helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

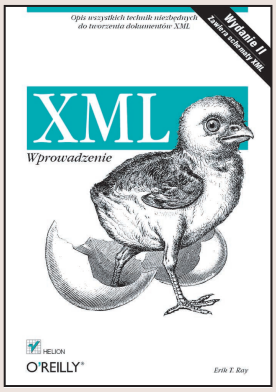
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

XML. Wprowadzenie. Wydanie II



Autor: Erik T. Ray
Tłumaczenie: Bartłomiej Garbacz
ISBN: 83-7361-379-X
Tytuł oryginału: Learning XML, 2nd Edition
Format: B5, stron: 446
Przykłady na ftp: 36 kB

Od czasu swojego pojawienia się pod koniec lat 90. rozszerzalny język znaczników XML (ang. Extensible Markup Language) spowodował powstanie ogromnej liczby akronimów, standardów oraz reguł, które części społeczności internetowej każą się zastanawiać, czy rzeczywiście wszystko to jest potrzebne. Wszakże język HTML istnieje od lat i sprowokował powstanie zupełnie nowej ekonomii i kultury, więc pojawia się pytanie: po co zmieniać to, co dobre? Celem opracowania XML nie jest zastąpienie tego, co już istnieje w sieci WWW, lecz utworzenie solidniejszych i bardziej elastycznych podwalin. Jest to bezprecedensowe przedsięwzięcie konsorcjum organizacji i firm, którego celem jest utworzenie zrębów struktury informacyjnej XXI wieku, którą HTML może jedynie pośrednio wspierać.

Jeżeli Czytelnik jest w jakikolwiek sposób zaangażowany w zarządzanie informacjami lub rozwój sieci WWW, musi poznać XML. Celem niniejszej książki jest przedstawienie Czytelnikowi ogólnego obrazu standardu XML, który obecnie przyjmuje swoją ostateczną formę. Aby w jak największym stopniu skorzystać z tej książki, należy posiadać pewne doświadczenie w zakresie oznaczeń strukturalnych, takich jak HTML lub TEX oraz znać pojęcia dotyczące sieci WWW, takie jak łącza hipertekstowe oraz reprezentacja danych. Jednak aby móc zrozumieć pojęcia dotyczące XML, nie trzeba być programistą.

W niniejszej książce skoncentrujemy się na teorii i praktyce tworzenia dokumentów bez wnikania w zbytnie szczegóły dotyczące pisania aplikacji lub pozyskiwania narzędzi programistycznych. Omówienie zawiłości programowania w XML pozostawiamy innym książkom. Co więcej, gwałtowne zmiany zachodzące na rynku czynią niemal pewnym, że i tak nigdy nie udałoby się zachować zgodności z najnowszym oprogramowaniem XML. Mimo wszystko przedstawione tu informacje będą stanowić dla Czytelnika odpowiedni punkt wyjścia na drodze, którą będzie mógł kroczyć, korzystając z XML.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Przedmowa *7*

Wstęp *9*

Rozdział 1. ***Wprowadzenie*** ***13***

Czym jest XML? 14

Rodowód XML 25

Zastosowania XML 30

Rozpoczynanie pracy z XML 42

Rozdział 2. ***Znaczniki i podstawowe pojęcia*** ***61***

Znaczniki 61

Dokumenty 63

Prolog dokumentu 64

Elementy 69

Encje 79

Inne znaczniki 86

Rozdział 3. ***Modelowanie informacji*** ***91***

Proste metody składowania danych 91

Dokumenty narracyjne 98

Złożone dane 113

Dokumenty opisujące dokumenty 117

Rozdział 4. ***Kontrola jakości za pomocą schematów*** ***123***

Podstawowe pojęcia 123

DTD 129

W3C XML Schema 148

RELAX NG ... 155
Schematron ... 175
Porównanie schematów ... 178

Rozdział 5. ***Prezentacja, część 1: CSS*** ... ***181***
Arkusze stylów ... 182
Podstawy CSS ... 191
Dopasowywanie do reguł ... 195
Właściwości ... 202
Przykłady ... 215

Rozdział 6. ***Języki XPath i XPointer*** ... ***223***
Wierzchołki i drzewa ... 223
Znajdowanie wierzchołków ... 227
Wyrażenia XPath ... 231
XPointer ... 238

Rozdział 7. ***Transformacje przy użyciu języka XSLT*** ... ***247***
Historia ... 248
Pojęcia ... 249
Uruchamianie transformacji ... 253
Element arkusza stylów ... 254
Szablony ... 255
Formatowanie ... 264

Rozdział 8. ***Prezentacja, część 2: XSL-FO*** ... ***285***
Sposób działania ... 287
Prosty przykład ... 292
Model obszarów ... 295
Obiekty formatujące ... 299
Przykład: TEI ... 312
Większy przykład: DocBook ... 318

Rozdział 9. ***Internacjonalizacja*** ... ***341***
Kodowanie znaków ... 341
MIME i typy mediów ... 351
Określanie języków ... 354

Rozdział 10. *Programowanie* ***357***
Ograniczenia 357
Strumienie i zdarzenia 358
Drzewa i obiekty 361
Analiza składniowa przez pobieranie 361
Standardowe interfejsy API 365
Wybór parsera 365
PYX 366
SAX 368
DOM 373
Inne rozwiązania 387

Dodatek A *Zasoby* ***391***
Sieć 391
Książki 393
Organizacje normalizacyjne 395
Narzędzia 396
Różne 397

Dodatek B *Taksonomia standardów* ***399***
Zbiór znaczników i struktura 399
Łączenie 402
Adresowanie i wykonywanie zapytań 404
Style i transformacje 405
Programowanie 407
Przygotowywanie publikacji 409
Hipertekst 410
Zastosowania opisowe i proceduralne 411
Multimedia 412
Nauka 413

Słowniczek ***415***

Skorowidz ***429***

6

Języki XPath i XPointer

Język XML często porównuje się z bazami danych ze względu na sposób, w jaki przechowuje informacje w celu umożliwienia ich łatwego pobierania. Ignorując oczywiste kwestie szybkości i optymalizacji, nie jest to zła analogia. Nazwy elementów i atrybuty nadają danym uchwyty, podobnie jak tabele SQL wykorzystują nazwy tabel i pól. Struktura elementów dostarcza jeszcze więcej informacji w postaci kontekstu (np. element A jest potomkiem elementu B, który występuje po elemencie C itd.). Przy niewielkiej znajomości języka znaczników można zlokalizować i dotrzeć do każdej porcji informacji.

Jest to przydatne z wielu powodów. Po pierwsze, może być konieczne zlokalizowanie określonych danych ze znanej lokalizacji (zwanej *ścieżką*) w danym dokumencie. Posiadając adres URI oraz ścieżkę powinno okazać się możliwe automatyczne pobranie danych. Inna korzyść polega na tym, że można wykorzystać informacje o ścieżce w celu bardzo szczegółowego określenia charakteru przetwarzania całej klasy dokumentów. Zamiast po prostu podawać nazwę elementu lub wartość atrybutu w celu opracowania arkusza styli, na przykład CSS, można użyć wszelkiego rodzaju dodatkowych szczegółów kontekstowych, w tym danych znajdujących się w dowolnym miejscu w dokumencie. Przykładowo, można określić w sekcji metadanych na początku dokumentu, że elementy listy powinny używać określonego symbolu punktora.

W celu wyrażania informacji o ścieżkach w unormowany sposób organizacja W3C zaleca użycie języka XML Path Language (znanego również jako XPath). Pojawił się on wkrótce po opublikowaniu rekomendacji XML i daje wiele nowych możliwości dokumentom i technikom pomocniczym, takim jak XSLT lub DOM. XML Pointer Language (XPointer) stanowi rozszerzenie XPath, pozwalając na lokalizowanie informacji w innych dokumentach.

Wierzchołki i drzewa

W rozdziale 2., kiedy była mowa o drzewach i języku XML, stwierdzono, że każdy dokument XML posiada reprezentację graficzną w postaci struktury drzewiastej. Za chwilę zostanie wyjaśnione, dlaczego jest to tak ważne. Z uwagi na fakt, że istnieje tylko jedna

możliwa konfiguracja drzewa dla dowolnego dokumentu, istnieje unikatowa ścieżka z korzenia (lub dowolnego wierzchołka wewnętrznego) do dowolnego innego punktu. Język XPath opisuje po prostu, w jaki sposób „wspiąć się" po drzewie w serii kolejnych kroków w celu dotarcia do miejsca przeznaczenia.

Przy okazji warto nadmienić, że w niniejszym rozdziale jest często używana terminologia związana z drzewami. Zakłada się, że Czytelnik zapoznał się z krótkim wprowadzeniem do tej problematyki w rozdziale 2.

