XML DATABASES AND EFFECTIVE SEARCH ENGINE

R. SRINIVASAN

A dissertation submitted to the

FACULTY OF SCIENCE AND HUMANITIES

in partial fulfillment of the requirements
for the award of the degree of

MASTER OF SCIENCE (INTEGRATED)
in
COMPUTER SCIENCE

ANNA UNIVERSITY
CHENNAI – 600 025
APRIL 2007

BONAFIDE CERTIFICATE

Certified that this dissertation entitled “**XML DATABASES AND EFFECTIVE SEARCH ENGINE**” is the bonafide work of **Mr.R.SRINIVASAN** who carried out the project under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

Dr., K.EKAMBAVANAN

Professor and Head,
Department of Mathematics,
Anna University,
Chennai-600 025

Mr., D.GEORGE WASHINGTON

Internal Guide,
Ramanujam Computing Centre,
Anna University,
Chennai-600 025

ABSTRACT

This project entitled “XML Databases and Effective search Engine” was developed at IIT Madras.

The aim of this project is to develop a search engine for XML and to improve the processing of XML files by writing a sophisticated API in java.

X-Search is a search engine for XML, which returns related documents for a given search string. X-Search ranks these search results and uses indexing techniques to facilitate efficient retrieval. The performances of different techniques were tested experimentally. These experiments indicate that X-Search is efficient, scalable and ranks results correctly.

A more sophisticated API was written in Java for processing XML which includes converting a database table into XML, parsing XML and inserting them in the corresponding database table and converting HTML to XML .This API solves the problem of storing and managing XML.

Tamil

ACKNOWLEDGEMENT

It gives me great pleasure to pay gratitude to various individuals who motivated and encouraged me in my academic achievements.

I am indebted to, DR. K. EKAMBAVANAN, Head of the department, Department of Mathematics, Anna University for permitting me to undertake this project work.

I further extend my deepest gratitude to Mr.D.GEORGE WASHINGTON, Ramanujam Computing Centre, Anna University for his excellent guidance and co-operation, cheerful encouragement and inspiration during the course of this project.

I am gratified in thanking PROF S. SUNDAR, Department of Mathematics, Indian Institute of Technology, Madras for granting his valuable consent for undertaking this project work in the esteemed organization.

I am grateful to **Dr.P.** DEVARAJ, Department of Mathematics, Anna University for his support during the course of this project.

I convey my thanks to my parents and friends for their great support during the course of the project.

(SRINIVASAN, R)

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF ABBREVIATIONS

XML	Extensible Markup Language
HTML	Hyper Text Markup Language
API	Application Programming Interface
JDBC	Java Database Connectivity
ODBC	Open Database Connectivity
JVM	Java Virtual Machine
RDBMS	Relational Database Management System
IR	Information Retrieval
XML-IR	Extensible Markup Language Information Retrieval
OODB	Object Oriented Databases
IDE	Integrated Development Environment
SGML	Standard Generalized Markup Language
DOM	Document Object Model
WWW	World Wide Web
PDF	Portable Document Format
X-SEARCH	Extensible Markup Language Search
W3C	World Wide Web Consortium

CHAPTER 1

INTRODUCTION

1.1 ORGANIZATION PROFILE

The Indian Institute of Technology Madras (IIT Madras) is an engineering university located in Adyar the southern part of Chennai city in Southern India.

Founded in 1959 with technical and financial assistance from the Government of the erstwhile West Germany, IIT Madras was third among the seven Indian Institutes of Technologies established by the Government of India, through an Act of Parliament, to provide world class education and research facilities in engineering and technology.

The Institute has several departments and advanced research centers in various disciplines of engineering and the pure sciences, with nearly 100 laboratories organized on a unique pattern of functioning.

Faculties of international repute, a brilliant student community, excellent technical and supporting staff and an effective administration have all contributed to the pre-eminent status of IIT Madras.

1.2 PROBLEM FORMULATION

XML is a widely accepted standard for exchanging business data. To optimize the management of XML and help companies build up their business partner networks over the Internet, database servers have introduced new XML storage and query features. However, each enterprise defines its own data elements in XML and modifies the XML documents to handle the evolving business needs. This makes XML data conform to heterogeneous schemas or schemas that evolve over time, which is not suitable for XML database storage.

We analyzed the current XML database strategies and present an API for XML processing; enabling databases to handle multiple XML formats seamlessly.

Most XML-enabled database management systems can only translate a few relations into an XML document. Moreover, the translation is without data semantics constraints consideration, which may not be sufficient for information highway on the Web. The demand on database is increased in e-commerce. Not only relational database (RDB) is needed for traditional data processing, but also its equivalent XML documents (database) are needed for B2B applications. Therefore, performance for online conversion from relational data to XML document is an issue.

This project aims to create an API for XML processing to improve database performance. For any successful update to the RDB, corresponding update will be applied to its replicate XML database. The result is an incrementally maintained XML database for efficient and effective computing. Using this API, a search engine for XML can be developed which will require an indexing mechanism for efficient search.

1.3 INFORMATION RETRIEVAL

Information retrieval (IR) is the science of searching for information in documents, searching for documents themselves, searching for metadata which describe documents, or searching within databases, whether relational stand-alone databases or hypertext networked databases such as the Internet or World Wide Web or intranets, for text, sound, images or data.

There is a common confusion, however, between data retrieval, document retrieval, information retrieval, and text retrieval, and each of these has its own bodies of literature, theory, praxis and technologies. IR is like most nascent fields interdisciplinary, based on computer science, mathematics, library science, information science, cognitive psychology, linguistics, statistics, and physics.

Automated IR systems are used to reduce information overload. Many universities and public libraries use IR systems to provide access to books, journals, and other documents. IR systems are often related to object and query. Queries are formal statements of information needs that are put to an IR system by the user.

An object is an entity which keeps or stores information in a database. User queries are matched to objects stored in the database. A document is, therefore, a data object. Often the documents themselves are not kept or stored directly in the IR system, but are instead represented in the system by document surrogates.

Information or data of 3 types

1. Unstructured Data (e.g. html files)
2. Structured Data (e.g. RDBMS)
3. Semi-structured Data (e.g. XML files)

Web search engines such as Google, Live.com, or Yahoo search are the most visible IR applications.

1.4 XML-IR

3 Approaches:

1. Database-Oriented Approach
2. IR-based Approach
3. Hybrid Approach

1.4.1 Database-Oriented Approach

- 75% of web-data come from Databases
- Store data in XML format

Advantages

1. Data relations, constraints and integrity can be modeled and checked
2. Query languages are similar to SQL
3. Standard (RDB and OODB) database engines

Disadvantages

1. Different databases with different entity-relationship structures
2. DTDs have very poor data and referential integrity constraints

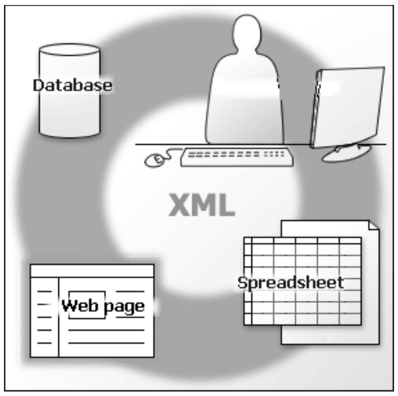


Figure 1.4.1 XML Information Retrieval

CHAPTER 2

TECHNOLOGY OVERVIEW

2.1 INTRODUCTION

XML was originally designed to exchange data over the World Wide Web Currently, it has become a public and widely accepted standard. Now it is used for exchanging data between any numbers of computer systems.

2.2 JAVA PROGRAMMING LANGUAGE

Java is a reflective, object-oriented programming language developed by Sun Microsystems. Initially called Oak, it was intended to replace C++, although the feature set better resembles that of Objective-C. Sun Microsystems currently maintains and updates Java regularly.

Specifications of the Java language, the Java Virtual Machine (JVM) and the Java API are community-maintained through the Sun-managed Java Community Process. Java was developed in 1991 by James Gosling and other Sun engineers, as part of the Green Project.

2.2.1 Language characteristics

1. Object-oriented programming methodology.
2. Platform-Independence.
3. Built-in support for using computer networks.
4. Designed to execute code from remote sources securely.

2.3 ECLIPSE (SOFTWARE)

Eclipse is a free software / open source platform-independent software framework for delivering what the project calls "rich-client applications", as opposed to "thin client" browser-based applications. So far this framework has typically been used to develop IDEs, such as the highly-regarded Java IDE called Java Development Toolkit and compiler that comes as part of Eclipse. However, it can be used for other types of client application as well.

2.4 XML

The XML is a W3C-recommended general-purpose markup language for creating special-purpose markup languages. It is a simplified subset of SGML, capable of describing many different kinds of data.

Its primary purpose is to facilitate the sharing of data across different systems, particularly systems connected via the Internet.

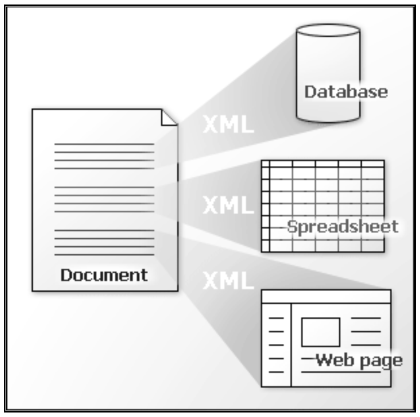


Figure 2.4.1 XML Representing Multiple data Formats

2.4.1 Goals of XML

1. XML shall be straightforwardly usable over the Internet
2. XML shall support a wide variety of applications
3. XML shall be compatible with SGML
4. It shall be easy to write programs which process XML documents
5. XML documents should be human-legible and reasonably clear
6. The XML design can be prepared quickly
7. The design of XML shall be formal and concise
8. XML documents shall be easy to create

2.5 CORRECTNESS IN AN XML DOCUMENT

For an XML document to be correct, it must be:

- Well-formed. A well-formed document conforms to all of XML's syntax rules. For example, if a non-empty element has an opening tag with no closing tag, it is not well-formed. A document that is not well-formed is not considered to be XML; a parser is required to refuse to process it.

- Valid. A valid document has data that conforms to a particular set of user-defined content rules that describe correct data values and locations.

  For example, if an element in a document is required to contain text that can be interpreted as being an integer numeric value, and it instead has the text "hello", is empty, or has other elements in its content, then the document is not valid.

2.6 DTD

The oldest schema format for XML is the Document Type Definition (DTD), inherited from SGML. While DTD support is ubiquitous due to its inclusion in the XML 1.0 standard, it is seen as limited for the following reasons:

- It has no support for newer features of XML, most importantly namespaces.
- It lacks expressivity. Certain formal aspects of an XML document cannot be captured in a DTD.

