącznie do organizowania danych w strukturalne schematy. Sam w sobie nie niesie żadnych informacji, co do sposobu ich prezentacji. Dlatego mówiąc o publikowaniu dokumentów XML-owych w Internecie należy przedstawić dodatkowe narzędzia, określające reguły nadawania tymże dokumentom odpowiedniego wyglądu. Opisuje je standard Extensible Stylesheet Language (XSL). W chwili obecnej XSL składa się z dwóch części: języka transformacji dokumentu XML (XSL Transformations[18]) i języka formatującego (XSL Formatting Objects[19]). XSLT definiuje reguły przekształcania dokumentów XML do innego (niekoniecznie XML-owego) schematu, który z kolei może, lecz nie musi używać np. elementów języka formatującego. Części te są więc w pewnym stopniu od siebie niezależne. Przetwarzanie informacji przy użyciu XSL-a przedstawiono na rysunku 3[20].



Rys. 3

[18] World Wide Web Consortium: *XSL Transformations (XSLT). Version 1.0. W3C Recommendation.* http://www.w3.org/TR/1999/REC-xslt-19991116.

[19] Ostatnia wersja robocza (World Wide Web Consortium: *Extensible Stylesheet Language (XSL). W3C working draft.* http://www.w3.org/TR/2000/WD-xsl-20000301.) wprowadza zamieszanie, gdyż pod tytułem XSL opisuje wyłącznie język formatujący jednocześnie zaznaczając, że w jego skład wchodzi również XSLT.

[20] Schemat za: A. Góralski: *XML – i co dalej?*, „Netforum" 1999 nr 6 s.29.

Procesor stylów XSL, korzystając z danych wejściowych w postaci pliku XML i arkusza XSL, dokonuje prezentacji zawartości dokumentu w sposób określony przez projektanta arkusza. Proces ten jest dwuetapowy: po pierwsze z wejściowego drzewa dokumentu XML tworzona jest nowa, wyjściowa struktura drzewiasta, której poszczególne elementy z kolei poddawane są odpowiedniej obróbce przystosowującej je do określonej prezentacji. Pierwszy proces nazywany jest transformacją, drugi to formatowanie. Rezultatem transformacji może być drzewo diametralnie różne od drzewa wejściowego, z danymi odfiltrowanymi i zreorganizowanymi w zupełnie nowych strukturach. Z kolei formatowanie opisuje jego węzły klasami wyrażeń nazwanymi **obiektami formatującymi** (*ang.* ***formatting object***), które określają abstrakcyjne konstrukcje typograficzne, takie jak tabela, strona, paragraf itp. Wszystkie te operacje dokonywane są na danych dostarczanych w pliku XML, istniejących niezależnie poza warstwą prezentacyjną. XSL stanowi więc platformę umożliwiającą odtwarzanie tej samej informacji wejściowej w różnych formatach wyjściowych za pomocą prostej zmiany dołączanego arkusza stylów.

Ponieważ prace standaryzacyjne nad częścią formatującą języka XSL dopiero trwają i nie istnieją obecnie żadne implementacje przedstawianych przez W3C propozycji, dlatego w dalszej części przedstawiony zostanie pokrótce XSLT, jako narzędzie do prezentacji dokumentów XML w sieci poprzez przekształcanie ich do formatu HTML. Proces ten może odbywać się na trzy różne sposoby:

1. Zarówno dokument XML jak i arkusz stylów są przesyłane przez serwer do klienta (tu przeglądarki WWW), który dokonuje transformacji (rozwiązanie to wykorzystywane jest przez Internet Explorera 5.0).
2. Serwer dokonuje transformacji dokumentu XML i plik wynikowy przesyła klientowi.
3. Niezależny program dokonuje transformacji a serwer i klient pracują wyłącznie z dokumentami wynikowymi.

W dalszej części skupię się na przedstawieniu pierwszego rozwiązania. Dokument XML z dołączonym arkuszem powinien zawierać w prologu zastępującą instrukcję:

<?xml-stylesheet type="text/xml" href="adres_pliku_XSL"?>

Plik XSL, będący również poprawnym dokumentem XML, powinien także rozpoczynać się od deklaracji XML, po której następuje określenie przestrzeni nazw XSL:

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

IE 5.0 nie stosuje się tu do zaleceń specyfikacji wymagając adresu http://www.w3.org/TR/WD-xsl oraz deklaracji typu text/xsl.