Typy wierzchołków

Każdy krok na ścieżce jest związany z rozgałęziającym się lub końcowym punktem w drzewie, noszącym nazwę *wierzchołka* (ang. *node*). Zgodnie z terminologią drzewiastą wierzchołek końcowy (taki, który nie posiada potomków) jest czasem nazywany *liściem* (ang. *leaf*). W przypadku języka XPath istnieje siedem różnych rodzajów wierzchołków.

Korzeń
: Korzeń dokumentu jest szczególnym rodzajem wierzchołka. Nie jest to element, jak można by sądzić, ale raczej wierzchołek zawierający element dokumentu. Zawiera również wszelkie komentarze i instrukcje przetwarzania, które otaczają element dokumentu.

Element
: Elementy oraz wierzchołek korzenia cechuje szczególna właściwość — mogą one zawierać inne wierzchołki. Wierzchołek elementu może zawierać inne elementy oraz inne rodzaje wierzchołków oprócz korzenia. W drzewie jest to punkt, w którym spotykają się dwie gałęzie a w przypadku elementu pustego jest to wierzchołek liścia.

Atrybut
: Dla uproszczenia zagadnienia język XPath traktuje atrybuty jako wierzchołki oddzielone od elementów. Pozwala to na wybranie elementu jako całości lub tylko atrybutu z tego elementu przy użyciu tej samej składni ścieżki. Atrybut przypomina element zawierający jedynie tekst.

Tekst
: Obszar nieprzerwanego tekstu jest traktowany jako wierzchołek liścia. Element może jednak posiadać więcej niż jeden wierzchołek tekstowy, jeśli jest rozdzielony elementami lub wierzchołkami innego rodzaju. Należy o tym pamiętać, kiedy przetwarza się tekst w elemencie — może okazać się konieczne sprawdzenie więcej niż tylko jednego wierzchołka.

Komentarz
: Choć z technicznego punktu widzenia komentarze XML nie wnoszą niczego do treści dokumentu i większość procesorów XML po prostu je odrzuca, są one traktowane jako poprawne wierzchołki. Daje to możliwość wyrażenia dokumentu w taki sposób, że będzie go można zrekonstruować z dokładnością do pojedynczego znaku (choć, jak zostanie później wyjaśnione, nie jest do końca możliwe). A poza tym czasem jest wskazane zachowywanie komentarzy.

Instrukcja przetwarzania
: Podobnie jak w przypadku komentarzy, instrukcja przetwarzania może występować w dowolnym miejscu dokumentu pod wierzchołkiem korzenia.

Przestrzeń nazw
: Może wydawać się to dziwne, że deklaracja przestrzeni nazw powinna być traktowana inaczej niż atrybut. Wystarczy jednak przeanalizować następującą uwagę: przestrzeń nazw stanowi w rzeczywistości fragment dokumentu, a nie tylko określenie posiadania jednego elementu. Ma to wpływ na wszystkie potomki takiego elementu. Procesory XML muszą zwracać szczególną uwagę na przestrzenie nazw, tak więc XPath tworzy z nich unikatowy typ wierzchołków.

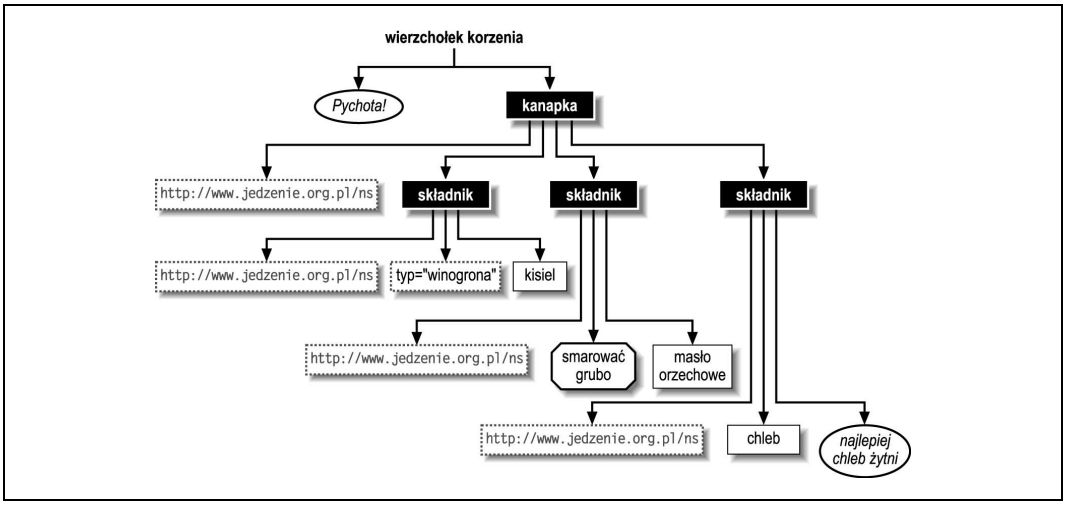
To, czego nie zawarto na powyższej liście to DTD. Nie można używać języka XPath w celu prowadzenia pewnego rodzaju poszukiwań w podzbiorach wewnętrznych lub zewnętrznych. XPath traktuje te informacje po prostu jako ukryte i niewarte bezpośredniego uzyskiwania dostępu. Zakłada również, że wszelkie odwołania do encji są rozstrzygane zanim XPath zacznie przetwarzanie drzewa. Jest to zaletą, ponieważ encje mogą zawierać drzewa elementów, do których zwykle chce się mieć dostęp.

Nie jest do końca prawdą, że XPath zachowuje wszystkie informacje o dokumencie, tak że później można go odtworzyć litera po literze. Zachowywana jest struktura i treść, co zapewnia równoważność *semantyczną*. Oznacza to, że gdyby należało zamienić dokument na program a następnie z powrotem odbudować jego strukturę w pamięci, to prawdopodobnie otrzymany rezultat nie przeszedłby poprawnie testu *diff*[1]. Zmianie ulegają drobne elementy, takie jak kolejność atrybutów (w przypadku języka XML kolejność atrybutów nie ma znaczenia). Znaki białe między elementami mogą zostać usunięte lub zmienione, zaś encje zostają rozwinięte. W celu porównania dwóch równoważnych semantycznie dokumentów trzeba użyć specjalnego narzędzia. Jednym z nich, używanym w środowisku języka Perl, jest moduł XML::SemanticDiff, który stwierdza czy struktura oraz treść są identyczne.

W celu pokazania wierzchołków w ich naturalnym otoczeniu warto spojrzeć na zaprezentowany przykład. Poniższy dokument zawiera wszystkie rodzaje wierzchołków, zaś na rysunku 6.1 przedstawiono go w postaci drzewa.

```
<!-- Pychota! -->
<kanapka xmlns="http://www.jedzenie.org.pl/ns">
  <składnik typ="winogrona">kisiel</składnik>
  <składnik><?nóż smarować grubo?>
    masło orzechowe</składnik>
  <składnik>chleb
    <!-- najlepiej chleb żytni --></składnik>
</kanapka>
```

[1] *diff* to program występujący w systemach uniksowych, który porównuje dwa pliki tekstowe i raportuje o występujących różnicach między odczytanymi wierszami. Nawet jeśli tylko jeden znak znajdzie się nie na swoim miejscu, zostanie to wykryte i udokumentowane.



Rysunek 6.1. Drzewo przedstawiające wszystkie rodzaje wierzchołków

Drzewa i poddrzewa

Jeżeli odetnie się gałąź wierzby i zasadzi ją w ziemi, istnieje duże prawdopodobieństwo, że wyrośnie z niej nowe drzewo. Podobnie jest w przypadku języka XML, każdy wierzchołek w drzewie można postrzegać jako odrębne drzewo. Nie musi posiadać on wierzchołka korzenia, tak więc analogia nie zostaje do końca zachowana ale poza tym drobnym uchybieniem wszystko jest na swoim miejscu: wierzchołek przypomina element dokumentu, posiada potomki i zachowuje strukturę drzewiastą — niejako w sposób fraktalny. Drzewo utworzone z dowolnego wierzchołka nosi nazwę *poddrzewa* (ang. *subtree*).

Rozważmy na przykład poniższy dokument XML.

```
<?xml version="1.0"?>
<podręcznik typ="montaż" id="model-rakiety">
  <lista-części>
    <część etykieta="A" liczba="1">kadłub, lewa połowa</część>
    <część etykieta="B" liczba="1">kadłub, prawa połowa</część>
    <część etykieta="F" liczba="4">brzechwa sterująca</część>
    <część etykieta="N" liczba="3">dysza rakiety</część>
    <część etykieta="C" liczba="1">kapsuła załogowa</część>
  </lista-części>
  <instrukcje>
    <etap>
      Sklej części A i B w celu utworzenia kadłuba.
    </etap>
    <etap>
      Nałóż klej na brzechwy sterujące (część F) i wstaw je do otworów w kadłubie
    </etap>
    <etap>
      Przy pomocy niewielkiej ilości klej przymocuj dysze rakiety (część N)
      w dolnej części kadłuba.
```

```
      </etap>
      <etap>
        Przymocuj kapsułę załogową do górnej części kadłuba. Nie używaj kleju,
        gdyż jest ona zamocowana na sprężynie, umożliwiającej odłączanie się
        od kadłuba.
      </etap>
    </instrukcje>
  </podręcznik>
```

Cały dokument to drzewo, którego elementem korzenia (lub elementem dokumentu) jest element `podręcznik`. Elementy `lista-części` oraz `instrukcje` również mają formę drzew z własnymi korzeniami i gałęziami.

