STUDIA INFORMATICA 2004
Volume 25 Number 3 (59)

Paweł CZYŻ, Lesław PAWLACZYK
Politechnika Śląska, Instytut Informatyki

# NATYWNE[1] SYSTEMY SKŁADOWANIA DOKUMENTÓW XML

**Streszczenie**. W artykule autorzy przedstawiają nową technologię składowania dokumentów XML. Artykuł opisuje pokrótce standard XML i jego specyfikę. W kolejnych rozdziałach omówiono technologię natywnych baz danych, standard dostępu do bazy o nazwie XML:DB i języki zapytań. Na końcu dokonano porównania istniejących rozwiązań.

**Słowa kluczowe**: XML, Xindice, eXist, XPath, XQuery, XML:DB.

# THE NATIVE XML STORAGE SYSTEMS

**Summary**. In the article authors present a new technology of storing XML documents. The article introduce shortly XML standard and it's specification. In the following chapters technology of native XML databases, access standard XML:DB and query languages are described. At the end the comparison of existing XML databases implementations is made.

**Keywords**: XML, Xindice, eXist, XPath, XQuery, XML:DB.

## 1. Standard *XML* – uniwersalny format danych

Język *XML – the eXtensive Markup Language* – ostatnio wyłonił się jako nowy standard reprezentacji danych i wymiany informacji poprzez sieć Internet. Wywodzi się bezpośrednio z języka *SGML* i zgodnie z założeniami projektowymi przejął jego znaczną cześć funkcjonalności. Został opracowany w roku 1998 przez niezależną organizację *World Wide Web Consortium* (*W3C*) [1], jest standardem otwartym. Wspierany jest przez największe firmy branży *IT*, takie jak *IBM, SUN, Microsoft*. Poparcie tych firm zapewniło standardowi *XML* globalny sukces.

*XML* jest metajęzykiem opisu elektronicznych dokumentów tekstowych. Oznacza to, że wspiera tworzenie nowych języków przystosowanych do opisu dokumentów o określonej zawartości. Dane zawarte w dokumencie mają reprezentację znakową, która jest czytelna dla człowieka bez zastosowania żadnych przekształceń. Dane są otoczone tekstowymi znacznikami, które opisują ich znaczenie. Zadaniem znaczników jest dostarczenie aplikacji informacji o danych poprzez ich związanie z dokumentem. Podstawową jednostką danych jest *element*, który może posiadać opcjonalnie *atrybuty*. *Elementy* są ułożone w hierarchiczną strukturę, która definiuje postać logiczną dokumentu. *XML* jest strukturalnym formatem danych, który pozwala na przechowywanie złożonych informacji zorientowanych tekstowo, binarnie lub obiektowo.

**Przykład:** dokument *XML*:

```
<book ISBN=”83-204-2743-6”>
    <author> Grandy Booch </author>
    <author> James Rumbaugh </author>
    <author> Ivar Jacobson </author>
    <title> UML Przewodnik użytkownika </title>
    <language> polski </language>
    <publisher> Wydawnictwo Naukowo-Techniczne </publisher>
    <date> 2002 </date>
 </book>
```

*XML* jest całkowicie niezależny od platformy sprzętowej i używanego oprogramowania, powyższą własność uzyskano poprzez wykorzystanie uniwersalnego zestawu znaków narodowych, czyli *Unicode*. Głównym obszarem zastosowań języka *XML* jest wspieranie wymiany informacji, zagadnienie to znajduje szerokie zastosowanie w systemach biznesowych wymieniających informacje w postaci dokumentów, dobrym przykładem są systemy elektronicznych transakcji finansowych.

## 2. Modele dokumentów *XML*

Dokumenty ze względu na logiczną strukturę można podzielić na zorientowane na dane lub tekst [2]. Dokumenty zorientowane na dane odznaczają się sztywną strukturą, natomiast dokumenty zorientowane na tekst posiadają bardzo luźną strukturę. W praktyce różnica pomiędzy poszczególnymi modelami dokumentów nie zawsze jest wyraźna. Na przykład, dokument zorientowany na dane, taki jak faktura, może zawierać dużo nieregularności w strukturach danych typu tekstowy opis produktu.