Działanie arkusza stylów XSLT, zbudowanego z odpowiednich **wzorców** (*ang.* ***pattern***), opiera się na ich **dopasowywaniu** (*ang.* ***matching***) do

kolejnych węzłów poruszając się w głąb struktury drzewa XML. Najczęściej spotykanym tagiem jest xsl:template definiujący pojedynczy wzorzec z parametrem match określającym węzły, do których niniejszy wzorzec się stosuje. Przykład arkusza XSL odnoszący się do dokumentu XML z poprzedniego podrozdziału:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
    <xsl:template match="/">
    <xsl:apply-templates select="CONFERENCE"/>
</xsl:template>
</xsl:stylesheet>
```

Jak widać wskazania węzłów (wartość atrybutu match) dokonuje się z użyciem wyrażeń standardu XPath[21] („/" odnosi się więc do węzła podstawowego). Tag xsl:apply-templates (bez atrybutu select) oznaczałby nakaz zastosowania odpowiednich wzorców do wszystkich podelementów elementu, do którego jest stosowany bieżący wzorzec. Select mówi, że dalsze wzorce zostaną zastosowane tylko do podelementu CONFERENCE.

Zadaniem elementu **xsl:value-of** jest pobranie wartości danego węzła (najczęściej wskazanego atrybutem select) i wstawienie jej do dokumentu wynikowego. Jeśli wskazanie to nie jest jednoznaczne (np.: select="PARTICIPANT" – ponieważ istnieją trzy takie elementy), można stosować instrukcję **xsl:for-each**, która przetwarza każdy element wybrany atrybutem select. Np.:

```
<xsl:template match="PARTICIPANTS">
    <xsl:for-each select="PARTICIPANT">
        <xsl:value-of select="."/>
    </xsl/for:each>
</xsl:template>
```

Wybierane są tu wartości wszystkich elementów o nazwie PARTICIPANT będących potomkami elementu PARTICIPANTS. Wartość atrybutu select („.") oznacza wybór wartości elementu, do którego stosuje się podany wzorzec (tu: PARTICIPANT).

Aby otrzymać wyjściowy plik otagowany znacznikami HTML-a można użyć następującego arkusza (5.4.a i 5.4.b):

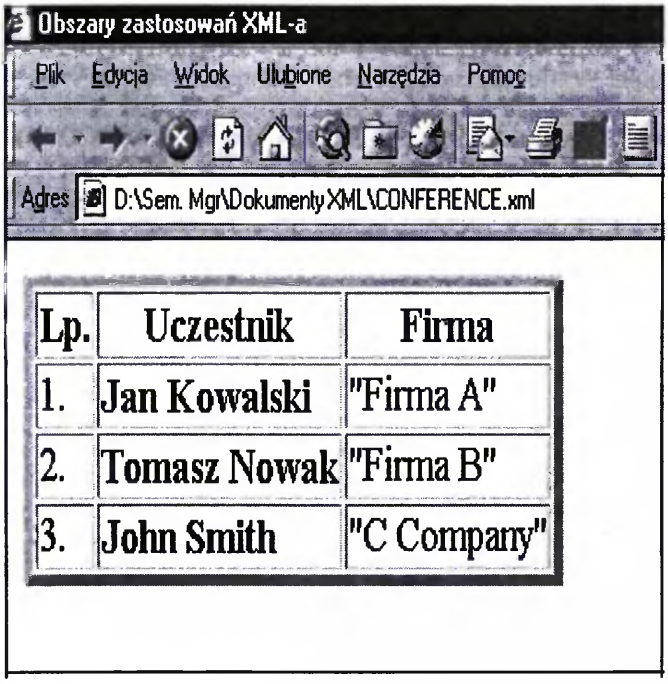
```
<?xml version="1.0" encoding="windows-1250"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
    <html><head>
        <title><xsl:value-of select="//TOPIC"/></title>
```

[21] Patrz też: 5.3. XPointer.

```
    </head><body>
        <xsl:apply-templates/>
        </body>
    </html>
</xsl:template>
<xsl:template match="text()">
        <xsl:value-of select="."/>
</xsl:template>

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

<xsl:template match="PARTICIPANTS">
        <TABLE BORDER="5"><TH>Lp.</TH>
                                <TH>Uczestnik</TH>
                                <TH>Firma</TH>
        <xsl:apply-templates/>
</TABLE>
</xsl:template>

<xsl:template match="PARTICIPANT">
    <tr>
    <td><xsl:value-of select="@ID"/>.</td>
    <td><B><xsl:value-of select="NAME"/></B></td>
    <td>"<xsl:value-of select="COMPANY"/>"</td>
    </tr>
</xsl:template>
</xsl:stylesheet>
```

Przetwarzanie pliku wejściowego rozpoczyna się zastosowania wzorca do podstawowego węzła dokumentu poprzez match=„/". Wzorzec ten wstawia podstawowe tagi dokumentu HTML. Do tytułu dokumentu zostaje wstawiona wartość elementu TOPIC. Ponadto instrukcja xsl:apply-template nakazuje zastosowanie odpowiednich wzorców do kolejnych podwęzłów. Wzorzec match="text()" służy do kopiowania fragmentów tekstu znajdujących się między tagami. Oznacza to, że kiedykolwiek szablon zostanie zastosowany do węzła tekstowego, jego zawartość zostanie zwrócona na wyjściu.

Dalej przetwarzane są kolejne węzły drzewa XML począwszy od elementu podstawowego (CONFERENCE) przez atrybuty (ID), aż po nie mające potomków NAME i COMPANY. Wybierane są ich poszczególne wartości i wstawione do HTML-owej tabeli. Otrzymany dokument jest prawidłowo renderowany przez przeglądarkę IE 5.0 (rys. 4).



Rys. 4

W powyższym przykładzie wszystkie elementy i atrybuty HTML zostały stworzone literalnie tj. poprzez napisanie otwierającego i zamykającego znacznika w wybranym punkcie arkusza. XSL udostępnia specjalne instrukcje, które pozwalają bezpośrednio umieścić w dokumencie wejściowym odpowiednie węzły. Używając xsl:element i xsl:attribute można stworzyć np. HTML-ową tabelę w następujący sposób:

```
<xsl:element name="TABLE">
        <xsl:attribute name="BORDER>5</xsl:attribute>
        <!-- tu dalsze instrukcje -->
</xsl:element>
```

Przedstawione powyżej sposoby transformacji XSL pokazują jedynie niewielką część jego rzeczywistych możliwości. Kopiowanie zbiorów węzłów, ich sortowanie według określonych kryteriów, możliwość wywoływania wzorców poprzez ich nazwę i przekazywanie im parametrów to jedne z wielu cech, które w microsoftowej implementacji XSL-a nie mają wsparcia. Również wyspecjalizowane interpretery XSL (np. XT Jamesa Clarka) nie do końca zgadzają się z charakterystyką języka zawartą w specyfikacji W3C.

ZAKOŃCZENIE

Na zakończenie warto zastanowić się, w jakich sytuacjach XML tak naprawdę okaże się przydatny i jakie – mimo wszystkich swoich plusów – ma wady. Przede wszystkim należy zdać sobie sprawę, że głównym celem XML-a jest strukturalizowanie najważniejszych informacji zawartych w dokumentach a następnie obudowywanie tej logicznej struktury innymi informacjami jak np. grafiką. Struktura XML-owa powinna stanowić centrum, które integruje całość. Warto tu zwrócić uwagę, że termin dokument jest tu rozumiany szeroko: może to być zarówno dokument tekstowy jak i arkusz kalkulacyjny czy baza danych. Również rozumienie tekstu w postaci elektronicznego dokumentu XML powinno różnić się od tradycyjnego wyobrażenia. Wciąż jeszcze gdzieniegdzie pokutuje przekonanie o tym, że to, co widać na ekranie monitora to ledwie prototyp „prawdziwego" tekstu, który powstanie dopiero w wyniku wydrukowania. Dokumenty XML istnieją niezależnie od warstwy prezentacyjnej, której jedną z form jest wydruk. Zawarte w nich odpowiednio poukładane dane stanowią podstawę do dalszej manipulacji, a jej efektem może być nie tylko zadrukowana kartka, ale też strona WWW czy inna publikacja elektroniczna.

Po drugie standard ten pełni niezwykle ważną funkcję wymiany danych pomiędzy aplikacjami. Gdyby jedynym przeznaczeniem dokumentów było np. oglądanie ich w postaci stron internetowych dostarczających doraźnych informacji to HTML w zupełności by wystarczył. Jeśli natomiast sieć ma służyć systematycznej wymianie różnorodnych danych, których zarówno nadawcami jak i odbiorcami mogą być programy, to wymusza potrzebę istnienia wspólnej platformy komunikacji. Rozwiązanie to jest szczególnie istotne w działalności gospodarczej „business to business", gdzie istnieje konieczność powszechnej i jak najsprawniejszej (więc automatycznej) wymiany informacji z kontrahentami. Oczywiście informacje te są wstępnie ustrukuralizowane i co więcej, często dają się w łatwy sposób wyrazić w kategoriach języka XML, co w efekcie znacznie upraszcza jego adaptację do środowiska e-commerce.

Oczywiście XML nie jest rozwiązaniem idealnym w każdej sytuacji. Paradoksalnie to, co stanowi jego największą zaletę a więc prostota stanowi jednocześnie największe ograniczenie. Jest to wyłącznie format tekstowy, który nie nadaje się do przesyłania plików binarnych (kody, wywołania

funkcji), wyliczeń, danych zmiennoprzecinkowych i innych bardziej złożonych typów danych. Ponadto XML nie zapewnia pewnych usług systemowych jak weryfikacja tożsamości nadawcy komunikatu. W takich zastosowaniach wciąż lepszymi rozwiązaniami są inne systemy wymiany danych jak np. CORBA (Common Object Request Brokerage Architecture).

Z drugiej strony XML sprawdza się lepiej przy obsłudze dużych zbiorów danych tekstowych, pozwala na zapis w formie pliku całej struktury dokumentu a w przypadku np. problemów przy transmisji prosta jest ingerencja w jego treść w celu wybrania żądanych danych. Te cechy predestynują go do używania w systemach, których aplikacje są zintegrowane na poziomie danych alfanumerycznych.

Jak do tej pory jest to wciąż standard otwarty i nikomu nie udało się stworzyć implementacji, która zawładnęłaby rynkiem narzucając wszystkim lub większości jego uczestników jej stosowanie. Np. Microsoft do wymiany danych w handlu elektronicznym promuje własne rozwiązanie BizTalk a konkurenta ma w projekcie ebXML lansowanym przez OASIS i ONZ. Z kolei Oracle zawarł umowę Fordem na opracowanie systemu wymiany informacji z partnerami handlowymi opartego na XML-u. Wciąż jednak rynek nie zdecydował, który z tych formatów zostanie oficjalnie przyjęty.