Techniki przetwarzania kodu XML często opierają się na drzewach zagnieżdżonych. Ułatwia to programowanie rekurencyjne, które w tym przypadku jest prostsze i łatwiejsze niż przetwarzanie iteracyjne. Na przykład język XSLT jest elegancki dlatego, że jego reguły traktują każdy element jako drzewo.

Istotną rzeczą jest pamiętanie o tym, że nie można wziąć dowolnego fragmentu dokumentu XML i oczekiwać, że będzie to drzewo wierzchołków. Musi ono być *zrównoważone* (ang. *balanced*). Innymi słowy, każdemu znacznikowi końcowemu musi odpowiadać znacznik początkowy. Z niezrównoważonym fragmentem kodu XML jest bardzo trudno pracować, w szczególności w przypadku języka XPath.

Znajdowanie wierzchołków

Na świecie istnieją osoby, które potrafią nazwać swoich przodków do dziesiątego pokolenia wstecz lub jeszcze dalej. Taka wiedza może pomóc w ustaleniu tożsamości osoby, pokazując że jest ona członkiem takiego lub innego rodu lub jest spokrewniona z kimś poprzez wspólnego prapradziadka.

Język XPath również używa łańcuchów kroków, z tym że są one krokami w drzewie XML, a nie w faktycznym drzewie genealogicznym. Wciąż mają zastosowanie pojęcia „dziecko" i „rodzic". *Ścieżka lokalizacji* (ang. *location path*) to łańcuch *kroków lokalizacji* (ang. *location steps*), które pozwalają dotrzeć z jednego punktu dokumentu do drugiego. Jeżeli ścieżka rozpoczyna się od pozycji bezwzględnej (na przykład od wierzchołka korzenia), to nosi nazwę *ścieżki bezwzględnej* (ang. *absolute path*). W przeciwnym wypadku nosi ona nazwę *ścieżki względnej* (ang. *relative path*), ponieważ rozpoczyna się z nieokreślonego z góry miejsca.

Krok lokalizacji posiada trzy części: *oś* (ang. *axis*), która opisuje kierunek przemieszczania się, *test wierzchołka* (ang. *node test*), który określa, jakiego rodzaju wierzchołki są istotne oraz zestaw opcjonalnych *predykatów* (ang. *predicates*), które wykorzystują testy logiczne (zwracające wartość prawda-fałsz) w celu jeszcze dokładniejszego zbadania wierzchołków kandydujących.

Osią jest słowo kluczowe, które określa kierunek przemieszczania się z dowolnego wierzchołka. Można przechodzić przez przodki, przez potomki lub liniowo przez elementy siostrzane. W tabeli 6.1 zawarto listę wszystkich typów osi wierzchołków.

Tabela 6.1. Osie wierzchołków

Typ osi	Dopasowanie
Ancestor (przodek)	Wszystkie wierzchołki znajdujące się nad wierzchołkiem kontekstowym, w tym rodzic, rodzic rodzica i tak dalej aż do wierzchołka korzenia.
Ancestor-or-self (przodek lub bieżący)	Wierzchołek przodka oraz wierzchołek kontekstowy.
Attribute (atrybut)	Atrybuty wierzchołka kontekstowego.
Child (dziecko)	Dzieci wierzchołka kontekstowego.
Descendant (potomek)	Dzieci wierzchołka kontekstowego oraz ich dzieci i tak dalej aż do poziomu liści poddrzewa.
Descendant-or-self (potomek lub bieżący)	Wierzchołek potomka oraz wierzchołek kontekstowy.
Following (następujący)	Wierzchołki, które następują po wierzchołku kontekstowym na dowolnym poziomie w dokumencie. Nie zalicza się do nich potomków wierzchołka kontekstowego, ale zalicza jego wierzchołki siostrzane wraz z ich potomkami.
Following-sibling (następujący siostrzany)	Wierzchołki, które następują po wierzchołku kontekstowym na tym samym poziomie (tzn. te, które mają tego samego rodzica co wierzchołek kontekstowy).
Namespace (przestrzeń nazw)	Wszystkie wierzchołki przestrzeni nazw elementu.
Parent (rodzic)	Rodzic wierzchołka kontekstowego.
Preceding (poprzedzający)	Wierzchołki, które występują przed wierzchołkiem kontekstowym na dowolnym poziomie w dokumencie. Nie zalicza się do nich żadnych potomków wierzchołka kontekstowego, ale zalicza jego poprzedzające elementy siostrzane wraz z ich potomkami.
Preceding-sibling (poprzedzający siostrzany)	Wierzchołki, które występują przed wierzchołkiem kontekstowym na tym samym poziomie (tzn. te, które mają tego samego rodzica co wierzchołek kontekstowy).
Self (bieżący)	Wierzchołek kontekstowy.

Po osi występuje parametr testu wierzchołka, połączony z osią dwoma znakami dwukropka (`::`). Zamiast typu wierzchołka można użyć nazwy i w takim przypadku typ wierzchołka jest wnioskowany na podstawie osi. W przypadku osi atrybutu przyjmowanym wierzchołkiem jest atrybut, zaś w przypadku osi przestrzeni nazw przyjmowanym wierzchołkiem jest przestrzeń nazw. W przypadku wszystkich innych osi przyjmowanym wierzchołkiem jest element. W razie braku specyfikatora osi wierzchołka przyjmowaną osią jest `child`, zaś typem wierzchołka — element. W tabeli 6.2 wymieniono testy wierzchołków.

Tabela 6.2. Testy wierzchołków

Warunek	Dopasowanie
`/`	Wierzchołek korzenia: nie element korzenia, lecz jego wierzchołek, zawierający wszelkie komentarze lub instrukcje przetwarzania poprzedzające go.
`node()`	Każdy wierzchołek. Przykładowo, krok `attribute::node()` wybrałby wszystkie atrybuty wierzchołka kontekstowego.
`*`	W przypadku osi atrybutów — każdy atrybut. W przypadku osi przestrzeni nazw — każda przestrzeń nazw. W przypadku wszystkich innych osi — dowolny element.
`ciastko`	W przypadku osi atrybutów — atrybut o nazwie `ciastko` wierzchołka kontekstowego. W przypadku osi przestrzeni nazw — przestrzeń nazw o nazwie `ciastko`. We wszystkich innych przypadkach — każdy element o nazwie `ciastko`.
`text()`	Dowolny wierzchołek tekstowy.
`processing-instruction()`	Dowolna instrukcja przetwarzania.
`processing-instruction ('do-sieci')`	Dowolna instrukcja przetwarzania, której celem jest `do-sieci`.
`comment()`	Dowolny komentarz.

Kroki lokalizacji ścieżki są łączone ze sobą za pomocą znaku ukośnika (`/`). Każdy krok przybliża do wierzchołka, który chce się zlokalizować. Przypomina to tłumaczenie komuś drogi do restauracji (*Dojdź do Placu Wolności, skręć w Aleję Niepodległości; na rondzie ONZ skręć w lewo i zobaczysz świetną wietnamską restaurację*). Przykładowo, w celu dostania się z wierzchołka korzenia do elementu `para`, znajdującego się wewnątrz elementu `section` wewnątrz elementu `chapter` wewnątrz elementu `book` można by zapisać następującą ścieżkę:

```
book/chapter/section/para
```

Taka składnia może okazać się bardzo rozwlekła, jednak XPath oferuje pewne przydatne skróty, które wymieniono w tabeli 6.3.

Aby zobaczyć, w jaki sposób można używać osi oraz testów wierzchołków w celu docierania do wierzchołków, warto przeanalizować pewien przykład. Weźmy pod uwagę kod przedstawiony na listingu 6.1.