It uses custom non-XML syntax, inherited from SGML, to describe the schema

2.7 PROCESSING XML FILES

SAX and DOM is APIs widely used to process XML data. SAX is used for serial processing whereas DOM is used for random-access processing. Another form of XML Processing API is data binding, where XML data is made available as a strongly typed programming language data structure, in contrast to the DOM.

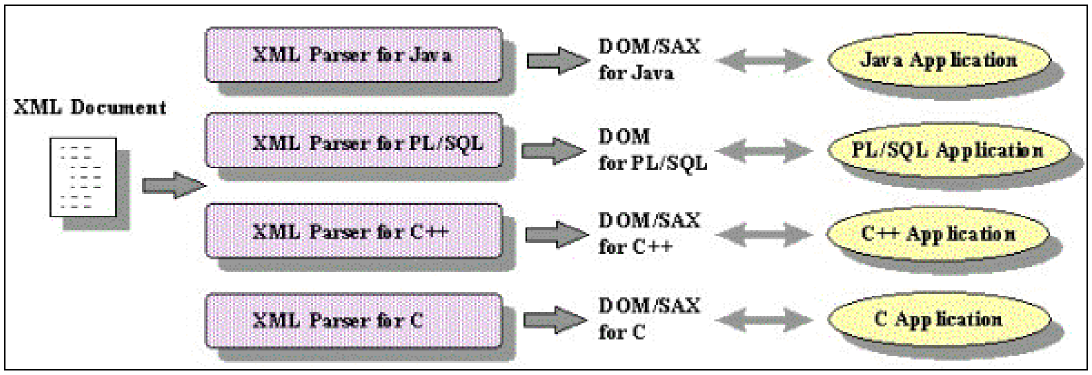


Figure 2.7.1 Different XML Parsers

2.7.1 Java API for XML Processing

The Java API for XML Processing, or JAXP, is one of the Java XML programming APIs. It provides the capability of validating and parsing XML documents. The two basic parsing interfaces are:

- the Document Object Model parsing interface or DOM interface
- the Simple API for XML parsing interface or SAX interface

CHAPTER 3

XML DATABASES

3.1 INTRODUCTION

An XML database is a data persistence software system that allows data to be imported, accessed and exported in the XML format.

Two major classes of XML database exist:

1. Native XML Databases
2. XML-enabled Databases

XML-enabled Databases map all XML to a traditional database such as relational databases, accepting XML as input and rendering XML as output. Native XML (NXD) the internal model depends on XML and uses XML documents as the fundamental unit of storage.

3.2 RATIONALE FOR XML IN DATABASES

O'Connell states that: the increasingly common use of XML for data transport, which has meant that "data is extracted from databases and put into XML documents and vice-versa". It may prove more efficient in terms of conversion costs and easier to store the data in XML format.

3.3 NATIVE XML DATABASES

People may need more features than are offered by one of the simple systems described above; then we should consider a native XML database. Native XML databases are databases designed especially to store XML documents.

Like other databases, they support features like transactions, security, multi-user access, programmatic APIs, query languages, and so on. The only difference from other databases is that their internal model is based on XML and not something else, such as the relational model.

Native XML databases are most commonly used to store document-centric documents. The main reason for this is their support of XML query languages, which allow you to ask questions like, "Get me all documents in which the third paragraph after the start of the section contains a bold word," or even just to limit full-text searches to certain portions of a document. Such queries are clearly difficult to ask in a language like SQL. Another reason is that native XML databases preserve things like document order, processing instructions, and comments, and often preserve CDATA sections, entity usage, and so on, while XML-enabled databases do not.

Native XML databases are also commonly used to integrate data. While data integration has historically been performed through federated relational databases, these require all data sources to be mapped to the relational model. This is clearly unworkable for many types of data and the XML data model provides much greater flexibility. Native XML databases also handle schema changes more easily than relational databases and can handle schema less data as well. Both are important considerations when integrating data from sources not under your direct control.

The third major use case for native XML databases is semi-structured data, such as is found in the fields of finance and biology, which change so frequently that definitive schemas are often not possible.

Native XML databases do not require schemas (as do relational databases), they can handle this kind of data, although applications often require humans to process it.

The final major use of native XML database is in handling schema evolution. While native XML databases do not provide complete solutions by any means, they do provide more flexibility than relational databases.

For example, native XML databases do not require existing data to be migrated to a new schema, can handle schema changes for which there is no data migration path, and can store data even if it conforms to an unknown version of a schema.

Other uses of native XML databases include providing data and metadata caches for long-running transactions, handling large documents, handling hierarchical data, and acting as a mid-tier data cache.

3.3.1 Examples of Native databases

Product	Developer	License	DB Type
Tamino	Software AG	Commercial	Proprietary Relational through ODBC.
eXist	Wolfgang Meier	Open Source	Relational
dbXML	dbXML Group	Open Source	Proprietary
Xindice	Apache Software Foundation	Open Source	Proprietary (Model-based)

Table 3.3.1 List of Native XML Databases

3.4 COMPARISON OF RELATIONAL AND XML DATABASES

Although relational databases fit a lot of problems very well, they don't really fit XML documents, at least not in their full generality. While we can shred an XML document enough to stuff it into a relational table or just treat it as one big blob.

Traditionally, information that doesn't naturally fit into tables has been stored in a file system. However, that approach is showing its age and probably should have been abandoned years ago.

A great deal of data is now being encoded in XML, and more is being created every day. However, many people dump these XML documents into file systems without giving much thought to managing the superstructures formed by the document collections.

A native XML database is one that treats XML documents and elements as the fundamental structures rather than tables, records, and fields. Such a database enables developers to use tools and languages that more naturally fit the structure of the documents they're working with, thereby enhancing productivity. It is also widely believed that native XML databases can significantly outperform traditional relational databases for tasks that involve heavy document processing, such as newspaper publishing, Web site management, and Web services.

Relational database	XML database
A relational database contains tables.	An XML database contains collections.
A relational table contains records with the same schema.	A collection contains XML documents with the same schema.
A relational record is an unordered list of named values.	An XML document is a tree of nodes.
A SQL query returns an *unordered* set of records.	An XQuery returns an *ordered* sequence of nodes.

Table 3.4.1 Relational databases compared to XML databases

3.5 XML DATABASE ARCHITECTURE

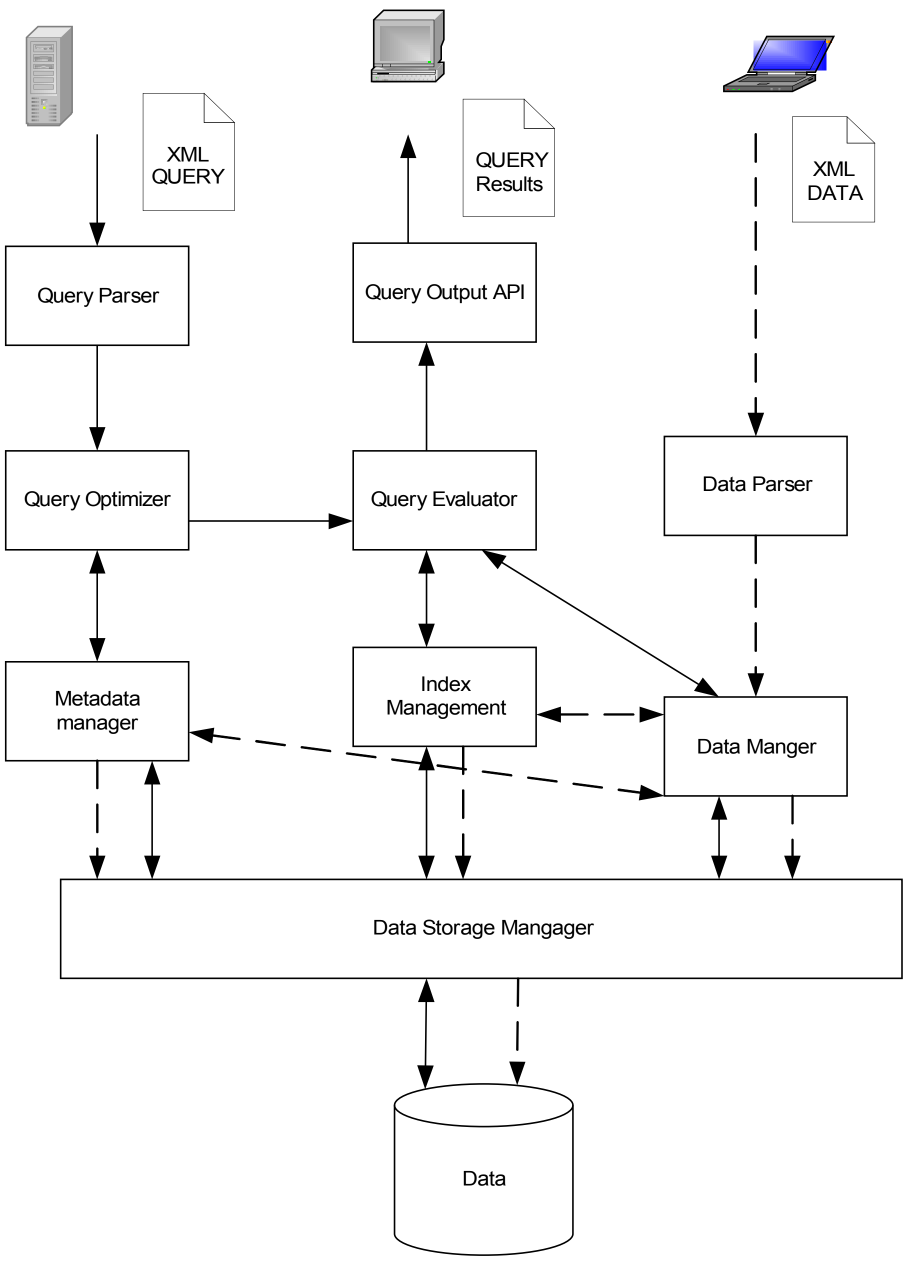


Figure 3.5.1 XML Database Architecture

CHAPTER 4

CONVERTING JDBC RESULT SETS TO XML

4.1 XML

XML has become the common format for passing data between components residing on different platforms. With the move to XML-based services, developers often find themselves converting various data structures to and from XML. The most prevalent form of persisting data is in relational databases. The problem is to convert the relational database table into XML. This project presents an API for the conversion from RDBMS to XML and vice versa.