Należy wreszcie podkreślić, że celem XML-a nie jest wyparcie z WWW HTML-a. Po prostu gwałtowny rozwój sieci i wzrost wymagań co do standardów strukturalizacji danych, jakie powinny obowiązywać w serwisach informacyjnych w praktyce wymógł rozszerzenie zakresu zastosowania rodzącego się standardu, który w chwili obecnej wykracza nie tylko poza Web, ale znajduje zastosowanie w każdej sferze, gdzie mamy do czynienia z informacją w postaci elektronicznej.

BIBLIOGRAFIA

Bolek Piotr: *SGML, XML*. „Netforum" 2000 nr 2 s. 62-64.

Bosak Jon: *Four myths about XML*.
http://www.oasis-open.org/cover/bosak-4myths.html.

Bosak Jon: *XML, Java, and the future of the Web*.
http://sunsite.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm.

Bosak Jon: *XML – Questions & Answers*.
http://www.isgmlug.org/n3-1/n3-1-18.htm.

Bosak Jon, Bray Tim: *XML and the Second-Generation Web*.
http://www.sciam.com/1999/0599issue/0599bosak.html.

Bosworth Adam: *Microsoft's vision for XML*.
http://www.oasis-open.org/cover/bosworthXML98.html.

Bray Tim: *Beyond HTML: XML and automated Web processing*.
http://developer.netscape.com/viewsource/bray_xml.html.

Bray Tim: *XML namespaces by example*.
http://www.xml.com/xml/pub/1999/01/namespaces.html.

Bryan Martin: *An introduction to the Extensible Markup Language (XML)*.
http://www.personal.u-net.com/~sgml/xmlintro.htm.

Clark James: *XML namespaces*. http://www.jclark.com/xml/xmlns.htm.

Connolly Dan: *The XML revolution*.
http://helix.nature.com/webmatters/xml/xml.html.

Connolly Dan, Khare Rohit, Rifkin Adam: *The evolution of Web documents: the ascent of XML*.
http://www.cs.caltech.edu/~adam/papers/xml/ascent-of-xml.html.

Dmoch Andrzej. *XML i XSL, czyli treść i forma*.
http://webmaster.pckurier.pl/1999/grudzien/dmoch/xmlxsl.html.

DeRose Steven J.: *XML linking: an introduction by Steven J. DeRose.* http://www.stg.brown.edu/~sjd/xlinkintro.html.

Freter Todd: *Beyond text and graphics: XML makes Web pages function like applications.* http://www.sun.com/980414/xml/.

Freter Todd: *Document and information management.* http://www.sun.com/980908/xml/.

Freter Todd: *XML: it's the future of HTML.* http://www.sun.com/980602/xml/.

Freter Todd: *XML: mastering information on the Web.* http://www.sun.com/980310/xml/.

Garshol Lars Marius: *Introduction to XML.* http://www.stud.ifi.uio.no/~lmariusg/download/xml/xml_eng.html.

Gibbs W. Wayt: *Internet uczy się czytać.* http://www.proszynski.com.pl/swiatnauki/1998/sierpien/cyber/cyber.htm.

Gorman Trisha: *20 questions on XML.* http://www.builder.com/Authoring/Xml20/?tag=st.cn.sr1.ssr.bl_xml_questions.

Góralski Andrzej: *XML – i co dalej?* „Netforum" 1999 nr 6 s. 29-31.

Góralski Andrzej: *XML – i co dalej?* „Netforum" 1999 nr 7/8 s. 30-32.

Góralski Andrzej: *XML— i co dalej?* „Netforum" 1999 nr 9 s. 34-37.

Harold Elliotte Rusty: *XML bible.* IDG Books Worldwide 1999.

Jagodziński Marcin: *XML: nieprzewidziane konsekwencje.* http://www.portfolio.art.pl/xml.html.

Khare Rohit, Rifkin Adam: *eXtensible Markup Language: The Least You Need To Know.* http://www.cs.caltech.edu/~adam/papers/xml/tutorial/XML-Basel.html.

Kopacz Tomasz: *BizTalk – rozmówki w XML.* „Computerworld" 1999 nr 40 s. 36.

Księżyk Rafał: *XML dla o(d)pornych - cz.1.: Jak skomputeryzować Internet?* „PC Kurier" 1999 nr 9 s. 63-65.

Księżyk Rafał: *XML dla o(d)pornych – cz.2.: Spójrzmy z góry na HTML!* „PC Kurier" 1999 nr 10 s. 65-70.

Księżyk Rafał: *XML, czyli jak posprzątać Internet.* http://www.fuw.edu.pl/~ksiezyk/a_xml1_pl.html.

Księżyk Rafał: *XML/SGML w Polsce.* http://www.fuw.edu.pl/~ksiezyk/sgml.html.

Księżyk Rafał, Staszelis Jacek: *Na początku było słowo...*
http://www.fuw.edu.pl/~ksiezyk/sgml4gw.html.

Ladd Eric, O'Donnell Jim i inni: *HTML 4, DHTML, VRML, XML*. Warszawa Wydawnictwo Lynx-SFT 1999 str. 371 - 390.

Laurent Simon: *Why XML?*
http://webdevelopersjournal.com/articles/why_xml.html.

Lewandowski Rafał: *Standard Generalized Markup Language – SGML*.
http://rafal.clpz.poznan.pl/Sgml/.