Listing 6.1. Przykładowy dokument XML

```
<lista-cytatów>
  <cytat styl="mądrość" id="c1">
    <tekst>Nie oczekuj niczego, bądź gotów na wszystko.</tekst>
    <źródło>Powiedzenie samurajskie</źródło>
  </cytat>
  <cytat styl="polityczne" id="c2">
```

Tabela 6.3. Skróty lokalizacji ścieżek

Wzorzec	Dopasowanie
`@rola`	Odpowiada atrybutowi o nazwie `rola`. Jest to równoważne zapisowi `attribute::rola`.
`.`	Wierzchołek kontekstowy. Jest to równoważne zapisowi `self::node()`.
`/*`	Odpowiada elementowi dokumentu. Każda ścieżka lokalizacji rozpoczynająca się od znaku ukośnika (`/`) jest ścieżką bezwzględną, w której pierwszy krok reprezentuje wierzchołek korzenia. Kolejnym krokiem jest `*`, co dopasowuje dowolny element.
`parent::/*/ following-sibling::para`	Odpowiada wszystkim elementom `para`, które występują po rodzicu wierzchołka kontekstowego.
`..`	Odpowiada wierzchołkowi rodzica. Dwie kropki (`..`) to skrócona forma zapisu `parent::node()`.
`.//para`	Odpowiada dowolnemu elementowi typu `para`, który jest potomkiem wierzchołka bieżącego. Dwa ukośniki (`//`) to skrócona forma zapisu `/descendant-or-self::node()//`.
`//para`	Odpowiada każdemu elementowi `<para>`, który jest potomkiem wierzchołka korzenia. Innymi słowy, odpowiada wszystkim elementom `para` znajdującym się w dowolnym miejscu w dokumencie. Co do ścieżki lokalizacji rozpoczynającej się od podwójnego ukośnika (`//`) przyjmuje się, że zaczyna się ona od wierzchołka korzenia.
`../*`	Odpowiada wszystkim wierzchołkom siostrzanym (oraz wierzchołkowi kontekstowemu, o ile jest elementem).

```
      <tekst>Gdybym któregoś ranka przeszedł o suchej stopie z jednego brzegu
      Potomac na drugi, wieczorem nagłówki gazet głosiłyby "Prezydent nie umie
      pływać".</tekst>
      <źródło>Lyndon B. Johnson</źródło>
    </cytat>
    <cytat styl="głupawe" id="c3">
      <?śmiech?>
      <tekst>A co jeśli w tym wszystkim chodzi o czary mary?</tekst>
    </cytat>
    <cytat styl="mądrość" id="c4">
      <tekst>Jeśli dadzą ci papier w linie, pisz w poprzek.</tekst>
      <źródło>Juan Ramon Jiminez</źródło>
    </cytat>
    <!-- książeczka czekowa jest silniejsza od miecza? -->
    <cytat styl="polityczne" id="c5">
      <tekst>Banki są groźniejsze od stałych armii.</tekst>
      <źródło>Thomas Jefferson</źródło>
    </cytat>
  </lista-cytatów>
```

W tabeli 6.4 zawarto pewne ścieżki lokalizacji oraz zwracane przez nie wartości.

Należy zauważyć, że krok `id()` działa jedynie w przypadku atrybutów, które zadeklarowano jako typu `ID` w DTD. To właśnie ta deklaracja mówi parserowi sprawdzającemu, że należy wymagać, aby atrybut posiadał unikatową wartość.

Tabela 6.4. Przykłady ścieżek lokalizacji

Ścieżka	Dopasowania
`/lista-cytatów/child::node()`	Wszystkie elementy `cytat` oraz komentarz XML.
`/lista-cytatów/cytat`	Wszystkie elementy `cytat`.
`/*/*`	Wszystkie elementy `cytat`.
`//comment()/following-sibling::*/@styl`	Atrybut `styl` ostatniego elementu `cytat`.
`id('c1')/parent::`	Pierwszy element `cytat`.
`id('c2')/..`	Element dokumentu.
`id('c1')/ancestor-or-self::*`	Element dokumentu oraz pierwszy element `cytat`.
`id('c3')self::aforyzm`	Nic. Pierwszy krok powoduje dopasowanie trzeciego elementu `cytat` ale kolejny krok zmienia to, gdyż wyszukuje elementy typu `aforyzm`. W kontekście, w którym nie wiadomo, jakiego typu jest element, jest to dobry sposób sprawdzenia tego.
`//processing-instruction()/../following::źródło`	Elementy `źródło` ostatnich dwóch elementów `cytat`.

Jeżeli oś i typ wierzchołka nie wystarczą do ograniczenia wyboru, można użyć jednego lub większej liczby predykatów. Predykat to wyrażenie logiczne ujęte w nawiasy kwadratowe (`[]`). Każdy wierzchołek, który poprawnie przechodzi taki test (oprócz testu wierzchołka i specyfikatora osi) zostają uwzględniony w wynikowym zbiorze wierzchołków. Wierzchołki, które nie przechodzą testu (wyliczoną wartością predykatu jest fałsz), są pomijane. W tabeli 6.5 zawarto pewne przykłady.

Wyrażenia XPath

Ścieżki lokalizacji stanowią podzbiór bardziej ogólnej koncepcji *wyrażeń XPath* (ang. *XPath expressions*). Są to instrukcje, które umożliwiają wydobywanie przydatnych informacji z drzewa. Zamiast po prostu znajdować wierzchołki można je zliczać, dodawać wartości numeryczne, porównywać ciągi znaków i wykonywać inne działania. W dużym stopniu przypominają one instrukcje funkcjonalnego języka programowania. Istnieje pięć typów wyrażeń, które wymieniono poniżej.

Logiczne
: Typ wyrażenia o dwóch możliwych wartościach: `true` (*prawda*) i `false` (*fałsz*).

Zbiór wierzchołków
: Kolekcja wierzchołków, które odpowiadają kryteriom wyrażenia, zwykle otrzymana za pomocą ścieżki lokalizacji.

Liczbowe
: Wartość numeryczna, przydatna do zliczania wierzchołków i wykonywania prostych operacji arytmetycznych.

Tabela 6.5. Predykaty języka XPath

Ścieżka	Dopasowanie
`//cytat[@id="c3"]/tekst`	Element tekstowy w trzecim elemencie `cytat`. Jest to przykład testu równości, gdzie wartość znakowa atrybutu jest porównywana z innym ciągiem znakowym Można również przeprowadzać testy numeryczne i logiczne.
`//cytat[źródło]`	Wszystkie elementy `cytat` oprócz trzeciego, który nie posiada elementu `źródło`. Tutaj jest sprawdzana obecność elementu potomnego `źródło`. Jeżeli zostanie znaleziony przynajmniej jeden pasujący wierzchołek, wartością testu jest prawda, a w przeciwnym wypadku — fałsz.
`//cytat[not(źródło)]`	Trzeci element `cytat`. Funkcja `not()` zwróci wartość prawdy, kiedy nie występują elementy `źródło`.
`/*[@id="c2"]/preceding-sibling::*/źródło`	Element `źródło` z treścią „Powiedzenie samurajskie".
`//*[źródło='Thomas Jefferson']/tekst`	Element `tekst` w ostatnim elemencie `cytat`.
`//*[źródło='Thomas Jefferson'][@id='c7']`	Nic. Wyliczana wartość to iloczyn logiczny `and` obu predykatów. Oba z nich muszą zwrócić wartość prawdy, aby ścieżka istniała. Ze względu na fakt, że nie istnieje element spełniający oba te testy, w wyniku nie pozostaje nic.
`/*/*[position()=last()]`	Ostatni element `cytat`. Funkcja `position()` zwraca pozycję ostatniego kroku wśród adekwatnych kandydatów. Funkcja `last()` zwraca całkowitą liczbę kandydatów (w tym przypadku 5).
`//cytat[position()!=2]`	Wszystkie elementy `cytat` oprócz drugiego.
`//cytat[4]`	Czwarty element `cytat`. Sam numer występujący w predykacie stanowi skrócony zapis dla `position()=...`
`//cytat[@typ='głupawe' or @typ='mądrości']`	Pierwszy, trzeci i czwarty element `cytat`. Słowo kluczowe `or` funkcjonuje jako operator sumy logicznej.

Ciąg znaków
: Fragment tekstu, który może pochodzić z drzewa wejściowego, przetworzonego lub uzupełnionego o tekst generowany.

Wynikowe drzewo częściowe
: Tymczasowe drzewo wierzchołków, które posiada własny wierzchołek korzenia ale nie może być indeksowane do wykorzystania przez ścieżki lokalizacji.

W języku XPath typy są determinowane przez kontekst. Operator lub funkcja może przekształcać jeden typ wyrażenia do drugiego w miarę potrzeby. Z tego powodu istnieją dobrze zdefiniowane reguły określania, na jakie wartości należy dokonać odwzorowania w czasie transformacji do innego typu.

XPath oferuje bogaty zbiór operatorów i funkcji służących do pracy z każdym typem wyrażenia. Zarówno one, jak również reguły konwersji typów zostaną opisane w kolejnych podrozdziałach.

Wyrażenia logiczne

Wyrażenia logiczne posiadają dwie wartości: prawda oraz fałsz. Jak pokazano w przykładach predykatów kroku lokalizacji, wszystko co znajduje się między nawiasami kwadratowymi i nie zwraca wartości numerycznej jest traktowane w kontekście logicznym. Istnieją inne sposoby wymuszenia na wyrażeniu zachowania zgodnego z kontekstem logicznym. Funkcja `boolean()` zwraca na podstawie swoich argumentów wartość *prawda* lub *fałsz*. Istnieją również różne operatory, które łączą i porównują wyrażenia, dając wynik logiczny.

Wartość otrzymywana na podstawie wyrażenia zależy od pewnych reguł, które wymieniono w tabeli 6.6.