[1] Natywna baza danych to baza oparta na idei dostępu do pojedynczych rekordów

### 2.1. Dokumenty zorientowane na dane

Dokument *XML* jest jednym ze sposobów opisu strukturalnych danych. Poziom ich strukturalności dorównuje danym przechowywanym w modelu relacyjnym. Aby sprostać powyższemu wymaganiu, dokumenty *XML* muszą posiadać schemat, analogicznie jak schemat relacji w modelu relacyjnym. *Document Type Definition* (*DTD*s) [1] i *XML Schemas* [3] są mechanizmami, które są używane do specyfikowania poprawnych elementów, jakie mogą wystąpić w dokumencie.

**Przykład:** Dokument zorientowany na dane:

```
<Order key="1">
   <Customer>
      <Name>ABC S.A.</Name>
      <Email>parts@ABC.com</Email>
   </Customer>
   <Part color="czarny">
      <Key>32</Key>
      <Quantity>12</Quantity>
      <Price>321.20</Price>
    </Part>
</Order>
```

### 2.2. Dokumenty zorientowane na tekst

Dokumenty zorientowane na tekst zazwyczaj są odczytywane przez ludzi. Takimi dokumentami mogą być artykuły, książki, instrukcje obsługi. Są to dokumenty o bardzo luźnym schemacie bądź nawet nie posiadające go. Mogą również zawierać regularne struktury danych, takie jak nazwisko autora, wersja, data.

**Przyk³ad:** fragment sztuki Williama Shakespeare'a „Macbeth"

```
<ACT>
   <TITLE>ACT I</TITLE>
   <SCENE>
      <TITLE>SCENE I.  A desert place.</TITLE>
      <STAGEDIR>Thunder and lightning. Enter three Witches</STAGEDIR>
      <SPEECH>
         <SPEAKER>First Witch</SPEAKER>
         <LINE>When shall we three meet again</LINE>
         <LINE>In thunder, lightning, or in rain?</LINE>
      </SPEECH>
      <SPEECH>
         <SPEAKER>Second Witch</SPEAKER>
         <LINE>When the hurlyburly's done,</LINE>
         <LINE>When the battle's lost and won.</LINE>
      </SPEECH>
      <SPEECH>
```

```
         <SPEAKER>Third Witch</SPEAKER>
         <LINE>That will be ere the set of sun.</LINE>
         </SPEECH>
      <SPEECH>
         <SPEAKER>First Witch</SPEAKER>
         <LINE>Where the place?</LINE>
         </SPEECH>
      <SPEECH>
         <SPEAKER>Second Witch</SPEAKER>
         <LINE>Upon the heath.</LINE>
      </SPEECH>
...
   </SCENE>
...
</ACT>
```

Podsumowując, główna różnica pomiędzy dokumentami, obu wspomnianych wyżej typów, polega na stosowaniu w dokumentach zorientowanych na dane schematu, nazywanego arkuszem *DTD* lub *XSLT*. W przypadku dokumentów ukierunkowanych na tekst stosowanie takich arkuszy jest niewskazane, gdyż często nie posiadają one wyraźnej struktury, którą można byłoby opisać wspomnianymi metodami.

## 3. Natywne bazy danych *XML* (*NXD*)

W coraz liczniejszej grupie aplikacji wykorzystuje się dokument *XML* jako podstawowy nośnik informacji. Wraz ze wzrostem liczby przetwarzanych dokumentów bardziej złożonym problemem staje się zarządzanie nimi. Natywne bazy danych *XML (NXD – Native XML Databases)* są systemami zaprojektowanymi do rozwiązywania problemów z tej dziedziny. Analogicznie, do baz danych opartych na innym modelu, wspierają takie funkcje, jak język zapytań, bezpieczeństwo, dostęp wielu użytkowników, programistyczne API. Jedyną różnicą jest to, że ich model logiczny bazuje na *XML*, a nie na innej technologii.