4.2 JAVA DATABASE CONNECTIVITY

Java Database Connectivity, or JDBC, is an API for the Java programming language that defines how a client may access a database. (To be strictly correct, JDBC is not an acronym.) It provides methods for querying and updating data in a database. JDBC is oriented towards relational databases.

4.2.1 Overview

JDBC allows multiple implementations to exist and be used by the same application. The API provides a mechanism for dynamically loading the correct Java packages and registering them with the JDBC Driver Manager. The Driver Manager is used as a connection factory for creating JDBC connections.

JDBC connections support creating and executing statements. These statements may be update statements such as SQL INSERT, UPDATE and DELETE or they may be query statements using the SELECT statement. Additionally, stored procedures may be invoked through a statement.

Statements are one of the following types:

- Statement - the statement is sent to the database server each and every time.
- Prepared Statement - the statement is compiled on the database server allowing it to be executed multiple times in an efficient manner.
- Callable Statement - used for executing stored procedures on the database.

4.2.2 Drivers

4.2.2.1 Types

There are commercial and free drivers available for most relational database servers. These drivers fall into one of the following types:

- Type 1, the JDBC-ODBC bridge
- Type 2, the Native-API driver
- Type 3, the network-protocol driver
- Type 4, the native-protocol drivers
- Internal JDBC driver, driver embedded with JRE in Java-enabled SQL databases. Used for Java stored procedures.

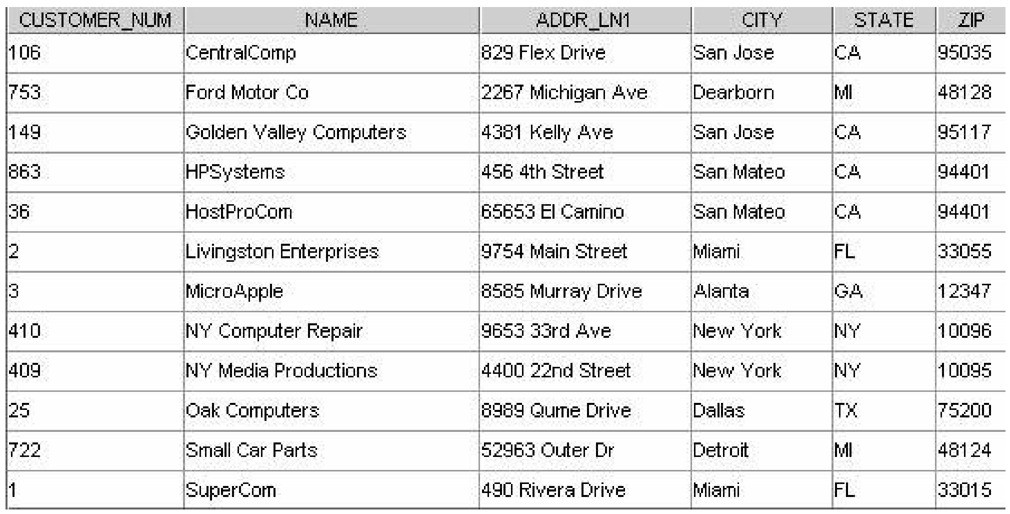
JDBC type 1 driver

The JDBC type 1 driver, also known as the JDBC-ODBC Bridge is a database driver implementation that employs the ODBC driver to connect to the database. The driver converts JDBC method calls into ODBC function calls. The bridge is usually used when there is no pure-Java driver available for a particular database.

The driver is platform-dependent as it makes use of ODBC which in turn depends on native libraries of the operating system. Also, using this driver has got other dependencies such as ODBC must be installed on the computer having the driver and the database which is being connected to must support an ODBC driver. Hence the use of this driver is not encouraged if the alternative of a pure-Java driver is available.

4.3 JDBC RESULT SET OVERVIEW

Relational data is represented in tables of rows and columns. In JDBC, the ResultSet object is used to work with tabular data. The ResultSet object has methods to iterate through rows of tabular data, and to access the columns that make up each row. Here is an example result set from the customer_tbl MS Access database:



CUSTOMER_NUM	NAME	ADDR_LN1	CITY	STATE	ZIP
106	CentralComp	829 Flex Drive	San Jose	CA	95035
753	Ford Motor Co	2267 Michigan Ave	Dearborn	MI	48128
149	Golden Valley Computers	4381 Kelly Ave	San Jose	CA	95117
863	HPSystems	456 4th Street	San Mateo	CA	94401
36	HostProCom	65653 El Camino	San Mateo	CA	94401
2	Livingston Enterprises	9754 Main Street	Miami	FL	33055
3	MicroApple	8585 Murray Drive	Alanta	GA	12347
410	NY Computer Repair	9653 33rd Ave	New York	NY	10096
409	NY Media Productions	4400 22nd Street	New York	NY	10095
25	Oak Computers	8989 Qume Drive	Dallas	TX	75200
722	Small Car Parts	52963 Outer Dr	Detroit	MI	48124
1	SuperCom	490 Rivera Drive	Miami	FL	33015

Figure 4.3.1 Rresult set from the customer_tbl

In order to build a utility that can convert *any* result set into XML, we need an abstract way of understanding what each result set contains. A ResultSetMetaData object provides just the right information. It will tell us the number of columns in a given row of data as well as the names and types of each column.

4.4 DOCUMENT OBJECT MODEL (DOM) OVERVIEW

An XML document is a tree structure of elements that contain other elements, attributes, and data. It is a very suitable data structure for storing the rows and columns of a result set.

The following diagram shows how we might model the result set displayed above into an XML document. At the top of the tree is a Results node. It contains one or more rows. Each Row contains values of the named columns in that row.

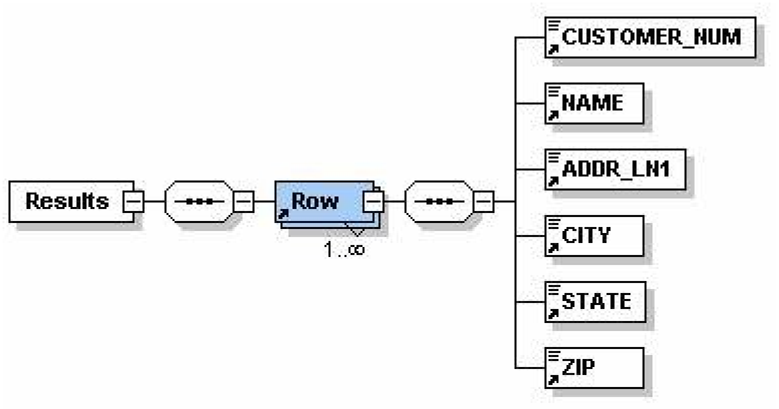


Figure 4.4.1 Resultset Representation

The DOM API provides an object representation of an XML document. It can be used to create, read, update, and delete elements. DOM provides methods for creating a new document, and then for adding elements to the document. The standard JAXP and DOM API were used.

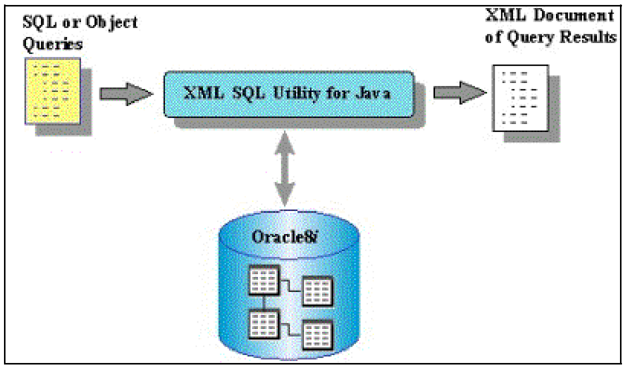


Figure 4.4.2 Table to XML Conversion

4.5 DATABASE TO XML

We found three approaches for converting Result set to XML the efficient one being the DOM tree approach which outperforms the other two methods when we consider large Result Sets, the three approaches are listed below

Three approaches of converting the Result set to XML:

1. Using Map file
2. Using the String class
3. Using DOM Tree

4.5.1 The algorithm (Map file method)

The process for creating the new XML document is as follows:

1. Parse the map file to make the necessary information, including the data to be retrieved, available.

2. Retrieve the source query. This allows for the dynamic retrieval of data based on the map file.

3. Store the data in a Document object. This temporary document will then be available to pull the data from to create the destination document according to the mapping.

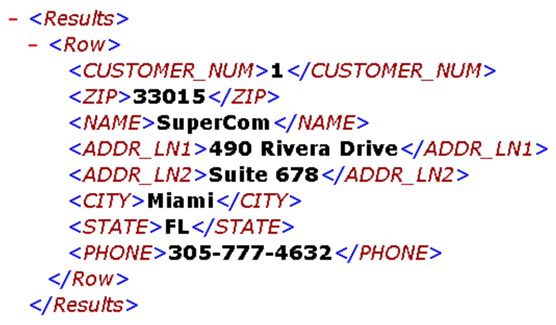
4. Retrieve the data mapping to make it available to the application.

5. Loop through the original data. Each row of the data is analyzed and re-mapped to the new structure.

6. Retrieve element mappings. The mapping file determines what data is pulled from the temporary document, and in what order.

7. Add elements to the new document. Once the data is retrieved, add it to the new document under new names.

8. Add attributes to the new document. Finally, add any attributes to the appropriate elements.



```
- <Results>
  - <Row>
      <CUSTOMER_NUM>1</CUSTOMER_NUM>
      <ZIP>33015</ZIP>
      <NAME>SuperCom</NAME>
      <ADDR_LN1>490 Rivera Drive</ADDR_LN1>
      <ADDR_LN2>Suite 678</ADDR_LN2>
      <CITY>Miami</CITY>
      <STATE>FL</STATE>
      <PHONE>305-777-4632</PHONE>
    </Row>
  </Results>
```

Figure 4.5.1 XML Document

4.6 OTHER MODULES OF THE API

1. Parsing XML and Inserting it into Database table
2. Text file to XML
3. HTML to XML using Tidy Parser
4. Constructing DOM tree (Generalized for any XML file) Using JTree of Java swing

4.7 PARSING XML AND INSERTING INTO DATABASE

Although XML files are great for storing small amounts of data, eventually there are needs to use XML with a database.

Data was entered into the database based on an external XML mapping file that determined the database, tables, column names, and element and attribute names in the XML file holding the data to be inserted into the database.

4.8 STRUCTURE OF THE MAPPING FILE

The purpose of the mapping file is to specify all aspects of the data transfer, including the structure and name of the file containing the orders and the location and structure of the database.

A mapping file can become quite complex, so certain assumptions are made for the purpose of narrow scope of this Project.