Tabela 6.6. Reguły konwersji do postaci logicznej

Typ wyrażenia	**Reguła**
Zbiór wierzchołków	Prawda, jeżeli zbiór zawiera co najmniej jeden wierzchołek, fałsz, jeśli jest pusty.
Ciąg znaków	Prawda, chyba że ciąg ma długość zerową.
Liczba	Prawda, chyba że wartością jest zero lub NaN (*not a number*, nie liczba)
Wynikowe drzewo częściowe	Zawsze prawda, ponieważ każdy fragment zawiera przynajmniej jeden wierzchołek — korzeń.

Pewne operacje (wymienione w tabeli 6.7) służą do porównywania wartości numerycznych, czego wynikiem są wyrażenia logiczne. Są to *porównania szczegółowe* (ang. *existential comparisons*), co oznacza, że testują one wszystkie wierzchołki danego zbioru wierzchołków w celu określenia, czy któryś z nich spełnia warunek porównania.

W tabeli 6.8 zawarto funkcje, które zwracają wartości logiczne.

Wyrażenia zbioru wierzchołków

Wyrażenie zbioru wierzchołków stanowi w rzeczywistości to samo, co ścieżka lokalizacji. Wyrażenie jest wyliczane jako zbiór wierzchołków. Jest to zbiór w ściśle matematycznym sensie, co oznacza, że nie zawiera on duplikatów. Ten sam wierzchołek może być dodawany wielokrotnie, jednak zbiór zawsze będzie zawierał tylko jedną jego kopię.

Język XPath definiuje wiele funkcji, które operują na zbiorach wierzchołków. Wymieniono je w tabeli 6.9.

Tabela 6.7. Operatory porównania

Operator	Zwracana wartość
wyr = *wyr*	Prawda, jeżeli oba wyrażenia (ciągi znaków lub liczby) mają tę samą wartość. W przeciwnym wypadku fałsz.
wyr != *wyr*	Prawda, jeżeli wyrażenia (ciągi znaków lub liczby) nie mają tej samej wartości. W przeciwnym wypadku fałsz.
wyr < *wyr*[a]	Prawda, jeżeli wartość pierwszego wyrażenia numerycznego jest mniejsza niż wartość drugiego. W przeciwnym wypadku fałsz.
wyr > *wyr*[a]	Prawda, jeżeli wartość pierwszego wyrażenia numerycznego jest większa niż wartość drugiego. W przeciwnym wypadku fałsz.
wyr <= *wyr*[a]	Prawda, jeżeli wartość pierwszego wyrażenia numerycznego jest mniejsza lub równa wartości drugiego. W przeciwnym wypadku fałsz.
wyr >= *wyr*[a]	Prawda, jeżeli wartość pierwszego wyrażenia numerycznego jest większa lub równa wartości drugiego. W przeciwnym wypadku fałsz.

[a] Jeżeli użyje się tych operatorów w dokumencie XML, na przykład arkuszu styli XSLT lub w schemacie Schematron, należy skorzystać z odwołań do znaków `<` oraz `>` zamiast symboli < i >.

Tabela 6.8. Funkcje logiczne

Funkcja	Zwracana wartość
wyr `and` *wyr*	Prawda, jeżeli oba wyrażenia logiczne mają wartość *prawda*. W przeciwnym wypadku *fałsz*.
wyr `or` *wyr*	Prawda, jeżeli przynajmniej jedno z wyrażeń logicznych ma wartość *prawda*. W przeciwnym wypadku *fałsz*.
`true()`	Prawda.
`false()`	Fałsz.
`not(`*wyr*`)`	Negacja wartości wyrażenia logicznego: *prawda*, jeżeli wyrażenie jest fałszywe, *fałsz*, jeśli wyrażenie jest prawdziwe.

Istnieją również funkcje, które tworzą zbiory wierzchołków, zbierając razem wierzchołki z całego dokumentu. Przykładowo, funkcja `id(`*ciąg_zn*`)` zwraca zbiór elementów, które posiadają atrybut `ID` równy wartości *ciąg_zn* lub zbiór pusty, jeśli nie zostanie dopasowany żaden wierzchołek. W poprawnym dokumencie powinien zostać zwrócony tylko jeden wierzchołek, ponieważ atrybut typu `ID` musi posiadać unikatową wartość. Język XPath nie wymaga jednak, aby dokument był poprawny, tak więc istnieje możliwość, że zostanie zwrócony więcej niż jeden wierzchołek.

Wyrażenia liczbowe

XPath dopuszcza numeryczne wyliczanie wartości wyrażenia, co jest przydatne w przypadku porównywania pozycji w zbiorze, dodawania wartości elementów numerycznych, zwiększania liczników i tak dalej. Liczba w języku XPath to 64-bitowa wartość

Tabela 6.9. Funkcje zbioru wierzchołków

Funkcja	Zwracana wartość
`count(zbior_wierzch)`	Liczba elementów w zbiorze *zbior_wierzch*. Przykładowo, `count(parent::*)` zwróci wartość 0, jeżeli wierzchołek kontekstowy jest elementem dokumentu. W przeciwnym wypadku zostanie zwrócone 1, gdyż wierzchołek może mieć tylko jednego rodzica.
`generate-id (zbior_wierzch)`	Ciąg znaków zawierający unikatowy identyfikator dla pierwszego wierzchołka w zbiorze *zbior_wierzch* lub dla wierzchołka kontekstowego, jeżeli pominie się argument wywołania. Ciąg ten jest generowany przez procesor i jest gwarantowana jego unikatowość dla każdego wierzchołka.
`last()`	Numer ostatniego wierzchołka w kontekstowym zbiorze wierzchołków. Funkcja `last()` jest podobna do funkcji `count()` z wyjątkiem tego, że operuje tylko na kontekstowym zbiorze wierzchołków, a nie na dowolnym zbiorze.
`local-name (zbior_wierzch)`	Nazwa pierwszego wierzchołka w zbiorze *zbior_wierzch* bez prefiksu przestrzeni nazw. W przypadku pominięcia argumentu wywołania zwracana jest lokalna nazwa wierzchołka kontekstowego.
`name(zbior_wierzch)`	Nazwa pierwszego wierzchołka w zbiorze *zbior_wierzch* wraz z prefiksem przestrzeni nazw.
`namespace-uri (zbior_wierzch)`	Adres URI przestrzeni nazw dla pierwszego wierzchołka w zbiorze *zbior_wierzch*. W przypadku pominięcia argumentu wywołania zwracany jest adres URI przestrzeni nazw wierzchołka kontekstowego.
`position()`	Numer wierzchołka kontekstowego w kontekstowym zbiorze wierzchołków.

zmiennopozycyjna (bez względu na to czy posiada część ułamkową, czy nie). Wartością może również być NaN (*not a number*, nie liczba) w przypadku, gdy konwersja zakończy się niepowodzeniem.

W tabeli 6.10 zawarto reguły konwertowania wyrażeń do postaci liczbowej.

Tabela 6.10. Reguły konwersji do postaci liczbowej

Typ wyrażenia	Reguła
Zbiór wierzchołków	Pierwszy wierzchołek jest konwertowany do ciągu znaków a następnie jest używana konwersja z postaci znakowej.
Logiczne	Wartość `true` jest konwertowana na liczbę 1, zaś wartość `false` na 0.
Ciąg znaków	Jeżeli ciąg znaków jest literalną serializacją liczby (np. `-123.5`), jest on konwertowany do tej liczby. W przeciwnym wypadku zwracana jest wartość NaN.
Wynikowe drzewo częściowe	Podobnie jak w przypadku zbiorów wierzchołków, wynikowe drzewo częściowe jest konwertowane do postaci ciągu znaków, który jest następnie konwertowany zgodnie z regułą dotyczącą ciągów znaków.

W celu manipulowania wartościami liczbowymi można korzystać z wielu operatorów i funkcji. Wymieniono je w tabeli 6.11.

Tabela 6.11. Operatory i funkcje liczbowe

Funkcja	Zwracana wartość
`wyr + wyr`	Suma dwóch wyrażeń liczbowych.
`wyr - wyr`	Różnica dwóch wyrażeń liczbowych.
`wyr * wyr`	Iloczyn dwóch wyrażeń liczbowych.
`wyr div wyr`	Iloraz dwóch wyrażeń liczbowych.
`wyr mod wyr`	Reszta z dzielenia pierwszego wyrażenia liczbowego przez drugie.
`round(wyr)`	Wartość wyrażenia zaokrąglona do najbliższej liczby całkowitej.
`floor(wyr)`	Wartość wyrażenia zaokrąglona w dół do najbliższej liczby całkowitej.
`ceiling(wyr)`	Wartość wyrażenia zaokrąglona w górę do najbliższej liczby całkowitej.
`sum(zbior_wierzch)`	Suma wartości wierzchołków ze zbioru `zbior_wierzch`. W przeciwieństwie do innych funkcji z niniejszej tabeli funkcja ta operuje na zbiorze wierzchołków, a nie na wyrażeniach.