Baza dostosowana do przechowywania danych *XML* daje możliwość efektywniejszego przechowywania oraz obsługiwania danych i metadanych. Dla szeregu aplikacji mechanizmy bazy danych *XML* będą często znacznie efektywniejsze od mechanizmów tradycyjnego przechowywania. Składa się na to wiele czynników: wygoda, łatwość projektowania i wydajność. Aplikacje idealnie dostosowane do użytkowania bazy danych *XML* to m.in. korporacyjne portale informacyjne, katalogi produktów, bazy danych części zamiennych, wymiana dokumentów typu *B2B (Business To Bisiness)*.

*NXD* są dogodnym rozwiązaniem problemu składowania, wydobywania, i przetwarzania trwałych dokumentów *XML*, ponieważ bezpośrednio operują na logicznej strukturze dokumentu, a zatem zachowują hierarchię i znaczenie danych. Na podstawie definicji wykreowanej przez inicjatywę *XML:DB* [7], natywne bazy danych *XML* definiują model dla dokumentów *XML* oraz przechowują i wydobywają dokumenty z nim zgodne. Fundamentalną jednostką składowania dla *NXD* jest dokument XML, analogicznie jak wiersz w tabeli dla relacyjnych baz danych.

Bardzo ważną cechą jest fakt, że *NXD* wspierają dokumenty zorientowane tekstowo, oznacza to, że zapewniają przechowywanie dokumentów *XML* bez z góry określonego schematu. Dokumenty nie muszą mieć powiązanego z nimi schematu lub definicji dokumentu. Jedynym wymaganiem jest to, aby były poprawnie sformatowane. Niektóre produkty umożliwiają walidację dokumentów przy użyciu technologii *DTD* [1] lub alternatywnej.

Wewnątrz bazy danych dokumenty są zarządzane w hierarchicznych kolekcjach, pozwalając na zadawanie zapytań i manipulacje nimi jak rekordami. Z punktu widzenia użytkownika, mechanizm ten jest porównywalny z przechowywaniem danych w systemie plików. Kolekcje mogą być dowolnie zagnieżdżane. Nie są one przypisane do predefiniowanego schematu, tak więc liczba i typy przechowywanych dokumentów w jednej kolekcji nie są narzucone.

## 4. Standard *XML:DB*

Czynnikiem zapewniającym stabilny rozwój technologii jest proces standaryzacji, który umożliwia konsolidację działań na rzecz rozwoju technologii. Organizacja *XML:DB* [7], której głównym zamierzeniem jest stworzenie przemysłowego standardu baz danych *XML*, stawia sobie następujące cele:

- Opracowanie specyfikacji technologicznej zarządzania danymi w bazie danych XML.
- Przygotowanie materiału do implementacji zaleceń takiej specyfikacji w modelu *Open Source License*.
- Utworzenie forum wymiany informacji, w ramach, którego dostawcy i użytkownicy baz danych *XML* będą mogli wymieniać wątpliwości i informacje z zakresu technologii i aplikacji *XML*.

Promotorami tej inicjatywy są *SMB GmBH, dbXML Group L.L.C* i *OpenHealth Care Group*. Inicjatywę wspiera też wiele innych organizacji. *XML:DB API* zostało zaprojektowane, aby umożliwić powszechny mechanizm dostępu do baz danych *XML* z poziomu języka *Java*. *API* wspiera konstruowanie przenośnych, nadających się do

wielokrotnego użycia aplikacji, wykorzystujących dostęp do natywnych baz danych *XML (NXD)*. Omawiane *API* może być postrzegane ogólnie jako rozwiązanie realizujące analogiczne funkcje do technologii, takich jak *ODBC, JDBC*.