• Database tables: There are only two tables involved in this project. One hosts the overall order information, including a record for each product ordered; and one hosts the aggregate information, with overall revenue per product. In most circumstances, the aggregate information considered to be derived data would be retrieved programmatically from the first table.

• Database types: While different databases typically have different names for specific data types, they generally fall into the overall categories of text, numbers, and dates. The assumption is made that even if the actual data types differ slightly, they remain within these overall classifications.

• XML structure: Although an XML file can have any number of parent-child layers and can be organized in many ways, here an assumption is made that products will always be the first-level children of orders, even if the names differ. Data can be either attributes or elements.

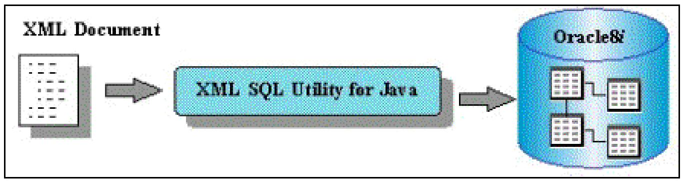


Figure 4.8.1 XML Document to Table

4.9 HTML TO XML CONVERSION

HTML Tidy is a computer program and a library whose purpose is to fix invalid HTML and give the source code a reasonable layout so that it can be converted to XML.

It was developed by Dave Raggett of W3C, and then passed on to become a Sourceforge project. Its source code is written in ANSI C for maximum portability and precompiled binaries are available for variety of platforms. It is available under the W3C license.

Examples of bad code Tidy are able to fix are:

1. Missing or mismatched end tags, mixed up tags
2. Adding missing items (some tags, quotes)
3. Reporting proprietary HTML extensions
4. Change layout owing to predefined style
5. Transform characters from some encodings into HTML entities
6. Cleaning up presentational markup

CHAPTER 5

SEARCHING

5.1 SEARCH ENGINE

A search engine is an information retrieval system designed to help find information stored on a computer system, such as on the World Wide Web, inside a corporate or proprietary network, or in a personal computer. The search engine allows one to ask for content meeting specific criteria and retrieves a list of items that match those criteria. This list is often sorted with respect to some measure of relevance of the results. Search engines use regularly updated indexes to operate quickly and efficiently.

Without further qualification, search engine usually refers to a Web search engine, which searches for information on the public Web. Other kinds of search engine are enterprise search engines, which search on intranets, personal search engines, and mobile search engines. Different selection and relevance criteria may apply in different environments, or for different uses.

Some search engines also mine data available in newsgroups, databases, or open directories. Unlike Web directories, which are maintained by human editors, search engines operate algorithmically or are a mixture of algorithmic and human input.

5.2 HOW SEARCH ENGINES WORK

A search engine operates, in the following order

1. Web crawling
2. Indexing
3. Searching

Web search engines work by storing information about a large number of web pages, which they retrieve from the WWW itself. These pages are retrieved by a Web crawler an automated Web browser which follows every link it sees. The contents of each page are then analyzed to determine how it should be indexed. Data about web pages are stored in an index database for use in later queries. Some search engines, such as Google, store all or part of the source page as well as information about the web pages, whereas others, such as AltaVista, store every word of every page they find.

This cached page always holds the actual search text since it is the one that was actually indexed, so it can be very useful when the content of the current page has been updated and the search terms are no longer in it. This problem might be considered to be a mild form of link rot, and Google's handling of it increases usability by satisfying user expectations that the search terms will be on the returned webpage.

This satisfies the principle of least astonishment since the user normally expects the search terms to be on the returned pages. Increased search relevance makes these cached pages very useful, even beyond the fact that they may contain data that may no longer be available elsewhere.

When a user comes to the search engine and makes a query, typically by giving key words, the engine looks up the index and provides a listing of best-matching web pages according to its criteria, usually with a short summary containing the document's title and sometimes parts of the text. Most search engines support the use of the Boolean terms AND, OR and NOT to further specify the search query. An advanced feature is proximity search, which allows users to define the distance between keywords.

The usefulness of a search engine depends on the relevance of the result set it gives back. While there may be millions of webpage that include a particular word or phrase, some pages may be more relevant, popular, or authoritative than others.

Most search engines employ methods to rank the results to provide the "best" results first. How a search engine decides which pages are the best matches, and what order the results should be shown in, varies widely from one engine to another. The methods also change over time as Internet usage changes and new techniques evolve.

5.3 XML AND SEARCH

Instead of searching the whole text of a page, search engines could use the XML tags to specify which parts of the pages to search, as *fields,* which should improve search results, avoid irrelevant pages and provide more precise listings of the information available.

For example, searching for "Albert Einstein" in a web wide search engine will bring up colleges named after him, university lectures mentioning him, links to sites about him, solar folklore pages, Internet philosophy in German, Jewish History, pages of quotations, biographies of his colleagues, and so on. While these are interesting, you may just have wanted to learn what books and articles he wrote. Wouldn't it be nice if you could just ask for Albert Einstein as an author, and avoid all the irrelevant pages? Even for local site and intranet searching, having fields of data would make it easier to find the most useful items.

However, adding this information will require extra effort from both site designers and search tools developers. It will only be worth an early investment for sites with very large amounts of data or intensive search needs, but the infrastructure will improve over time for the rest of us.

These are the things that must happen before XML can solve the search problems.

Older browsers won't even recognize XML pages. Internet Explorer 5 and Netscape 6 (Gecko) will render XML pages, but for the time being, most sites will need both HTML and XML versions of their pages.

Although the browsers may not display XML, you could set up a system to search XML pages based on the tags, but display the results in XML or HTML according to the browser version or user preference.

Each industry will have to set up its own standard structure, and there will be huge battles about who controls the standard.

From art to e-commerce to zoology, disciplines will have to agree on the appropriate structures for their various documents. For some groups, that's fairly easy: book reviews, for example, are fairly straightforward. Already, mathematicians, chemists, genealogists and real-estate brokers have set up standard formats for their documents.

But other areas will be much harder: for example, medical researchers will have one set of needs for disease information, while practitioners will have many other issues -- the basic information may be shared, but the context is different. Not everyone will want to use the same language to create their tags: English may be common but it's certainly not universal. Groups are now setting up Registries to store and exchange document structure information.

Web site content providers will have to tag the pages according to some standard structures. Tagging text is hard to do in the best of circumstances, because it requires writers and editors to understand the context of their data and how it fits in the structure. They will need good tools such as XML-aware editors and databases, and even then, there will be cases where the data doesn't fit into the structure very cleanly.

When data can be exported from databases and tagged automatically, the process is much simpler. However, there are millions of HTML pages, many of which include useful data, which will never be touched again.

Search engine indexing applications will have to hold the tag information as meta-data. The index must store the tags as metadata without them; the structure will be lost to searchers.

In addition to basic tags, the indexes should store the entire hierarchy, so that a searcher can specify the right tag even when the tag names are re-used in a structure.

For example, a job-listing document might have a `<location>` tag for the company headquarters, a different `<location>` tag for the job site, and yet another `<location>` tag for the headhunter's office. Obviously, these are very different and should be searched separately.

Search engines will have to deal with each standard structure in the collection of indexed documents. The search indexing program and search engine must understand the DTD or other schema of the documents in the indexed collection (this is easier on local sites and intranets than on the entire Web). Tools such as XSLT can convert from one known structure to another, but it can't be completely automatic: human judgment is required.

Search interfaces will have to display the options in a useful way, so that structural information is available to searchers. A popup menu with all the possible tags in the indexed collection is likely to be worse than useless.

Searching for fields rather than the entire document is harder to explain and requires more effort by the user. They have to adapt to the structure of the documents. The interface is more like a database search engine than a simple search field. Text search engines which search through the whole file are sometimes more flexible than database searches which must be limited to specific fields, so search interfaces should offer both options. Designing these interfaces will require usability expertise and testing.

XML, the Extensible Markup Language, will make this happen. XML is not a set of tags itself: it provides a standard system for browsers and other applications to recognize the data in a tag.

Unlike proprietary formats, XML formats are open to all. Consumers will benefit as we will have more control of our data, there will be many more useful little programs because it's easy to read the files, and we will be able to use the data even when the original applications are long gone. When applied to the Web, it makes information interchange much richer and more interesting: Tim Berners-Lee calls this the *Semantic Web*.

5.4 WORKING OF XML SEARCH ENGINE

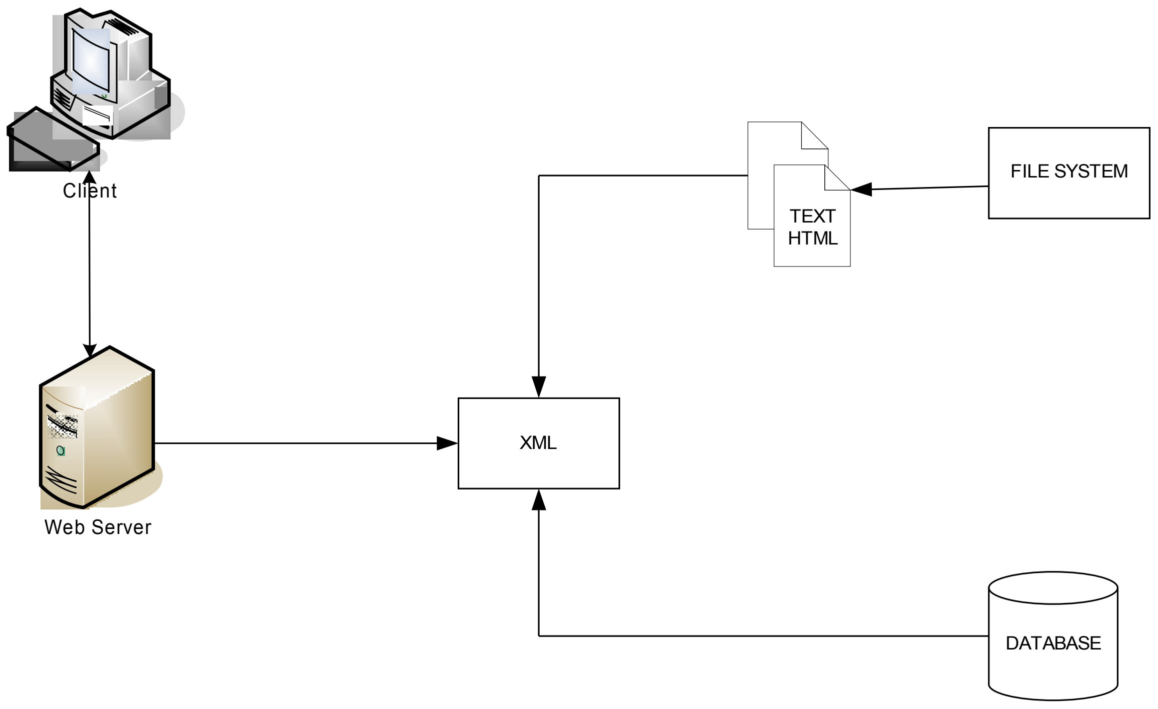


Figure 5.3.1 Working of XML Search Engine

CHAPTER 6

X SEARCH- XML SEARCH ENGINE

6.1 INTRODUCTION

It is becoming increasingly popular to publish data on the Web in the form of XML documents. Current search engines, which are an indispensable tool for finding HTML documents, have two main drawbacks when it comes to searching for XML documents. First, it is not possible to pose queries that explicitly refer to meta-data (i.e., XML tags). Hence, it is difficult, and sometimes even impossible, to formulate a search query that incorporates semantic knowledge in a clear and precise way.