Wyrażenia tekstowe

Ciąg znaków to segment danych znakowych, na przykład „Jak się masz?", „990" lub „z". Dowolne wyrażenie można przekonwertować do postaci ciągu znaków za pomocą funkcji `string()`, zgodnie z regułami podanymi w tabeli 6.12.

Tabela 6.12. Reguły konwersji wyrażeń do postaci tekstowej

Typ wyrażenia	Reguła
Zbiór wierzchołków	Jako ciąg znaków jest używana wartość tekstowa pierwszego wierzchołka.
Logiczne	Ciąg ma postać `true`, jeśli wyrażenie jest prawdziwe, a w przeciwnym wypadku ma wartość `false`.
Liczbowe	Wartość ciągu znaków jest liczbą, która zostałaby wydrukowana. Przykładowo, wartością wywołania funkcji `string(1 + 5 - 9)` jest ciąg znaków `-3`.
Wynikowe drzewo częściowe	Wartością ciągu znaków jest złączenie wartości tekstowych wszystkich wierzchołków należących do drzewa częściowego.

W tabeli 6.13 zawarto funkcje, które zwracają wartość ciągu znaków.

Pewne funkcje operują na ciągach znaków i zwracają wartość liczbową lub logiczną. Wymieniono je w tabeli 6.14.

Tabela 6.13. Funkcje, które tworzą ciągi znaków

Funkcja	Zwracana wartość
`concat(`*`ciąg`*`,`*`ciąg`*`,...)`	Ciąg znaków będący złożeniem argumentów wywołania funkcji.
`format-number (`*`liczba`*`, `*`wzorzec`*`, `*`format_dziesiętny`*`)`	Ciąg znaków zawierający liczbę *liczba* sformatowaną zgodnie ze wzorcem *wzorzec*. Opcjonalny argument *format-dziesiętny* wskazuje na deklarację formatu, która przypisuje znaki specjalne, takie jak znak grupowania, który rozdziela grupy cyfr w dużych liczbach w celu zwiększenia czytelności zapisu. W przypadku języka XSLT taka deklaracja formatu byłaby wartością atrybutu `name` w elemencie `decimal-format`.
`normalize-space(`*`ciąg`*`)`	Ciąg znaków *ciąg* z usuniętymi wiodącymi i końcowymi znakami białymi oraz wszystkimi innymi ciągami znaków białych zastąpionymi pojedynczym znakiem spacji. W przypadku, gdy argument wywołania zostanie pominięty, używana jest wartość wierzchołka kontekstowego.
`substring(`*`ciąg`*`, `*`przesunięcie`*`, `*`zakres`*`)`	Podciąg argumentu *ciąg*, rozpoczynający się *przesunięciem* znaków od jego początku i kończący *zakres* znaków od *przesunięcia*.
`substring-after(`*`ciąg`*`, `*`dopasowanie`*`)`	Podciąg argumentu *ciąg*, rozpoczynający się na końcu pierwszego wystąpienia ciągu *dopasowanie* i kończący na końcu argumentu *ciąg*.
`substring-before (`*`ciąg`*`, `*`dopasowanie`*`)`	Podciąg argumentu *ciąg*, rozpoczynający się na początku argumentu *ciąg* i kończący na początku pierwszego wystąpienia ciągu *dopasowanie*.
`translate(`*`ciąg`*`, `*`znaki-dopasowania`*`, `*`znaki-zastępowania`*`)`	Ciąg znaków *ciąg* ze wszystkimi znakami występującymi w ciągu *znaki-dopasowania* zastąpionymi ich odpowiednikami w ciągu *znaki-zastępowania*. Załóżmy, że pierwszym argumentem jest ciąg „wiele hałasu o nic.", drugi argument to „ai.", zaś trzeci to „eO!". Zwrócony ciąg będzie miał postać „wOele hełesu o nOc!". Funkcja `translate()` działa jedynie według schematu „znak za znak", tak więc nie można za jej pomocą zastępować dowolnych ciągów.

Tabela 6.14. Funkcje operujące na ciągach znaków

Funkcja	Zwracana wartość
`contains (`*`ciąg`*`, `*`podciąg`*`)`	Wartość `true`, jeżeli *podciąg* występuje w ciągu znaków *ciąg*. W przeciwnym wypadku wartość `false`.
`starts-with (`*`ciąg`*`, `*`podciąg`*`)`	Wartość `true`, jeżeli ciąg *ciąg* rozpoczyna się od podciągu *podciąg*. W przeciwnym wypadku wartość `false`.
`string-length(`*`ciąg`*`)`	Liczba znaków występujących w ciągu znaków *ciąg*.

XPointer

Blisko związany z językiem XPath jest język XML Pointer Language (XPointer). Wykorzystuje on wyrażenia XPath w celu znajdowania punktów wewnątrz encji zewnętrznych podlegających analizie składniowej jako rozszerzenie jednorodnych identyfikatorów zasobów (URI). Może on być na przykład wykorzystywany do tworzenia łączy z jednego dokumentu do elementu znajdującego się w innym dokumencie.

Początkowo zaprojektowany jako komponent języka XML Linking Language (XLink) XPointer stał się istotnym samodzielnym źródłem standardów stosowania składni *identyfikatora fragmentów* (ang. *fragment identifier*). W 2003 roku XPointer Framework stał się rekomendacją wraz ze schematem XPointer element() Scheme (pozwalającym na podstawowe adresowanie elementów) oraz XPointer xmlns() Scheme (wprowadzającym obsługę przestrzeni nazw). Schemat xpointer() zatrzymał się na etapie propozycji wstępnej i nie jest dalej rozwijany.

Realizacja standardu XPointer, którą będziemy określać jako *xpointer*, działa podobnie do identyfikatora fragmentów w języku HTML (jest to część adresu URL, którą czasem widać po prawej stronie symbolu krzyżyka). Jest jednak znacznie bardziej wszechstronna niż mechanizm używany w przypadku HTML, gdyż umożliwia odwoływanie się do dowolnego elementu lub punktu w ramach tekstu, a nie tylko elementu zakotwiczonego (`<a name="..."/>`). Wykorzystanie XPath oferuje kilka możliwości przewyższających identyfikatory fragmentów stosowane w HTML:

- można utworzyć łącze do samego elementu docelowego, a nie pewnego elementu pośredniego (np. `<a name="foo"/>`);
- nie trzeba posiadać zdefiniowanych zakotwiczeń w dokumencie docelowym. Można swobodnie tworzyć łącza do dowolnego obszaru w dowolnym dokumencie, bez względu na to czy jego autor o tym wie, czy nie;
- język XPath jest wystarczająco elastyczny, aby dotrzeć do dowolnego elementu w dokumencie docelowym.

XPointer w rzeczywistości wykracza poza możliwości języka XPath. Oprócz wierzchołków obsługuje on dwa dodatkowe rodzaje lokalizacji. *Punkt* (ang. *point*) to dowolne miejsce w dokumencie między dwoma sąsiadującymi znakami. XPath umożliwia lokalizowanie jedynie całych wierzchołków tekstowych a XPointer umożliwia większą szczegółowość i lokalizowanie miejsca w środku dowolnego zdania. Kolejnym rodzajem wprowadzonym przez XPointer jest *zakres* (ang. *range*), definiowany jako dane XML znajdujące się między dwoma punktami. Może to być przydatne na przykład w przypadku podświetlania obszaru tekstu, który może rozpoczynać się w jednym paragrafie i kończyć w drugim.

Ze względu na te dwa nowe typy wartością zwracaną w przypadku XPointer nie jest zbiór wierzchołków, jak to ma miejsce w przypadku wyrażeń XPath. Zamiast tego używany jest bardziej ogólny *zbiór lokalizacji* (ang. *location set*), gdzie *lokalizacja* jest definiowana jako punkt, zakres lub wierzchołek. Punkt jest reprezentowany przez parę obiektów: wierzchołek

kontenera (najbliższy element nadrzędny względem punktu) oraz liczbowy *indeks*, który zlicza liczbę znaków od początku treści wierzchołka kontenera do punktu. Zakres to po prostu dwa punkty a zawarte w nim informacje noszą nazwę *podzasobu* (ang. *subresource*).

Specyfikacja XPointer nie próbuje opisywać zachowania wskaźnika xpointer. Zwraca on po prostu listę wierzchołków lub ciągów znaków, które mają zostać poddane przetworzeniu, pozostawiając w gestii programisty kwestie funkcjonalne. Jest to dobre rozwiązanie, ponieważ dzięki temu XPointer może być używany na wiele różnych sposobów. W przypadku użycia w XLink opisywane informacje mogą być importowane do źródła docelowego lub pozostawione niezaładowane do momentu aktywowania łącza przez użytkownika. Całkowicie odmienne zastosowanie może polegać na użyciu wskaźników xpointer jako punktów zaczepienia dla komentarzy opisowych, przechowywanych w lokalnej bazie danych. Aplikacja użytkownika może wykorzystywać te informacje do wstawiania ikon do sformatowanych perspektyw dokumentu docelowego, które w momencie wybrania spowodują przywołanie kolejnego okna, zawierającego komentarz. Tak więc brak wyjaśnienia intencji używania języka XPointer powoduje zwiększenie jego elastyczności.