Podstawowymi składnikami *XML:DB API* [7] są: sterowniki, kolekcje, zasoby i usługi. Sterowniki zawierają mechanizmy dostępu do bazy danych *XML* specyficznego producenta. Kolekcje są kontenerami, zorganizowanymi w hierarchiczną strukturę, zawierającą zasoby, czyli dokumenty *XML*, oraz zagnieżdżone kolekcje. Kolekcje udostępniają usługi, pozwalające wykonywać operacje na nich, takie jak wykonywanie zapytań lub aktualizacja danych. Zasób reprezentuje dokument *XML* albo fragment dokumentu *XML*, wybrany na przykład przez zapytanie *XPath* [4].

### 4.1. Nawiązywanie połączenia z *NXD*

Ponieważ wszystkie nowoczesne technologie związane z *XML* są określane jako niezależne od platformy sprzętowej, najczęściej stosowanym językiem do implementacji oraz korzystania z tych rozwiązań jest język *Java*. Poniżej przedstawiony jest fragment kodu, który pokazuje, jak połączyć się z instancją bazy danych *eXist* [9].

```
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import org.xmldb.api.*;

public class RetrieveExample {
      protected static String URI =
             "xmldb:exist://localhost:8080/exist/xmlrpc";
      protected static String col_name = "/db/books";

    public static void main(String args[]) throws Exception {
        String driver = "org.exist.xmldb.DatabaseImpl";

        // inicjalizacja sterownika bazy danych
        Class cl = Class.forName(driver);
        Database database = (Database) cl.newInstance();
        DatabaseManager.registerDatabase(database);
```

Pierwsze linie kodu importują wymagane klasy dla *XML:DB API*. W kolejnym kroku tworzona jest instancja sterownika, która implementuje klasę *Database*. Następnie obiekt realizujący sterownik bazy danych jest rejestrowany przez zarządcę bazy danych (*DatabaseManager*).

```
        // pobierz kolekcjê
        Collection col = DatabaseManager.getCollection(URI + col_name);
```

```
        col.setProperty("pretty", "false");
```

Później pobierany jest obiekt klasy *Collection* reprezentujący kolekcję. Realizowane jest to poprzez wywołanie statycznej metody *DatabaseManger.getCollection()*. Metoda ta pobiera kwalifikowany parametr *URI,* który identyfikuje docelową kolekcję. Wywołania *setProperty()* określają parametry bazy danych. W rozważanym wypadku własność *„pretty”* jest ustawiona w celu wyboru standardowej strony kodowej *UTF-8*.

```
      // pobierz dokument XML
        XMLResource res = (XMLResource)col.getResource("hamlet.xml");
        if (res == null)
            System.out.println("document not found!");
        else
            System.out.println(res.getContent());    }
}
```

Wywołanie *col.getResource()* ostatecznie wydobywa dokument, który zwracany jako obiekt klasy *XMLResource*. Metodę *getContent()* zwraca zawartość zasobu XML.

### 4.2. Zapytania

Aby zadać zapytanie do bazy danych, należy pobrać obiekt usługi *XPathQueryService* z zarządcy bazy danych. *XML:DB API* definiuje kilka rodzajów usług, które może dostarczać baza danych. Metoda typu *getService()* klasy *Collection* zwraca obiekt odpowiedzialny za komunikację z daną usługą, jeżeli jest dostępna. Metoda jako pierwszy argument pobiera nazwę usługi, a jako następny jej wersję.

Do realizacji zapytania służy metoda *service.query(xpath)*. Zwraca ona obiekt klasy *ResourceSet*, zawierający zasoby odnalezione przez zapytanie. Funkcja *ResourceSet.getIterator()* zwraca obiekt typu *Iterator* dla tych zasobów. Każdy zasób zawiera pojedynczy fragment dokumentu albo wartość wyselekcjonowaną przez wyrażenie *XPath*.

```
        String xpath = ”/PLAY/TITLE”;
        XPathQueryService service =
            (XPathQueryService) col.getService("XPathQueryService", "1.0");
        service.setProperty("pretty", "true");

        ResourceSet result = service.query(xpath);
        ResourceIterator i = result.getIterator();
        while(i.hasMoreResources()) {
            Resource r = i.nextResource();
            System.out.println((String)r.getContent());}
```