The second drawback is that search engines return references (i.e., links) to documents and not to specific fragments thereof. This is problematic, since large XML documents may contain thousands of elements storing many pieces of information that are not necessarily related to each other. For example, an author is related to titles of papers she wrote, but not to titles of other papers. Actually, if a search engine simply matches the search terms against the documents, it may return documents that do not answer the user’s query.

6.2 INDEX

Search engine indexing entails how data is collected, parsed, and stored to facilitate fast and accurate retrieval. Index design incorporates interdisciplinary concepts from Linguistics, Cognitive psychology, Mathematics, Informatics, Physics, and Computer science. An alternate name for the process is Web indexing, within the context of search engines designed to find web pages on the Internet.

Popular engines focus on the full-text indexing of online, natural language documents, yet there are other searchable media types such as video, audio, and graphics. Meta search engines reuse the indices of other services and do not store a local index, whereas cache-based search engines permanently store the index along with the corpus.

Unlike full text indices, partial text services restrict the depth indexed to reduce index size. Larger services typically perform indexing at a predetermined interval due to the required time and processing costs, whereas agent-based search engines index in real time.

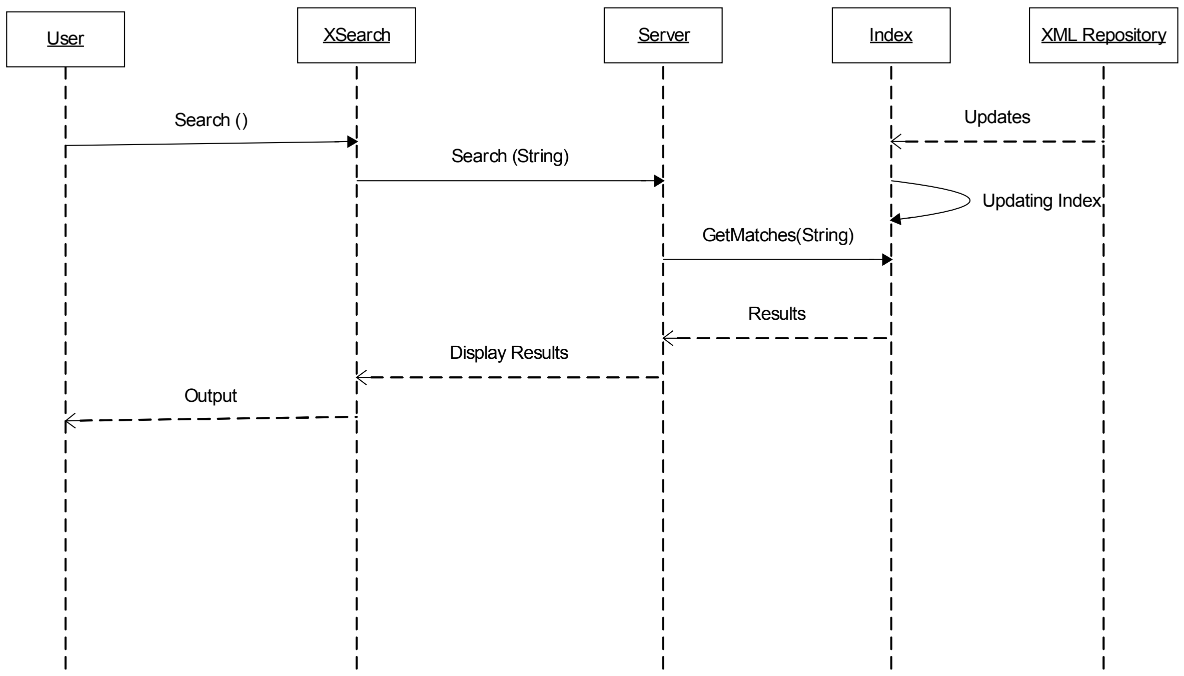


Figure 6.2.1 Sequence Diagram of X Search Working

6.3 X-SEARCH ARCHITECTURE

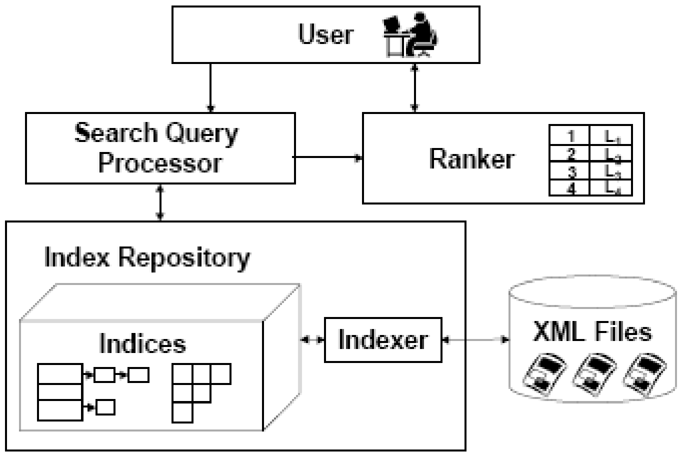


Figure6.3.1 X-Search Architecture

6.4 INDEXING

The goal of storing an index is to optimize the speed and performance of finding relevant documents for a search query. Without an index, the search engine would scan every document in the corpus, which would take a considerable amount of time and computing power.

For example, an index of 1000 documents can be queried within milliseconds, where a raw scan of 1000 documents could take hours.

No search engine user would be comfortable waiting several hours to get search results. The trade off for the time saved during retrieval is that additional storage is required to store the index and that it takes a considerable amount of time to update.

6.5 INDEX DESIGN FACTORS

Major factors in designing a search engine's architecture include:

Merge factors - how data enters the index, or how words or subject features are added to the index during corpus traversal, and whether multiple indexers can work asynchronously. The indexer must first check whether it is updating old content or adding new content. Traversal typically correlates to the data collection policy. Search engine index merging is similar in concept to the SQL Merge command and other merge algorithms.

Storage techniques - how to store the index data - whether information should be compressed or filtered

Index size - how much computer storage is required to support the index

Lookup speed - how quickly a word can be found in the inverted index. How quickly an entry in a data structure can be found, versus how quickly it can be updated or removed, is a central focus of computer science

Maintenance - maintaining the index over time

Fault tolerance - how important it is for the service to be reliable, how to deal with index corruption, whether bad data can be treated in isolation, dealing with bad hardware, partitioning schemes such as hash-based or composite partitioning and data replication

6.6 INVERTED INDICES

Many search engines incorporate an inverted index when evaluating a search query to quickly locate the documents which contain the words in a query and rank these documents by relevance.

The inverted index stores a list of the documents for each word. The search engine can retrieve the matching documents quickly using direct access to find the documents for a word. The following is a simplified illustration of the inverted index:

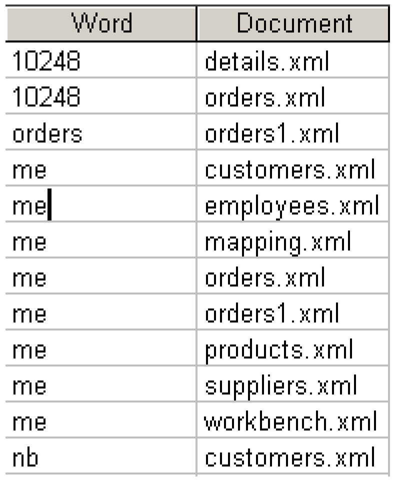
Inverted Index	
Word	Documents
the	Document 1, Document 3, Document 4, Document 5
cow	Document 2, Document 3, Document 4
says	Document 5
moo	Document 7

Table 6.6.1 Inverted Index Word Documents

The above figure is a simplified form of a Boolean index. Such an index would only serve to determine whether a document matches a query, but would not contribute to ranking matched documents. In some designs the index includes additional information such as the frequency of each word in each document or the positions of the word in each document. With position, the search algorithm can identify word proximity to support searching for phrases. Frequency can be used to help in ranking the relevance of documents to the query. Such topics are the central research focus of information retrieval.

The inverted index is a sparse matrix given that words are not present in each document. It is stored differently than a two dimensional array to reduce memory requirements. The index is similar to the term document matrices employed by latent semantic analysis.

The inverted index can be considered a form of a hash table. In some cases the index is a form of a binary tree, which requires additional storage but may reduce the lookup time. In larger indices the architecture is typically distributed.



Word	Document
10248	details.xml
10248	orders.xml
orders	orders1.xml
me	customers.xml
me	employees.xml
me	mapping.xml
me	orders.xml
me	orders1.xml
me	products.xml
me	suppliers.xml
me	workbench.xml
nb	customers.xml

Table 6.6.2 Inverted Index Table of X Search

6.7 THE FORWARD INDEX

The forward index stores a list of words for each document. The following is a simplified form of the forward index:

Forward Index	
Document	Words
Document 1	the,cow,says,moo
Document 2	the,cat,and,the,hat
Document 3	the,dish,ran,away,with,the,spoon

Table 6.7.1 Forward Index Table

The rationale behind developing a forward index is that as documents are parsed, it is better to immediately store the words per document.

The delineation enables asynchronous processing, which partially circumvents the inverted index update bottleneck. The forward index is sorting to transform it to an inverted index.

The forward index is essentially a list of pairs consisting of a document and a word, collated by the document. Converting the forward index to an inverted index is only a matter of sorting the pairs by the words. In this regard, the inverted index is a word-sorted forward index.

6.8 STRING SEARCHING ALGORITHM

String searching algorithms, sometimes called string matching algorithms, are an important class of string algorithms that try to find a place where one or several strings (also called patterns) are found within a larger string or text.