Łakomy Marian: *Wszyscy mówią XML*. „Computerworld" 2000 nr 13 s. 32-33.

Marais Hannes: *An XML tutorial*.
http://www.research.digital.com/SRC/personal/Johannes_Marais/talks/XMLTutorial/.

Mielecki Piotr: *Okienko na świat*. „Chip" 2000 nr 3 s. 168-170.

Muszyński Józef: *Nowe projekty standardów*. „Net World" 1999 nr 6 s. 69-73.

Muszyński Józef: *W3C łączy HTML i XML w XHTML*.
http://www.idg.pl/nws/aktualnosci/zrodlo/networld/55.html.

North Simon: *XML dla każdego*. Gliwice, Helion 2000.

Pawlak Marcin: *XML – czarodziej danych*. „Chip" 1998 nr 11 s. 267 -269.

Rafa Jarosław: *XML: powrót do źródeł*. http://www.wsp.krakow.pl/papers/xml.html.

Rifkin Adam: *A look at XML*.
http://www.webdeveloper.com/xml/xml_a_look_at_xml.html.

Sharpe Bruce: *XML: it's not your father's HTML*.
http://webreference.com/authoring/languages/xml/intro/.

Siegel David: *The Web is ruined and I ruined it.*
http://webreview.com/97/04/11/feature/.

Sall Ken: *XML: structuring data for the Web*.
http://wdvl.com/Authoring/Languages/XML/Intro/.

Sol Selena: *Introduction to XML for Web developers*.
http://wdvl.internet.com/Authoring/Languages/XML/Tutorials/Intro/toc.html.

Stanek William Robert: *Structuring data with XML*.
http://web-e7.zdnet.com/pcmag/pctech/content/17/10/tf1710.001.html.

Szaniawski Jacek: *Słownik angielsko-polski dla informatyków.* Warszawa, ArsKom 1994.

Tidwell Doug: *XML and how it will change the Web.* http://www-4.ibm.com/software/developer/library/xml-web/index.html.

Walsh Norman: *What is XML?* http://www.gca.org/conf/xml/xml_what.htm.

Walter Mark: *Namespaces in XML adopted by W3C.* http://xml.com/xml/pub/1999/01/3namespace.html.

World Wide Web Consortium's XML Special Interest Group: *Frequently asked questions about the Extensible Markup Language.* http://www.ucc.ie/xml/.

Wyke R. Allen: *Is XML changing the future of Web publishing?* http://fbp.icm.edu.pl/sunworldonline/swol-09-1999/swol-09-webmaster.html.

Zelnick Nate: *Why XML?* http://webdeveloper.com/xml/xml_050198.html.

DOKUMENTACJA TECHNICZNA

International Organization for Standardization: *ISO 639:1988. Code for the representation of names of languages.* Geneva, ISO 1988.

International Organization for Standardization: *ISO 8879:1986. Information processing. Text and Office Systems. Standard Generalized Markup Language (SGML).* Geneva, ISO 1986.

International Organization for Standardization: *ISO/IEC 10646-1993. Information technology. Universal Multiple-Octet Coded Character Set (UCS). Part 1: Architecture and Basic Multilingual Plane.* Geneva, ISO 1993.

Unicode Consortium: *The Unicode Standard. Version 2.0.* Reading Addison-Wesley, Developers Press 1996.

World Wide Web Consortium: *Cascading Style Sheets level 2. CSS2 specification. W3C recommendation.* http://www.w3.org/TR/1998/REC-CSS2-19980512.

World Wide Web Consortium: *Document Object Model (DOM) level 1 specification. Version 1.0. W3C recommendation.* http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001.

World Wide Web Consortium: *Extensible Markup Language (XML) version 1.0. W3C recommendation.* http://www.w3.org/TR/1998/REC-xml-19980210.

World Wide Web Consortium: *Extensible Stylesheet Language (XSL). W3C working draft.* http://www.w3.org/TR/2000/WD-xsl-20000301.

World Wide Web Consortium: *HTML 4.01 specification. W3C recommendation*. http://www.w3.org/TR/1999/REC-html401-19991224.

World Wide Web Consortium: *Namespaces in XML. W3C recommendation*. http://www.w3.org/TR/1999/REC-xml-names-19990114.

World Wide Web Consortium: *XHTML™ 1.0: The Extensible HyperText Markup Language – a reformulation of HTML 4 in XML 1.0. W3C recommendation*. http://www.w3.org/TR/2000/REC-xhtml1-20000126.

World Wide Web Consortium: *XML Linking Language (XLink). W3C working draft*. http://www.w3.org/TR/2000/WD-xlink-20000221.

World Wide Web Consortium: *XML Path Language (XPath) version 1.0. W3C Recommendation*. http://www.w3.org/TR/1999/REC-xpath-19991116.

World Wide Web Consortium: *XML Pointer Language (XPointer). W3C working draft*. http://www.w3.org/TR/1999/WD-xptr-19991206.