# 5. Języki zapytań *NXD*

Głównymi językami zapytań, które stały się standardem są *XPath* [6] i *XQuery* [5]. Zgodnie z wymaganiami sprecyzowanymi przez organizację *W3C* [4] język zapytań operujący na dokumentach *XML* powinien zawierać: przejrzystą i jasną semantykę, obsługę wyrażeń typu ścieżek w zapytaniu, zdolność do zwracania dokumentu *XML*, zdolność zadawania zapytań i zwracanie elementów i atrybutów *XML*, rozróżnianie typów danych takich jak numeryczne, logiczne, łańcuchy znakowe.

## 5.1. XML Path Language (XPath)

*XPath* (*XML* Path Language) został zaproponowany przez *W3C* jako język adresowania fragmentów dokumentu *XML*. Należy zaznaczyć, że nie był on projektowany jako język zapytań baz danych. Wyrażenia *XPath* są również wykorzystywane jako podstawowe elementy w innych *XML*'owych językach zapytań, takich jak *XQuery*, *XSLT*, *XPointer* [1,5].

*XPath* operuje na abstrakcyjnej, logicznej strukturze dokumentu *XML*. Został zaprojektowany do operowania na pojedynczym dokumencie, którego strukturę widzi się jako *drzewo* złożone z *węzłów*. Umożliwia on zlokalizowanie ściśle określonego *elementu*, jest językiem pozwalającym na łatwą nawigację po hierarchicznej strukturze dokumentu *XML*. *XPath* wykorzystuje zwięzłą, nie *XML*'ową składnię, którą można porównać z hierarchicznymi ścieżkami, używanymi w celu adresacji części zasobów systemu plików.

*XPath* nie spełnia całkowicie funkcji języka zapytań baz danych *XML*, ponieważ może przetwarzać tylko jeden dokument na raz, nie ma udogodnień pozwalających na złączenia albo grupowanie.

**Przykłady:**

1. Zapytanie wykonane na kolekcji przechowującej sztuki Williama Shakespeare'a
   - */PLAY/TITLE* – Wybranie tytułów sztuk znajdujących się w kolekcji.
   - *//SPEACH[SPEAKER='Macbeth']* – Odnalezienie wszystkich mów zawierających jako mówcę Macbetha.
2. Zapytanie wykonane na kolekcji przechowującej faktury zamówień
   - */Order[Part[Key = 32]/Quantity > 10 ]/Customer* - Odnalezienie danych o klientach, którzy zamówili więcej niż 10 części o identyfikatorze 32.

## 5.2. XML Query Language (*XQuery*)

Język *XQuery* [5] jest próbą sprostania narastającym potrzebom standaryzowania mechanizmu wyszukiwania danych w dokumentach *XML*. Został opracowany przez

organizację *W3C*. Zapewnia tak samo szeroką funkcjonalność jak *SQL* dla relacyjnych baz danych. *XQuery* jest językiem funkcyjnym, w którym każde zapytanie jest wyrażeniem. Wyrażenie *XQuery* składa się z siedmiu obszernych typów: wyrażenia *XPath*, elementu konstruktora, wyrażenia *FLWR*, wyrażenia wykorzystującego operatory i funkcje, wyrażenia kwalifikowanego i wyrażenia testującego bądź modyfikującego dane. Składnia i gramatyka różnych wyrażeń *XQuery* mogą się znacząco różnić, co jest przejawem odmiennych wpływów podczas projektowania języka *XQuery*.