6.8.1 Knuth-Morris-Pratt algorithm

Main features

1. performs the comparisons from left to right
2. preprocessing phase in O(m) space and time complexity
3. searching phase in O(n+m) time complexity (independent from the alphabet size)
4. delay bounded by log(m)

Description

The design of the Knuth-Morris-Pratt algorithm follows a tight analysis of the Morris and Pratt algorithm. It is closely related the Morris-Pratt algorithm. It is possible to improve the length of the shifts in the Morris-Pratt algorithm which actually lead to the invention of KMP algorithm.

CHAPTER 7

CONCLUSIONS

7.1 CONCLUSION

The main contribution of this project is in laying the foundations for a semantic search engine over XML documents. X-Search returns semantically related fragments, ranked by estimated relevance. Our system is extensible, and can easily accommodate different types of relationships between nodes. Thus, Search can be seen as a general framework for semantic searching in XML documents.

7.2 FUTURE ENHANCEMENTS

Right now, most search tools index and search mainly unstructured documents, such as text, HTML, word processor and PDF files, we are calling them text search engines. As XML documents provide structured data storage, many people see a need for database-style query languages which provide complex ways to access the data.

Text search works on indexed data, so only the indexing application must recognize the fields and hierarchies and store the information as metadata for each word entry. In a structured query, the system cannot assume a structured index: the query itself must convey more about the structure of the document.

Text search applies to documents as a whole, while query languages are concerned with different kinds of data, such as a single tagged field or a record made up of several fields, and there may be several of these records in a single document, or fields from several tables joined together.

APPENDIX 1

SCREEN SHOTS

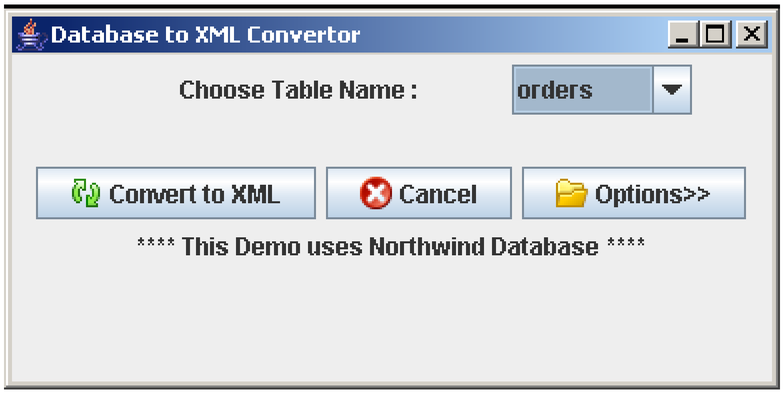

Database to XML Convertor
Choose Table Name :
orders
Convert to XML
Cancel
Options>>
**** This Demo uses Northwind Database ****

Figure A1.1 Database to XML Window

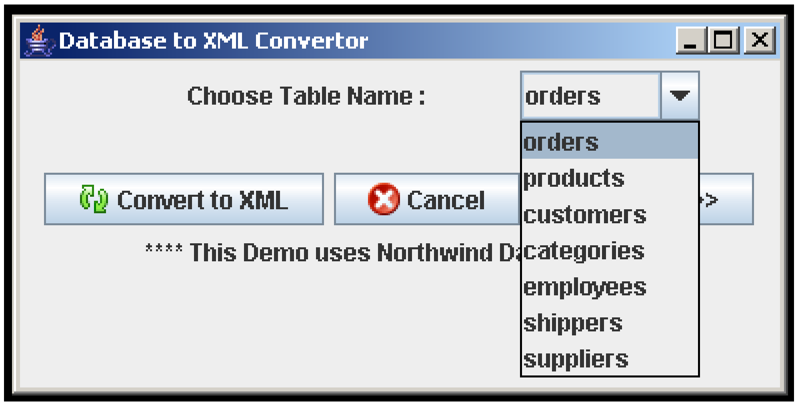
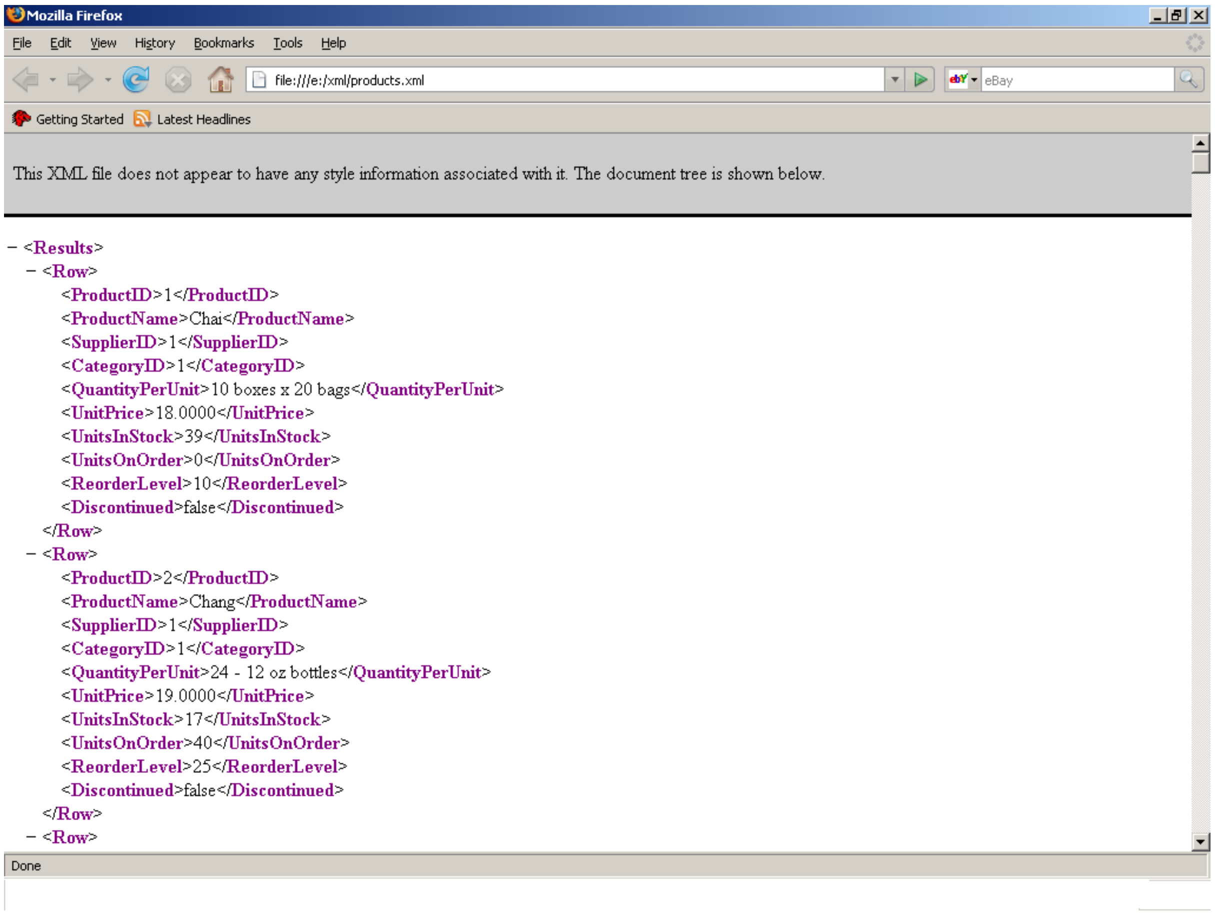


Figure A1.2 Choosing a table in Database to XML Window


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///e:/xml/products.xml
eBay
Getting Started Latest Headlines
Done