World Wide Web Consortium: *XSL Transformations (XSLT) version 1.0. W3C recommendation*. http://www.w3.org/TR/1999/REC-xslt-19991116.

Dodatek

Słownik terminów

adiustacja – patrz: **znakowanie**

aplet – mały program osadzony w dokumencie WWW, który po pobraniu tego dokumentu jest wykonywany po stronie klienta przez przegladarkę

ASCII ***(American Standard Code For Information Interchange)*** – standardowy, ośmiobitowy sposób kodowania znaków

back-end – część funkcjonalna programu realizująca jego zadania w sposób niewidoczny dla użytkownika

baza połączeń (*ang.* ***linkbase***) – dokument, którego jedynym zadaniem jest przechowywanie połączeń autonomicznych

białe miejsca (*ang.* ***white spaces)*** – niewidoczne znaki specjalne, do których zaliczamy znaki: spacji, końca akapitu, tabulacji i przesuwu o wiersz

case sensitive – czuły na wielkość znaków; oznacza że zapis np. **„<A>"** nie jest tożsamy z **„<a>"**

CSS ***(Cascading Style Sheets)*** – kaskadowe arkusze stylów; rozszerzenie języka HTML zwiększające możliwości formatowania i precyzję pozycjonowania elementów na stronie WWW

dane - patrz: **dane znakowe**

dane znakowe (*ang.* ***character data***) – czasem nazywane po prostu danymi; wszystko to, co zawiera dokument, a co nie jest elementem znakowania XML.

definicja typu dokumentu (*ang.* ***document type definition***; w skrócie **DTD**) – dokładny opis znakowania (gramatyki i słownictwa) zastosowanego w dokumencie

deklaracja tekstu (*ang.* ***text declaration***) – opcjonalny element zewnętrznej, parsowanej encji definiujący zastosowany w niej sposób kodowania znaków i ewentualnie określający jej zgodność z wersją XML-a.

deklaracja typu dokumentu (*ang.* ***document type declaration***) – część dokumentu XML, gdzie opisane jest znakowanie (tzw. DTD) dla danej klasy dokumentów

dokument dobrze uformowany (*ang.* ***well-formed document***) – dokument nie łamiący żadnych z zasad tworzenia dokumentów XML zawartych w specyfikacji

dokument właściwy (*ang.* ***valid document***) – dokument stworzony zgodnie z zasadami zawartymi w towarzyszącym mu DTD

DTD – patrz: **definicja typu dokumentu**

EBNF (***Extended Bachus-Naur Form***) – notacja do opisu składni języków programowania

EDI (***Electronic Data Interchange***) – standard elektronicznej wymiany danych opracowany w celu ograniczenia roli tradycyjnych dokumentów papierowych w obiegu informacji biurowej

element łączący (*ang.* ***linking element***) – element stanowiący hiperłącze

element podstawowy (ang. ***root element*** lub ***document element***) – element, w którym zawierają się wszystkie inne elementy składające się na dokument

element pusty (ang. ***empty element***) – element o pustej zawartości (tzn. nie zawierający żadnych danych ani potomka)

encja (ang. ***entity***) – obiekt przechowujący pewną zawartość

encja dokumentu (ang. ***document entity***) – obiekt, który zawiera każdy dokument XML; z niego wywodzą się wszystkie inne encje składające się na dany dokument

identyfikator ogólny (ang. **generic identifier**) – unikatowa nazwa określająca typ elementu

instrukcja przetwarzania (ang. ***processing instruction***) – zbiór poleceń przeznaczonych dla konkretnej aplikacji przetwarzającej

Java – zorientowany obiektowo język programowania niezależny od platformy sprzętowej i systemowej. Kod źródłowy programu Javy (plik z rorszerzeniem java) jest kompilowany do formatu pośredniego zwanego kodem bajtowym (rozszerzenie class), który stanowi podstawę wykonania programu przez interpreter Javy (tzw. *Java Virtual Machine*)

JavaBean – moduł programu Javy dostatecznie ogólny i elastyczny, by mógł być łatwo przystosowany na potrzeby innych programów.

język adjustacji – patrz: **język znakowań**

język znakowań (ang. ***markup language***) – zwany czasem językiem adjustacji; język korzystający z mechanizmów znakowania

krok dostępu (ang. ***location step***) – elementarna część ścieżki dostępu wskazująca na pewną część dokumentu XML

link – hiperłącze; element dokumentu elektronicznego prowadzący do innego fragmentu tego samego dokumentu lub do dokumentu znajdującego się w zupełnie innej lokacji