Składnia

Poniżej przedstawiono przykład schematu xpointer.

```
xpointer(id('blabla')/child::para[2])
```

W razie zakończenia powodzeniem zostanie zwrócony wierzchołek odpowiadający drugiemu potomkowi `<para>` elementu, którego atrybut `id` ma wartość `'blabla'`. W razie niepowodzenia zostanie zwrócony pusty zbiór lokalizacji.

Schematy i powiązane realizacje xpointer

Słowo kluczowe `xpointer` nosi nazwę *schematu* (ang. *schema*) i służy do identyfikowania metody składniowej oraz wyznaczenia granic zawartych danych. Danymi dla schematu `xpointer` jest wyrażenie języka XPath lub jego forma skrócona. Nie ma potrzeby ujmowania wyrażenia XPath w cudzysłów, gdyż nawiasy okrągłe w zupełności wystarczą do oznaczenia początku i końca.

Istnieje możliwość wiązania ze sobą wskaźników xpointer. Są one sprawdzane wówczas po kolei, do momentu aż któryś nie okaże się prawidłowy. Przykładowo:

```
xpointer(id('blabla'))xpointer(//*[@id='blabla'])
```

W tym przypadku dwa wskaźniki xpointer z semantycznego punktu widzenia oznaczają to samo. Pierwszy przypadek może nie dać pozytywnego rezultatu ze względu na fakt, że procesor XPointer może nieprawidłowo implementować funkcję `id()`. Mogłoby się tak zdarzyć w sytuacji, gdy procesor wymaga, aby definicja DTD określała, które atrybuty mają typ `ID` ale nie udostępniono by takiej definicji. Kiedy pierwsze wyrażenie zwraca błąd, proces przetwarzania przechodzi do kolejnego wskaźnika xpointer.

Co się stało z językiem XLink?

Plany opracowania języka XLink pojawiły się tuż po opublikowaniu specyfikacji XML. Wiązano z nim ogromne nadzieje. Ograniczenia związane z łączami języka HTML miały dać początek zupełnie nowym możliwościom — od możliwych do dostosowywania perspektyw nawigacji po zbiory dokumentów pochodzące od innych dostawców.

Rekomendacja dzieli się na dwa poziomy: podstawową i rozszerzoną. Rekomendacja podstawowa opisuje tradycyjne, wplatane łącza hipertekstowe, które są wszystkim znane. Łącza rozszerzone to doskonały nowo opracowany mechanizm, służący do opisywania łączy między zasobami z poziomu dowolnej strony a nawet innego dokumentu.

Obecnie minęły ponad dwa lata od czasu osiągnięcia przez XLink statusu rekomendacji (dotarcie do tego punktu zajęło cztery lata) i w zasadzie nie są dostępne jeszcze żadne implementacje. Żadna z obecnie dostępnych przeglądarek internetowych nie oferuje obsługi łączy rozszerzonych a tylko kilka obsługuje łącza proste.

Fakt, że standard XLink nie zdołał przyciągnąć uwagi programistów i użytkowników języka XML, może wynikać z popularności wbudowanych języków programowania w rodzaju JavaScript i Java. Kiedy XLink mozolnie przechodził kolejne etapy procesu normalizacji, producenci przeglądarek szybko dodawali obsługę różnych platform kodowania, co spowodowało wiele zamieszania, w tym zrodziło wiele problemów, których rozwiązanie miał zapewnić XLink.

Gdyby XLink pojawił się wcześniej, jego szanse powodzenia byłyby zapewne większe i prawdopodobnie pozwoliłoby to na oszczędzenie twórcom witryn internetowych wielu problemów. Wszystkie języki programowania (nawet Java!) stanowią rozwiązania zależne od platformy. Nie zawsze działają zgodnie z oczekiwaniami i nie są przystosowane do archiwizowania informacji przez dłuższy czas.

Prawdopodobnie przypadek języka XLink to przykład sytuacji, w której proces normalizacji nieco szwankuje. Zamiast zainspirować programistów do przejęcia najlepszych rozwiązań, udało się jedynie zainspirować powszechne ziewanie (w oczekiwaniu na jakieś rezultaty prac). Być może wynika to z faktu, że rekomendacja okazała się nie do końca prawidłowo podchodzić do problemu lub też okazała się stać w sprzeczności z planami marketingowymi komercyjnych dostawców oprogramowania. A może obsługa nowych funkcji nie uzasadniała ponoszenia kosztów związanych z ich implementacją? Osobie patrzącej z zewnątrz trudno udzielić jednoznacznej odpowiedzi.

Oprócz schematu `xpointer` są dostępne dwa inne schematy: `xmlns` oraz `element`. Celem schematu `xmlns` jest aktualizowanie bieżącego środowiska przetwarzania przy użyciu deklaracji nowej przestrzeni nazw. Poniższa deklaracja `xmlns` określa wartość prefiksu przestrzeni nazw, który jest używany w późniejszych wskaźnikach xpointer:

```
xmlns(foo=http://www.witryna.org.pl/)xpointer(//foo:cos)
```

Może się to wydawać dziwne ale schemat `xmlns` zwraca kod błędu i wymusza, aby przetwarzanie zostało przekazane do kolejnej części wskaźnika xpointer przy użyciu definicji prefiksu przestrzeni nazw `foo`.

Schemat `element` stanowi skróconą formę syntaktyczną. Reprezentuje on *n*-te dziecko elementu za pomocą samej liczby. Ciąg tego rodzaju liczb nosi nazwę *sekwencji potomków* (ang. *child sequence*) i jest zdefiniowany w rekomendacji XPointer element() Schema. W celu znalezienia trzeciego dziecka piątego dziecka elementu, którego `ID` jest `blabla`, można użyć następującego wskaźnika xpointer:

```
element(blabla/5/3)
```

Wskaźniki skrócone

Skrócony (ang. *shorthand*) wskaźnik xpointer zawiera jedynie ciąg, odpowiadający formie typu atrybutu ID. Zastępuje on składnik `id()`, co ułatwia czytanie i pisanie kodu. Poniższe dwa wskaźniki xpointer są równoważne:

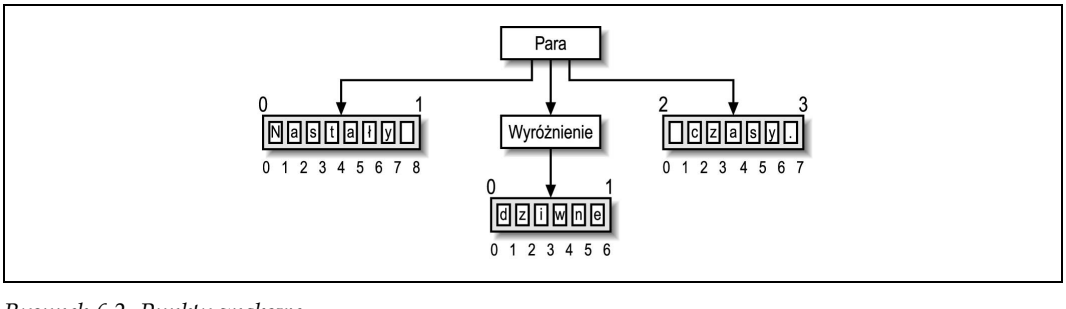
```
blabla
xpointer(id('blabla'))
```

Punkty

Punkt wewnątrz dokumentu jest reprezentowany przez dwa elementy: wierzchołek kontenera oraz indeks. Indeks zlicza liczbę punktów od początku wierzchołka, rozpoczynając od 0. Jeżeli punkt znajduje się wewnątrz tekstu, kontenerem jest wierzchołek tekstowy, w którym się on znajduje, a nie element zawierający tekst. Punkt może również leżeć poza tekstem, na przykład między dwoma elementami.

Na rysunku 6.2 przedstawiono sposób znajdowania indeksu dla punktów w niewielkim fragmencie kodu XML o następującej postaci:

```
<para>Nastały <wyróżnienie>dziwne</wyróżnienie> czasy.</para>
```



Rysunek 6.2. Punkty znakowe

Wewnątrz każdego wierzchołka tekstowego (oraz każdego wierzchołka nieposiadającego potomków) znajdują się punkty leżące między znakami tekstu, czyli *punkty znakowe* (ang. *character points*). Punkt znajdujący się na lewo od pierwszego znaku to zero. Ostatni

punkt znajduje się po ostatnim znaku wierzchołka tekstowego i jego indeks jest równy długości ciągu znaków. Istotną rzeczą jest pamiętanie, że pierwszy punkt pierwszego wierzchołka tekstowego w przedstawionym powyżej przykładzie nie jest równy pierwszemu punktowi elementu `para`. Są to dwa różne punkty.