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
- <Results>
  - <Row>
      <ProductID>1</ProductID>
      <ProductName>Chai</ProductName>
      <SupplierID>1</SupplierID>
      <CategoryID>1</CategoryID>
      <QuantityPerUnit>10 boxes x 20 bags</QuantityPerUnit>
      <UnitPrice>18.0000</UnitPrice>
      <UnitsInStock>39</UnitsInStock>
      <UnitsOnOrder>0</UnitsOnOrder>
      <ReorderLevel>10</ReorderLevel>
      <Discontinued>false</Discontinued>
    </Row>
  - <Row>
      <ProductID>2</ProductID>
      <ProductName>Chang</ProductName>
      <SupplierID>1</SupplierID>
      <CategoryID>1</CategoryID>
      <QuantityPerUnit>24 - 12 oz bottles</QuantityPerUnit>
      <UnitPrice>19.0000</UnitPrice>
      <UnitsInStock>17</UnitsInStock>
      <UnitsOnOrder>40</UnitsOnOrder>
      <ReorderLevel>25</ReorderLevel>
      <Discontinued>false</Discontinued>
    </Row>
  - <Row>
```

Figure A1.3 XML Converted by the XML Converter

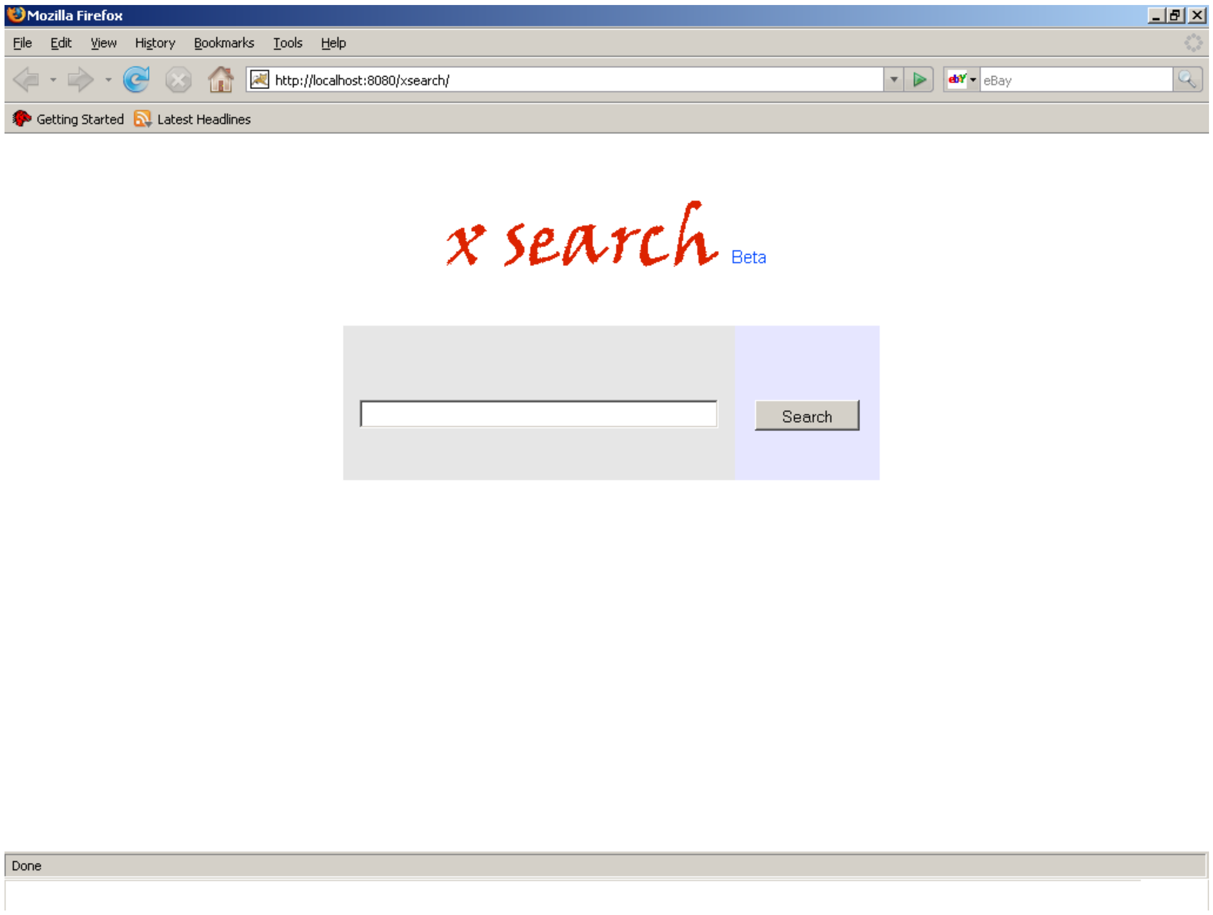
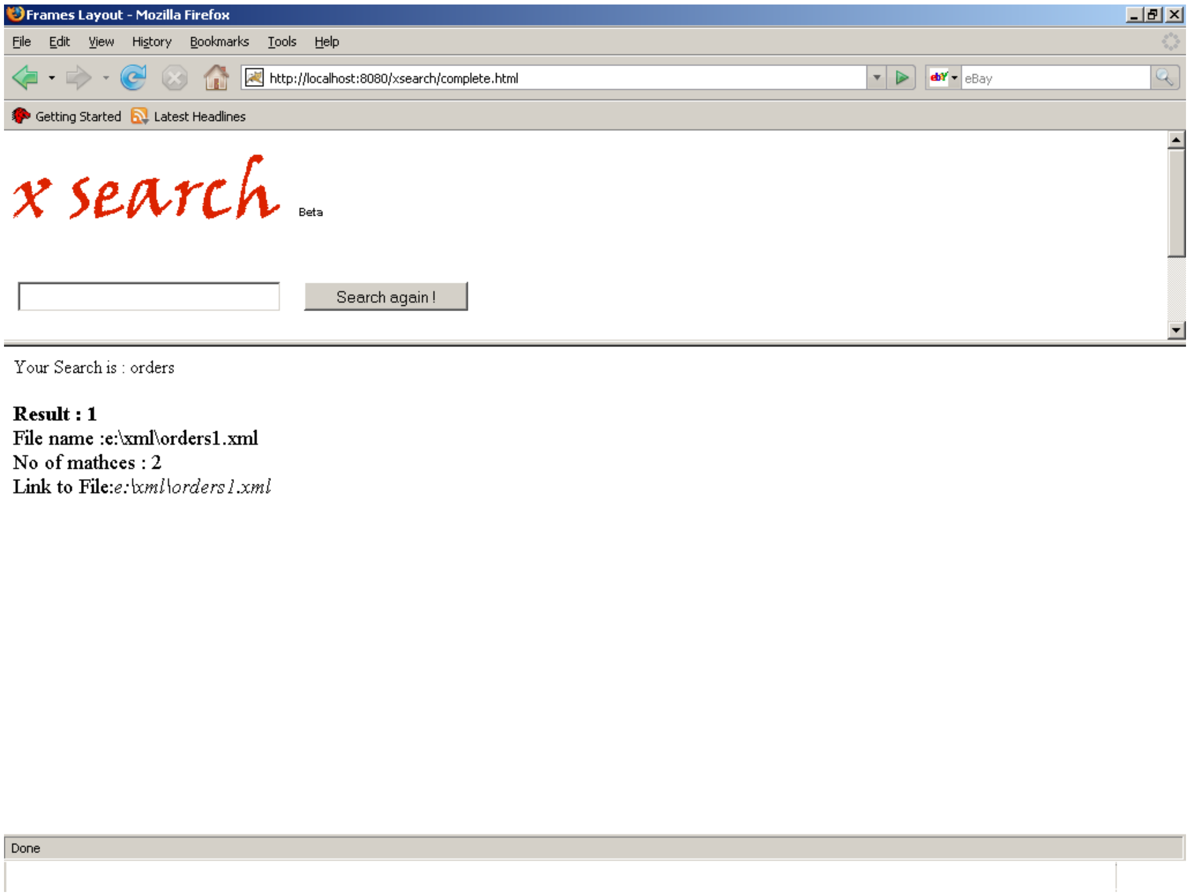

Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/xsearch/
eBay
Getting Started
Latest Headlines
x search
Beta
Search
Done

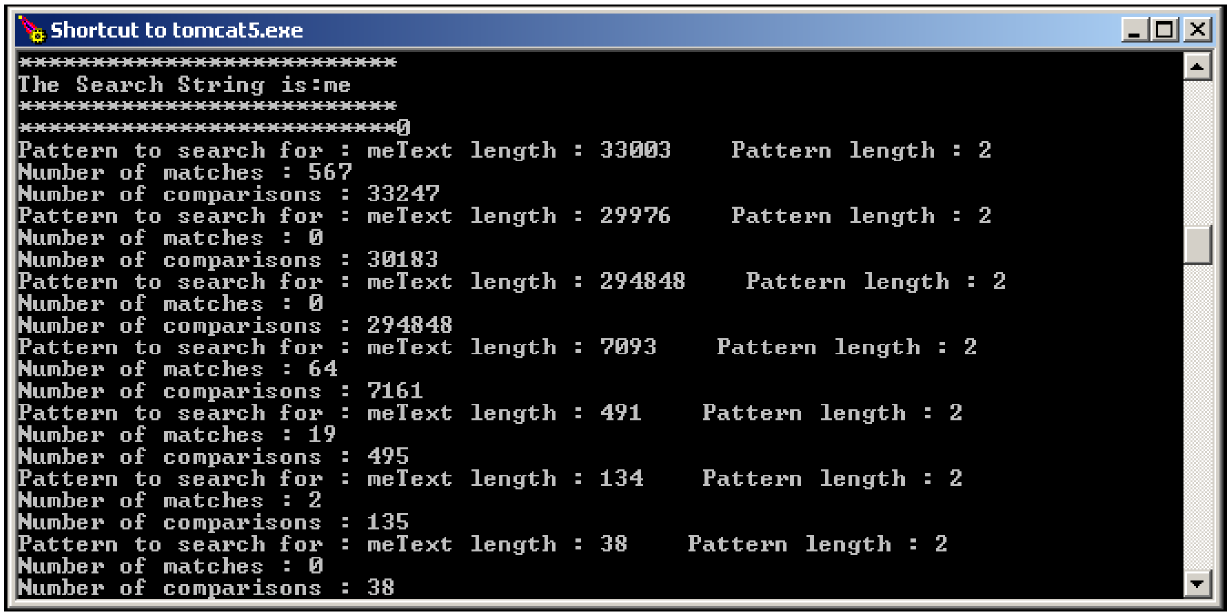
Figure A1.4 X Search Home Page


Frames Layout - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/xsearch/complete.html
eBay
Getting Started Latest Headlines
x search Beta
Search again !
Done

Your Search is : orders

Result : 1
File name :e:\xml\orders1.xml
No of mathces : 2
Link to File:*e:\xml\orders1.xml*

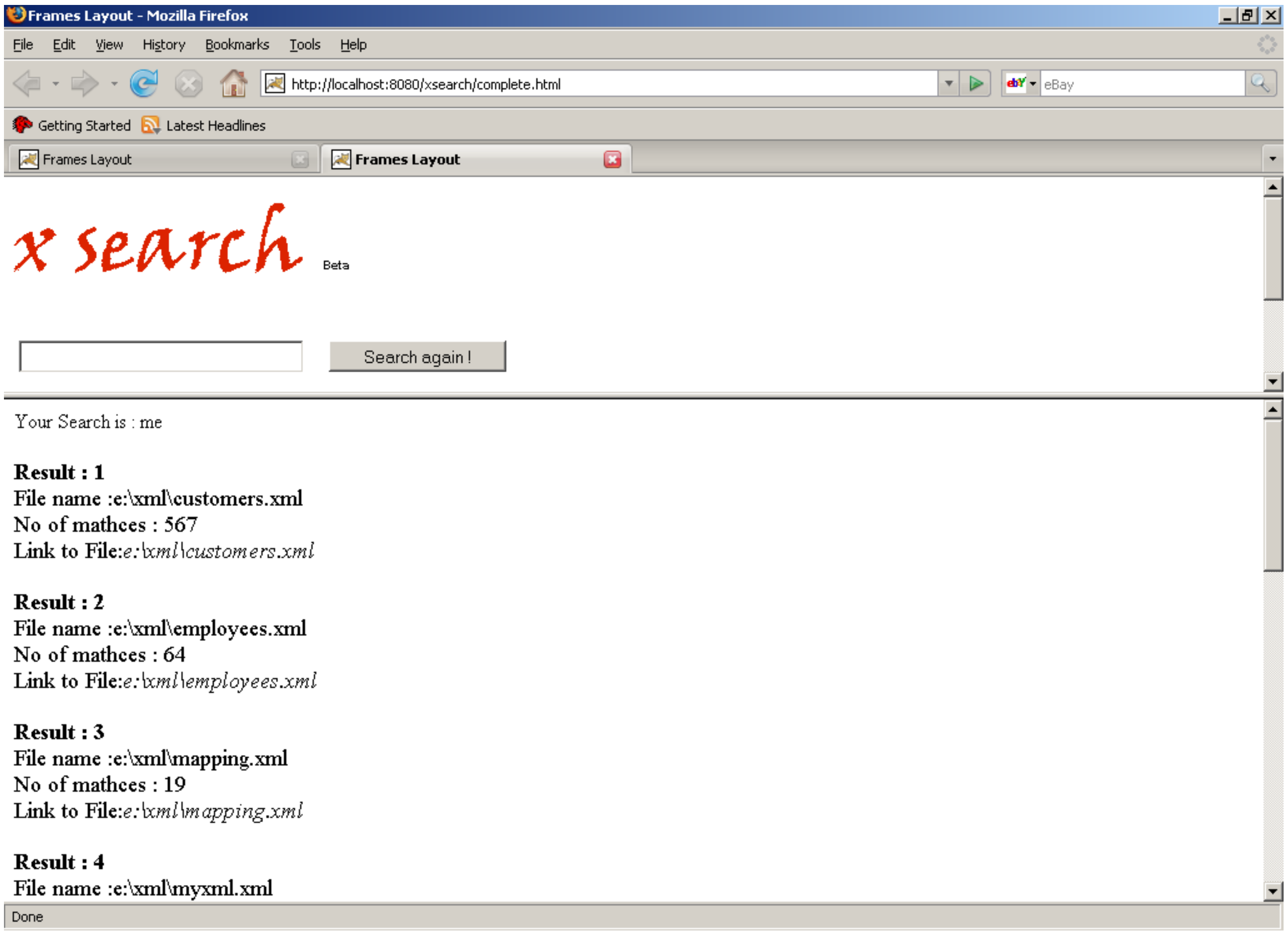
Figure A1.5 X Search Results Page



Shortcut to tomcat5.exe