link autonomiczny (ang. ***out-of-line link***) – połączenie, na które składają się wyłącznie zasoby odległe

link prosty (ang. ***simple link***) – link łączący dokładnie dwa zasoby

link rozszerzony (ang. ***extended link***) – link łączący dowolną liczbę zasobów.

link wplatany (ang. ***inline link***) – link, w którego skład wchodzi co najmniej jeden zasób lokalny

middleware – środki programowo-sprzętowe umożliwiające współpracę i wymianę danych między różnymi platformami sprzętowymi i programowymi, które pierwotnie nie były przewidziane do wzajemnego komunikowania się

model zawartości elementu (ang. ***element content model***) – schemat opisujący reguły określania zachowań poszczególnych elementów (np. ilość wystąpień) oraz zachodzących między nimi relacji (rodzic - potomek)

odniesienie do encji (ang. ***entity reference***) – stanowi pewnego rodzaju wywołanie, którego wynikiem jest dołączenie do dokumentu zawartości wywoływanej encji

odniesienie do znaku (ang. ***character reference***) – jeden z typów odniesienia do encji, stosowany w celu zastąpienia pewnych znaków ich szesnastkową (&#x;) bądź dziesiętną (&#...;) reprezentacją

oś (ang. **axis**) – zastrzeżona nazwa XPath określająca sekwencję wyodrębnianych danych

parsowanie – podział wyrażeń języka na poszczególne cząstki będące przedmiotem dokładnej analizy. Najczęściej składa się z analizy leksykalnej skupiającej się na podziale łańcuchów znakowych na poszczególne znaki zgodnie z regułami interpunkcji oraz analizy leksykalnej mającej za zadanie określenie znaczenia znaków.

potomek (ang. ***child***) – element zawierający się w innym elemencie zwanym rodzicem

predykat (ang. ***predicate***) – wyrażenie służące do odfiltrowania określonych zasobów według żądanych kryteriów

przestrzeń nazw (ang. ***namespace***) – abstrakcyjna konstrukcja kojarząca nazwę elementu lub atrybutu z odpowiednim kontekstem

rodzic (ang. ***parent***) – element zawierający w sobie inny element lub elementy zwane potomkami

sekcja warunkowa (ang. ***conditional section***) – konstrukcja pozwalająca dołączać lub wykluczać pewne części z definicji typu dokumentu

ścieżka dostępu (ang. ***location path***) – wyrażenie złożone z co najmniej jednego kroku do-stępu służące do wskazania konkretnej części dokumentu XML

TCP/IP (Transmission Control Protocol / Internet Protocol) – zestaw protokołów definiujących komunikację i wymianę danych w rozproszonych sieciach komputerowych (np. Internet)

typ identyfikacyjny (ang. ***tokenized type***) – jeden z typów atrybutów, służący do określania specjalnych zachowań opisanego nim elementu

typ wyliczeniowy (ang. ***enumerated type***) – jeden z typów atrybutów charakteryzujący się przedstawianiem grupy pewnych danych do wyboru jako wartości danego atrybutu

typ znakowy (ang. ***string type***) – jeden z typów atrybutów definiujący wartość atrybutu jako łańcuch znakowy

Unicode – 16-bitowy system kodowania znaków tekstu umożliwiający przedstawienie 65536 znaków

URL (***Uniform Resource Locator***) – jednolity schemat identyfikujący położenie zasobów w sieci Internet.

UTF 8 (***Universal Transformation Format***) – sposób kodowania znaków i ich konwesji z szesnastobitowego Unicode'a do ośmiobitowego ASCII

walidacja (ang. ***validation***) – procedura weryfikująca poprawność składniową dokumentu oraz jego zgodność z wykładnią zawartą w specyfikacji języka

wartość domyślna (ang. ***default value***) – założona z góry lub zadeklarowana jawnie wartość, jaką przyjmuje atrybut podczas przetwarzania

weryfikacja poprawności – patrz: **walidacja**

węzeł (ang. ***node***) – obiekt stanowiący kompletny zbiór informacji o określonym elemencie składowym dokumentu XML

węzeł podstawowy (ang. ***root node***) – abstrakcyjna konstrukcja obejmująca cały dokument XML, łącznie z deklaracją XML, DTD, komentarzami, instrukcjami przetwarzającymi występującymi przed i wewnątrz elementu podstawowego

wzorzec (ang. ***pattern***) – wyrażenie XSLT określające reguły transformacji konkretnego węzła dokumentu XML

zasób (ang. **resource**) – adresowalna jednostka informacji. Wyróżniamy zasoby lokalne (stanowiące część elementu łączącego) i zdalne (leżące poza elementem łączącym)