Podstawowa forma zapytania *XQuery* składa się z wyrażenia *FLWR*. Jest ono bardzo podobne do konstrukcji *SELECT-FROM-WHERE* języka SQL. Wyrażenie *FLWR* składa się z klauzul *FOR, LET, WHERE i RETURN*:

- *FOR*: Przypisuje zmiennej jeden lub wiele elementów dokumentu XML, spełniających kryteria wyrażenia *XPath*, będącego argumentem klauzuli.
- *LET*: Przypisuje zmiennej jeden element dokumentu, realizujący kryteria umieszczone w argumencie klauzuli.
- *WHERE*: zawiera jeden lub więcej predykatów, które filtrują albo ograniczają zbiór węzłów wygenerowany przez klauzule *FOR* i *LET*.
- *RETURN*: generuje wynik wyrażenia *FLWR*. Klauzula *RETURN* zazwyczaj zawiera odwołanie do zmiennej i jest wykonywana jeden raz dla każdego związanego z nią węzła, który został zwrócony przez poprzednie klauzule.

Predykaty mogą być grupowane wykorzystując nawiasy i następujące typy operatorów: porównania (=, !=, >, < …), mnogościowych *(EXISTS, EVERY, SOME*), logicznych (*AND, OR, NOT*), arytmetycznych (suma, różnica, iloczyn, iloraz, modulo), zawierania (*UNION, INTERSECT, EXCEPT*). Wyrażenie *IF-THEN-ELSE* również może być wykorzystane. W klauzuli *WHERE* złączenia mogą być wykonane poprzez porównania wartości, które są zawracane przez zapytania zagnieżdżone. Dozwolone są funkcje agregujące, włączając: *COUNT, SUM, AVG, MAX i MIN*. Mogą być zastosowane w klauzulach *LET, WHERE, RETURN*, również mogą być połączone ze słowem kluczowym *DISTINCT*. Sortowanie jest wykonywane przy pomocy klauzuli *SORT*.

**Przykład**: Odnalezienie wszystkich numerów części, których cena jest większa od średniej ceny wszystkich części.

```
for $b in //Part
let $avg := avg(//Part/Price)
where $b/Price > $avg
return $b/Key
```

### 5.3. XML Update Language (XUpdate)

Aktualizacje dokumentów *XML* są słabym ogniwem aktualnych implementacji *NXD*. Większość produktów wymaga wydobycia dokumentu, zmodyfikowania go, a następnie ponownego umieszczenia w bazie danych. Nieliczne natywne bazy danych *XML* wspierają *XUpdate* – język, pozwalający na dokonywanie uaktualnień w obrębie serwera, opracowany przez inicjatywę *XML:DB* [8].

*XUpdate* posiada zapis składni w języku *XML*, wykorzystuje wyrażenia *XPath* w celu wybrania węzłów do dalszego przetwarzania. Dostarczane są następujące typy operacji:

- *xupdate:insert* – pozwala wstawić *węzeł* bezpośrednio do dokumentu *XML*,
- *xupdate:append* – pozwala utworzyć nowy *węzeł* i dołączyć go do istniejącego,
- *xupdate:update* – pozwala uaktualnić zawartość istniejącego *węzła*,
- *xupdate:remove* – pozwala usunąć *element*,
- *xupdate:rename* – pozwala zmienić nazwę *atrybut* albo *element* po jego utworzeniu.

## 6. Przykładowe implementacje *NXD*

### 6.1. eXist

*eXist* [9] jest projektem *OpenSource*, mającym na celu opracowanie natywnego systemu składowania dokumentów *XML*, który może zostać zintegrowany z aplikacjami operujących na danych *XML*, w różnorodnych możliwych scenariuszach, od opartych na sieci *WWW* aplikacjach do systemów dokumentacyjnych uruchamianych z płyt *CD*. Baza danych jest zaimplementowana całkowicie w języku *Java* i może być wykorzystywana na wiele sposobów, uruchamiana jako samodzielny serwer, w serwlecie albo jako bezpośrednio wbudowywana w aplikację.

*eXist* dostarcza system składowania dokumentów *XML* bez schematu. Oznacza to, że składowane dokumenty nie muszą być zgodne z określonym schematem lub definicją typu dokumentu, warunkiem wystarczającym do zapisu jest ich poprawne sformatowanie. Istnieje też możliwość walidacji zgodności dokumentu *XML* ze schematem, w tym celu należy określić schemat kolekcji, wykorzystując *XML Schema* lub *DTD*.