```
***************************
The Search String is:me
***************************
***************************0
Pattern to search for : meText length : 33003    Pattern length : 2
Number of matches : 567
Number of comparisons : 33247
Pattern to search for : meText length : 29976    Pattern length : 2
Number of matches : 0
Number of comparisons : 30183
Pattern to search for : meText length : 294848    Pattern length : 2
Number of matches : 0
Number of comparisons : 294848
Pattern to search for : meText length : 7093    Pattern length : 2
Number of matches : 64
Number of comparisons : 7161
Pattern to search for : meText length : 491    Pattern length : 2
Number of matches : 19
Number of comparisons : 495
Pattern to search for : meText length : 134    Pattern length : 2
Number of matches : 2
Number of comparisons : 135
Pattern to search for : meText length : 38    Pattern length : 2
Number of matches : 0
Number of comparisons : 38
```

Figure A1.6 Apache Tomcat Web server Running X Search


Frames Layout - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/xsearch/complete.html
eBay
Getting Started Latest Headlines
Frames Layout
Frames Layout
x search
Beta
Search again !
Done

Your Search is : me

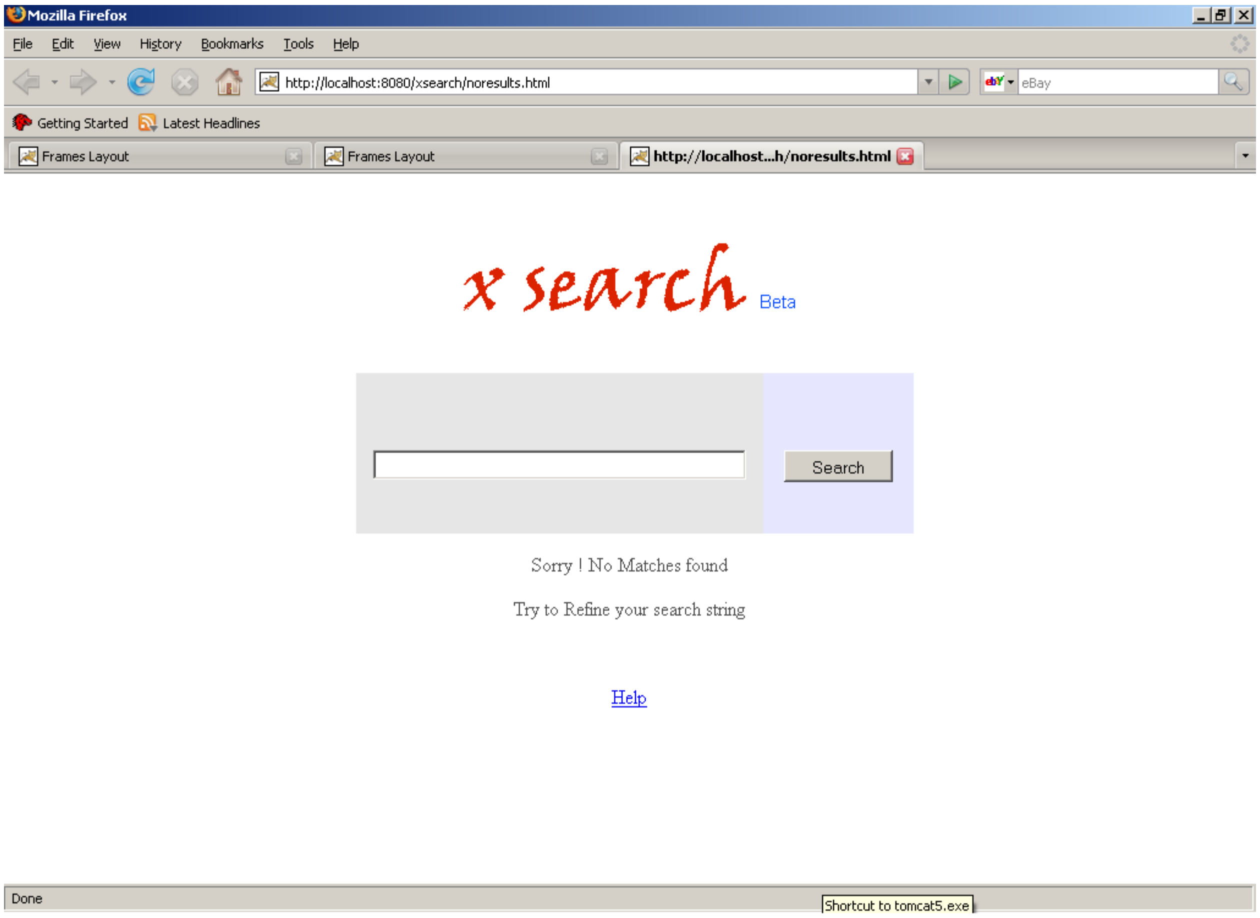
Result : 1
File name :e:\xml\customers.xml
No of mathces : 567
Link to File:*e:\xml\customers.xml*

Result : 2
File name :e:\xml\employees.xml
No of mathces : 64
Link to File:*e:\xml\employees.xml*

Result : 3
File name :e:\xml\mapping.xml
No of mathces : 19
Link to File:*e:\xml\mapping.xml*

Result : 4
File name :e:\xml\myxml.xml

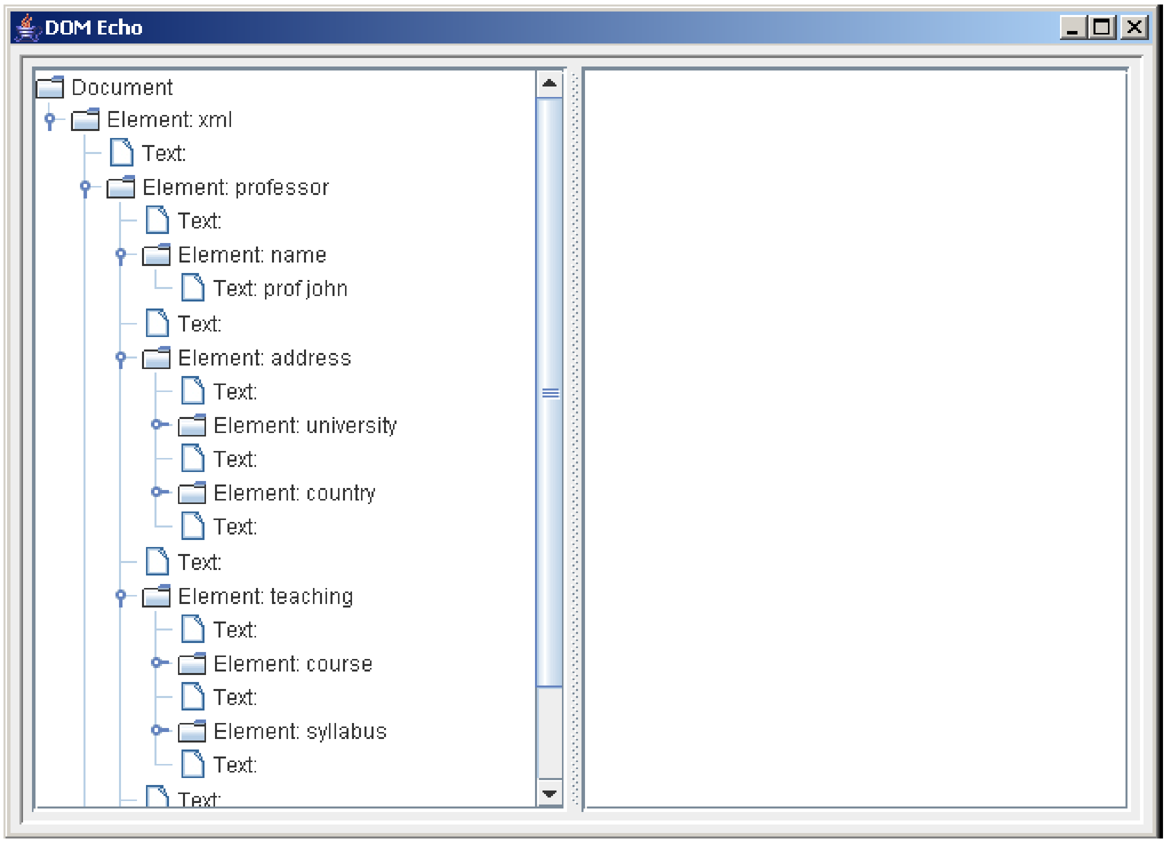
Figure A1.6 X Search Results page


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/xsearch/noresults.html
eBay
Getting Started Latest Headlines
Frames Layout
Frames Layout
http://localhost...h/noresults.html
x search Beta
Search
Help
Done
Shortcut to tomcat5.exe

Sorry ! No Matches found

Try to Refine your search string

Figure A1.7 X Search No results Found Page



```
Document
  Element: xml
    Text:
    Element: professor
      Text:
      Element: name
        Text: prof john
      Text:
      Element: address
        Text:
        Element: university
        Text:
        Element: country
        Text:
      Text:
      Element: teaching
        Text:
        Element: course
        Text:
        Element: syllabus
        Text:
      Text:
```

Figure A1.8 XML DOM Tree Displayed as a JTree

REFERENCES

1. A. Schmidt, M. Kersten, M. Windhouwer, F. Waas,Efficient relational storage and retrieval of XML documents, WebDB, 2000

2. Online free encyclopedia
http://www.en.wikipedia.org

3. Sun website
http://www.java.sun.com

4. Java programming tutorials at
http://www.java2s.com

5. SAX, Simple API for XML, http://sax.sourceforge.net

6. Software AG, Tamino XMLServer,SoftwareAG,
http://www.softwareag.com/tamino