zawartość mieszana (ang. ***mixed content***) – odnosi się do zawartości elementów, którą mogą stanowić potomkowie lub parsowane dane znakowe

znakowanie (ang. ***markup***) – ustrukturyzowany sposób kodowania danych za pomocą zdefiniowania pewnych wyrażeń znaczących nazywanych tagami bądź znacznikami

SPIS TREŚCI

Od Autora 5

Wstęp 7

1. FILOGENEZA XML-a 9

1.1. SGML (Standard Generalized Markup Language) 9
1.2. HTML (Hypertext Markup Language) 12
1.3. XML (Extensible Markup Language) a SGML i HTML 16

2. CHARAKTERYSTYKA XML-a 20

2.1. Założenia twórców 20
2.2. Cechy języka 22
2.3. Obszary zastosowań 24
2.4. Konsekwencje dla sieci 29

3. DOKUMENTY XML – ZASADY TWORZENIA 32

3.1. Struktura dokumentu 32
3.2. Dane a znakowanie 33
3.3. Sekcje CDATA 34
3.4. Komentarze 35
3.5. Instrukcje przetwarzania 35
3.6. Deklaracja XML 36
3.7. Elementy (Tagi i atrybuty) 36
3.8. Dokumenty dobrze uformowane a dokumenty właściwe – Definicja Typu Dokumentu 38
3.9. Umiejscowienie DTD 39
3.10. Deklaracje typów elementów 41
3.11. Deklaracje listy atrybutów 43
3.12. Definicje wartości domyślnych 44
3.13. Typy atrybutów 44
3.14. Atrybuty specjalne 46
3.15. Sekcje warunkowe 47
3.16. Encje 47
3.17. Deklaracje notacji 49

4. PRZYKŁADY DOKUMENTÓW 51

4.1. E-mail 51
4.2. Drzewo genealogiczne 53
4.3. Dyskografia 56

4.4. Praca magisterska 59
4.5. Dramaty Szekspira 61

5. STANDARDY STOWARZYSZONE 64

5.1. Przestrzenie nazw XML 64
5.2. XLink 66
5.3. XPointer 72
5.4. XSL 77

Zakończenie 82

Bibliografia 84

Dodatek – Słownik terminów 89

CONTENTS

Acknowledgements 5
Instroduction 7

1. ORIGINS OF XML 9

1.1. SGML (Standard Generalized Markup Language) 9
1.2. HTML (Hypertext Markup Language) 12
1.3. XML (Extensible Markup Language) vs SGML and HTML 16

2. XML CHARACTERISTICS 20

2.1. The Fundations 20
2.2. Language Features 22
2.3. Application Areas 24
2.4. Consequences for Internet 29

3. XML DOCUMENTS – PRINCIPLES OF FORMULATIONS 32

3.1. Document structure 32
3.2. Data vs Markup 33
3.3. CDATA Sections 34
3.4. Comments 35
3.5. Processing Instructions 35
3.6. XML Declaration 36
3.7. Elements (Tags and Attciibutes) 36
3.8. Well-formed Documents vs Valid Dokuments – Document Type Definiction 38
3.9. The Placing of DTD 39
3.10. Element type Declarations 41
3.11. Attribute-list Declarations 43
3.12. Attribute Defaults Definitions 44
3.13. Attribute Types 44
3.14. Special Attributes 46
3.15. Conditional Sections 47
3.16. Entities 47
3.17. Notation Declarations 49

4. EXAMPLE DOCUMENTS 51

4.1. E-mail 51
4.2. Family Tree 53

4.3. Discography 56
4.4. Master`s Thesis 59
4.5. Shakespeare`s Dramas 61

5. ASSOCIATED STANDARDS 64

5.1. XML Namespaces 64
5.2. XLink 66
5.3. XPointer 72
5.4. XSL (Extesible Stylesheet Language) 77

Conclusions 82

Bibliography 84

Appendix – Dictionary 89

DO WSZYSTKICH TYCH, KTÓRZY SIĘ KSZTAŁCĄ LUB DOSKONALĄ

Życzymy Wam sukcesów w nauce

Pamiętajcie o tym, że:

WYDAWNICTWO SBP

jest dla Was. Publikujemy większość literatury, która będzie potrzebna w trakcie studiów. Autorami tych książek są Wasi obecni i przyszli wykładowcy, sprawdzeni dydaktycy i naukowcy. Co roku wydajemy kilka pozycji książkowych z myślą o studentach. Także dla Was wydajemy:

CZASOPISMA

BIBLIOTEKARZ. Indeks 352624. Miesięcznik o charakterze fachowym i naukowym. Ukazuje się od 1929 r. Czasopismo wydawane przez Stowarzyszenie Bibliotekarzy Polskich oraz Bibliotekę Publiczną m.st. Warszawy.

PORADNIK BIBLIOTEKARZA. Indeks 369594. Miesięcznik instrukcyjno-metodyczny. Ukazuje się od 1949 r. Czasopismo wydawane przez Stowarzyszenie Bibliotekarzy Polskich.

ZAGADNIENIA INFORMACJI NAUKOWEJ. Od 1993 r. czasopismo jest wydawane przez Instytut Informacji Naukowej i Studiów Bibliologicznych Uniwersytetu Warszawskiego oraz Stowarzyszenie Bibliotekarzy Polskich, jako półrocznik.

BIULETYN INFORMACYJNY ZG SBP. Ukazuje się od 1993 r. jako „Biuletyn", wcześniej wydawany był „Komunikat ZG SBP". Ukazuje się 2-4 razy w roku. Dokumentuje działalność organizacyjną i merytoryczną Stowarzyszenia.

WYDAWNICTWO SBP

KUPUJĄC U NAS WSPIERASZ POLSKIE BIBLIOTEKARSTWO

WYDAWNICTWO
SBP