Baza danych aktualnie jest najlepiej dopasowana do aplikacji operujących na dokumentach *XML*, które są sporadycznie uaktualniane. *eXist* implementuje liczne rozszerzenia standardu *XPath,* aby poprawić efektywność przetwarzania zapytań operujących na tekście, włączając w to wyszukiwanie słów kluczowych i zapytania o bliskość szukanych elementów.

Używając rozszerzonej składni języka *XPath*, użytkownik może zadawać zapytania operujących na odrębnych częściach hierarchii kolekcji, a nawet wszystkich dokumentach zgromadzonych w bazie danych. Pomimo implementacji w *Javie* motor zapytań *eXista* realizuje efektywny, oparty na indeksach, proces interpretacji zapytań. Rozszerzony schemat indeksowania wspiera szybką identyfikację strukturalnych relacji pomiędzy *elementami* dokumentu, takich jak rodzic-dziecko, przodek-potomek, poprzedni krewny, następny krewny. Bazując na algorytmach złączeń i szerokim zakresie wyrażeń *XPath*, zapytania są przetwarzane, wykorzystując tylko informacje zawarte w indeksach. Aktualnie trwają intensywne prace nad implementacją znacznie bogatszego języka zapytań *XQuery*.

*eXist* dostarcza mechanizmu uaktualniania zawartości dokumentów *XML* po stronie serwera, rozwiązanie to bazuje na języku *XUpdate* opracowanym przez inicjatywę *XML:DB*. Dostęp do bazy danych został zrealizowany poprzez technologie *HTTP*, *XML-RPC, SOAP* i *WebDAV*. Aplikacje napisane w *Javie* mogą wykorzystać jedną z dwóch implementacji *XML:DB API*. Pierwsza służy do komunikowania się z odległą bazą danych poprzez wywołania *XML-RPC*, druga komunikuje się poprzez bezpośredni dostęp do pracującej lokalnie instancji *NXD*.

*eXist* wspiera podstawowe mechanizmy autoryzacji i kontroli dostępu. Schemat uprawnień bazuje na rozwiązaniach zaczerpniętych z systemów *Unix*. Implementacja *XML:DB API* pozwala na autoryzację użytkowników i posiada rozszerzenia umożliwiające na zarządzanie nimi oraz zasobami.

### 6.2. Xindice

*Xidince* [10], również znany pod nazwą *dbXML*, jest częścią projektu *Apache XML*. Jest to *OpenSource*'owa baza danych rozwijana przez ochotników na licencji *Apache*. Motor bazy danych jest napisany całkowicie w języku *Java* i wymaga do pracy *Wirtualnej Maszyny Java* (*JVM*). *Xindice* może być uruchomiony w *JVM* na dwa sposoby:

- Tryb wbudowany – aplikacja *Javy* tworzy instancję *Xindice* we własnej *JVM* i tylko ta aplikacja posiada dostęp do danych, znajdujących się w bazie danych.
- Tryb serwera – *Xindice* jest uruchamiany jako standardowa aplikacja sieciowa *J2EE*, w dowolnym serwerze aplikacji (*Tomcat*). W tym trybie *Xindice* jest wykonywany w *JVM* serwera aplikacji. Klienci mogą łączyć się z bazą danych, wykorzystując protokół *XML-RPC*, który opiera się na protokole *HTTP*.

Aktualnie *Xindice* wspiera język *XPath* i *XUpdate*. Implementacja *XPath* została zapożyczona z projektu *Apache Xalan*, który jest zgodny z zaleceniami *W3C*. Specyfikacja *XPath* nie wspiera zapytań operujących na wielu kolekcjach, oraz zapytań rekursywnych. Mechanizm indeksowania jest ważnym elementem pozwalającym przyśpieszyć

przetwarzanie zapytań. *Xindice* posiada wsparcie dla ręcznego mechanizmu tworzenia indeksów, co oznacza, że dla każdego elementu włączonego do indeksowania należy ręcznie skonfigurować bazę danych.