XPath oraz XPointer wykorzystują kodowanie znaków UCS, natomiast DOM wykorzystuje kodowanie UTF-16, XML domyślnie wykorzystuje UTF-8. Może to powodować pewne problemy przy porównywaniu ciągów znaków. Przykładowo, to co jest jednym znakiem w przypadku ciągu znaków języka XPath, może być dwoma znakami w przypadku ciągu znaków DOM. Więcej informacji na temat kodowania znaków zawarto w rozdziale 9.

Wewnątrz każdego wierzchołka kontenera punkt, którego indeks wynosi zero jest nazywany *punktem początkowym* (ang. *start point*). Punkt o najwyższym indeksie nosi nazwę *punktu końcowego* (ang. *end point*). Zakres również posiada punkt początkowy i końcowy, ale nie muszą one należeć do tego samego wierzchołka kontenera.

Znaki sterujące

Język XPointer charakteryzuje się dość skomplikowanymi regułami użycia znaków sterujących. Jest to efekt uboczny, wynikający z faktu, że mogą one występować w różnych kontekstach. Na przykład w dokumencie XML mają zastosowanie reguły poprawności sformułowania. Tak więc znaki w rodzaju `<` i `&` muszą być reprezentowane za pomocą odpowiednich odwołań do encji znakowych.

Używając wskaźników xpointer zawsze należy zachować ostrożność w przypadku trzech znaków: okrągłego nawiasu otwierającego i zamykającego oraz znaku `^`. Nawiasy oznaczają początek i koniec danych zawartych w składniku lokalizacji, tak więc wszelkie nawiasy, które stanowią część danych, mogą powodować problemy w pracy parsera języka XPointer. Sposób rozwiązania tego problemu polega na poprzedzaniu ich znakiem `^`. Ze względu na fakt, że jest to znak sterujący, także należy go oznaczyć znakiem sterującym, jeśli ma występować w tekście. Wówczas wystarczy postawić przed nim jeszcze jeden znak `^`.

Jeżeli wskaźnik xpointer ma być używany w ramach odwołania URI, należy uwzględnić reguły użycia znaków sterujących, określone w dokumencie IETF RFC 2396. Według tego schematu pewne znaki są reprezentowane za pomocą symbolu procenta (`%`) oraz wartości szesnastkowej. Przykładowo, znak spacji zostaje zastąpiony symbolem `%20`, zaś znak procenta — symbolem `%25`.

Poniżej przedstawiono początkową postać wskaźnika xpointer:

```
xpointer(string-range(//para,"Używam nawiasów (często)."))
```

Koniecznym minimum jest użycie następujących znaków sterujących:

```
xpointer(string-range(//para,"Używam nawiasów ^(często^)."))
```

Jeżeli xpointer występuje w odwołaniu URI, należy użyć dodatkowych znaków sterujących (nie pomijając znaku ^):

```
xpointer(string-
range(//para,"U%c5%bcywam%20nawias%c3%b3w%20%5E(cz%c4%99sto%5E)."))
```

Funkcje języka XPointer

Język XPointer dziedziczy z języka XPath wszystkie funkcje i testy zdefiniowane w jego rekomendacji. Do tego zbioru są dodawane pewne dodatkowe funkcje związane z punktami i zakresami.

Konstruowanie zakresów

Funkcja `range-to()` tworzy zakres zaczynający się od wierzchołka kontekstowego i rozciągający do punktu podanego jako argument. Innymi słowy, tworzy ona zakres od ostatnio wykonanego kroku do kolejnego kroku.

Przykładowo, załóżmy, że posiadamy dokument, który definiuje pojęcia indeksu, z których każde zajmuje kilka stron. Element oznaczający początek zakresu to `indexterm` i posiada atrybut `class="poczakresu"`. Element kończący zakres jest tego samego typu ale posiada atrybut `class="konieczakresu"`. Poniższy wskaźnik xpointer tworzy zakres dla każdej pary takich elementów:

```
xpointer(indexterm[@class='poczakresu']/range-to(following::
indexterm[@class='konieczakresu']))
```

Zakresy od punktów i od wierzchołków

Funkcja `range()` zwraca zakres pokrywający dla każdej lokalizacji określonej w jej argumencie — zbiorze lokalizacji. *Zakres pokrywający* (ang. *covering range*) to zakres, który całkowicie zawiera daną lokalizację. W przypadku punktu punkt początkowy i końcowy zakresu są równe temu punktowi (zakres o długości zerowej, zakres *zwinięty*). Zakres pokrywający w przypadku zakresu to zakres jako taki (rozciągający się od tego samego punktu początkowego do tego samego punktu końcowego). W przypadku każdego innego obiektu zakres pokrywający zaczyna się w punkcie poprzedzającym ten obiekt i kończy w punkcie występującym po nim. Oba punkty należą do wierzchołka kontenera obiektu.

Funkcja `range-inside()` zamienia wierzchołki na zakresy. Dla każdego wierzchołka w danym zbiorze lokalizacji funkcja traktuje go jako wierzchołek kontenera i znajduje dla niego punkt początkowy oraz końcowy. Zakresy i punkty są przekazywane w niezmienionej postaci.

Zakresy od ciągów znaków

W celu utworzenia zakresów dla dowolnych obszarów tekstu można użyć funkcji `string--range()`. Funkcja ta posiada cztery argumenty wywołania.

1. Zbiór lokalizacji — pozycje, z których mają być wyszukiwane ciągi znaków.
2. Ciąg znaków — wzorzec, względem którego ma następować dopasowywanie.
3. Przesunięcie od początku dopasowania (domyślnie 1).
4. Długość wynikowego ciągu znaków, domyślnie równa długości wzorca z drugiego argumentu.

Przykładowo, poniższy wskaźnik xpointer zlokalizuje w dokumencie dziewiąte wystąpienie słowa „procent":

```
xpointer(string-range(/,"procent")[9])
```

Natomiast poniższy wskaźnik xpointer zwróci zakres dla ośmiu znaków występujących po ciągu znaków „nazwa użytkownika: " (z jedną spacją używaną w celach wyrównania tekstu) w elemencie o atrybucie `id="user123"`. (należy zwrócić uwagę, że indeksowanie w przypadku znaków jest różne niż w przypadku punktów — w tym przypadku pozycja pierwszego znaku to 1, a nie zero).

```
xpointer(string-range(id('user123'),"nazwa użytkownika: ",1,8))
```

Określenie długości (ostatni argument) jako 0 spowodowałoby utworzenie zakresu dla ciągu znaków o długości zerowej. Taki zakres zwinięty jest w efekcie równoważny pojedynczemu punktowi.

Interesującą cechą funkcji `string-range()` jest to, że ignoruje ona ograniczenia wierzchołków. Polega to na tym, że niejako cała treść zostaje zrzucona do pliku tekstowego przy jednoczesnym usunięciu znaczników XML. W efekcie wierzchołki tekstowe zostają scalone w jeden długi ciąg tekstowy. Tak więc w przypadku oznaczenia:

```
stara <em>chata</em>
```

poniższy wskaźnik xpointer będzie mu odpowiadał bez względu na to, czy znaczniki `<em>` zostaną usunięte, czy nie:

```
xpointer(string-range(/,"stara chata")
```

Znajdowanie punktów końcowych zakresów

Funkcje `start-range()` oraz `end-range()` lokalizują punkty na, odpowiednio, początku oraz na końcu zakresu. Każda z nich przyjmuje jeden argument — zbiór lokalizacji. Jeżeli zbiór ten jest punktem, zwracaną wartością jest punkt. W przypadku wierzchołków wartością tą jest punkt początkowy lub końcowy zakresu pokrywającego danego wierzchołka. Wywoływanie tych funkcji kończy się jednak niepowodzeniem w przypadku wierzchołków typów atrybutów i przestrzeni nazw.

Zwracanie punktów z dokumentów

Jeżeli wskaźnik xpointer znajduje się wewnątrz dokumentu XML, może korzystać z funkcji `here()` w celu reprezentowania swojej lokalizacji. Może to być przydatne na przykład w celu określenia punktu pochodzenia łącza. Jeżeli wskaźnik xpointer występuje w wierzchołku tekstowym w elemencie, zwracaną wartością jest element. W przeciwnym wypadku jest zwracany wierzchołek, który bezpośrednio zawiera wskaźnik. Ze względu na fakt, że xpointer może znajdować się w danym momencie tylko w jednym miejscu, w zbiorze lokalizacji jest zwracany tylko jeden element.

Kolejną funkcja jest `origin()`. Jest ona istotna tylko w przypadku użycia w kontekście łączy, kiedy zwraca lokalizację miejsca pochodzenia łącza (skąd użytkownik lub program rozpoczął przeglądanie). Jest to wymagane w przypadku złożonych typów łączy XLink, gdzie informacje o łączu nie są przechowywane w żadnym z punktów końcowych.