*Xindice* najefektywniej przetwarza dokumenty niewielkich rozmiarów, w przypadku większych gwałtownie wzrasta zapotrzebowanie na pamięć, co jest następstwem przyjętego mechanizmu indeksowania. Wynika to z faktu, że dokumenty są składowane jako całość i za każdym razem, gdy baza danych wymaga odczytania lub zapisu danych, cały dokument jest wczytywany do pamięci operacyjnej. *Xidince* obsługuje *XML:DB API* dla języka *Java*, oraz niezależne od języka *XML-RPC API*.

## 7. Zakończenie

W artykule przedstawiono krótko technologię znaną pod nazwą *NXD (Native XML Database)*. We współczesnym świecie, gdzie liczba przetwarzanych informacji rośnie wykładniczo, nie zawsze jest czas, by projektować relacyjne schematy baz danych. Z tym problemem radzą sobie bazy *NXD*, które gromadzą dokumenty *XML* w kolekcjach. Przedstawione rozwiązania *eXist* oraz *Xindice* są szeroko znanymi i stosowanymi implementacjami *NXD*. Rozwiązania takich baz danych stają się coraz bardziej wyrafinowane i specjalizowane, pojawiają się znane w relacyjnych bazach pojęcia: walidacji i formatu danych, wykonywania zapytań i aktualizacji dokumentów. Niestety, znane autorom, implementacje nie są wolne od błędów, są to rozwiązania ciągle rozwijane i zmieniające się praktycznie z dnia na dzień. Należy jednak podkreślić, że technologia ta ma już swoje szerokie zastosowanie w korporacyjnych aplikacjach masowego przetwarzania dokumentów.

**LITERATURA**

1. WWW Consortium. XML 1.0 Recommendation. 11 lutego 2004. <http://www.w3.org/TR/1998/REC-xml-19980210>
2. Bourett R.: XML and Databases. 11 lutego 2004. <http://www.rpbourret.com/xml/XMLAndDatabases.htm>
3. Vlist E.: Using W3C XML Schema. 11 lutego 2004. <http://www.xml.com/pub/a/2000/12/13/schemas/part2.html>
4. Chamberlain D. et al.: XML Query Requirements. 11 lutego 2004. <http://www.w3.org/TR/2000/WD-xmlquery-req-20000815>

5. Boag S. et al.: XQuery 1.0: An XML Query Language. <http://www.w3.org/TR/xquery/>
6. Clark J., DeRose S.: XML Path Language (XPath) Version 1.0. 11 lutego 2004. <http://www.w3.org/TR/xpath>
7. XML:DB Initiative for XML Databases. 11 lutego 2004. <www.xmldb.org>
8. XML:DB XUpdate – XML Update Language. 11 lutego 2004. <www.xmldb.org/xupdate>
9. Meier W.: eXist: An Open Source Native XML Database. 11 lutego 2004. <http://exist.sourceforge.net>
10. The Apache Group.: Xindice Native XML Database. 11 lutego 2004. <http://xml.apache.org/xindice>

Recenzent: Dr inż. Maciej Bargielski

Wpłynęło do Redakcji 9 marca 2004 r.

**Abstract**

This article concerns on the topic of native XML databases (NXD). Authors introduce a basic knowledge about XML and type of documents. Several examples of XML are shown dividing them to data and text centric documents. The assumptions and standards of access to NXD is given. Few samples of Java code are shown concerning retrieving an XML document from NXD and connection establishing between client and server. Two different query methods of NXD are described, samples of XPath and XQuery statements are given with a short depiction of syntax. Authors reviewed two important implementations of NXD: eXist and Xindice. General advantages and drawbacks of mentioned XML storage solutions are discussed. At the end of the article future plans in the area of NXD are mentioned.

**Adresy**

Paweł CZYŻ: pczyz@polsl.gliwice.pl .
Lesław PAWLACZYK Politechnika Śląska, Instytut Informatyki, ul. Akademicka 16, 44-101 Gliwice, Polska, palles@polsl.gliwice.pl .