Czech Technical University in Prague

Faculty of Electrical Engineering

Department of Computer Science and Engineering

Bachelor of science thesis

# XML Native Database Systems
# Review of Sedna, Ozone, NeoCoreXMS

Author : Nguyen Viet Cuong

Supervisor : Ing. Michal Valenta PhD.

Prague, January 2006

# Declaration

*I, Nguyen Viet Cuong, declare that this thesis, submitted in fulfillment of the requirements for the award of Bachelor of Science at Department of Computer Science and Engineering, Czech Technical University in Prague, is my wholly own work unless otherwise referenced or acknowledged. The document has not been submitted for qualifications at any academic institution. I also agree that the department can use this thesis for any non-commercial purposes in the future.*

*Nguyen Viet Cuong,*
*January 2006*

# Abstract

When XML came along seven years ago, promising to rewrite the rules of data management, vendors of relational databases took note, but they didn't panic. They'd already seen this movie a decade before, when the object database had been cast in the role of paradigm shifter. Now, with the emerging of XML technologies and native XML database management systems, vendors of relational database certainly have a lot to worry about. Is XML really the new trend of future? Does XML database really have the strength? All the answers depend on how XML native database management systems can advantage over their ex-dominating relational database systems.

The topic of this thesis is "XML Native Database Systems - A Case Study" which is mainly about the study of XML native database management systems. In this process, several XML Native Database Systems will be chosen, the reviews of their features will be performed. In order to accomplish the task, an appropriate metric which allows their mutual comparisons (storage limits, storage models, access languages, architecture of the system, functional extensibility, developer support, concurrent access etc.) will be built. Along with the reviews, the installation process will also be mentioned with the experience of the author himself. An illustrative application using these XML Native Database Systems as the core database engine to store and retrieve its data will be developed to demonstrate the advantages of XML native database processing.

# Acknowledgements

I would like to express my gratitude toward my supervisor, Ing. Michal Valenta PhD., without whose assistance and support this thesis would not have been possible. I would like to thank my parents and my brother for their love, encouragements and faith in me. Many thanks also go to all my friends and all the people that helped me throughout this project.

# Contents

# List of Figures

# Chapter 1

# Introduction

The extraordinary growth of the World Wide Web has been fueled by the ability it gives people to easily and cheaply distribute electronic documents to an international environment. As Web documents have become larger and more complex, Web content providers have begun to experience the limitations of a medium that does not provide the extensibility, structure, and data checking needed for large-scale commercial publishing. The birth of XML technology seems to solve all the problems with the functionalities beyond the current Hypertext Markup Language (HTML). Out of this, a lot of people and experts believe XML is the future of the Web but that's not all what this technology can bring, XML is also a database and this field is even more interesting and attention-drawing. XML Native Database Systems meanwhile are the key accessing to the XML database world. Almost all the currently relational database system giant vendors like Oracle and IBM or Microsoft are now supporting XML as an irresistible trend of technology.

We will take the approach to XML Native Database Systems by gaining knowledge about XML, XML Databases and XML Database management systems. With the meaning of a case study and in the logical process of understanding, a quick look at some basic knowledge and remarks about XML and XML Native Database systems will first be introduced in Chapter 2 and 3. Follow right after this, XML Database Systems Benchmarks will be introduced in Chapter 4, after that the thesis focuses on the review of performance and functionalities of three XML Native Database Systems : Sedna, Ozone, NeoCoreXMS. The analysis of three chosen XML Native Database Systems will be described with the performance of an illustrative application written in Java. This program is a client application connecting to the server using the particular API provided by the corresponding XML Native Database System (if any) and. The XML collection used in this application has been created as a TV show database, with the information about the chanel, airing date, show type, actors, description and so on. The application will perform several different XPath, Xquery and Xupdate statements to evaluate the advantages as well as the disadvantages of the three mentioned XML Native Database Systems although some of them might not support Xquery and Xupdate. The final chapter will sum up the reviews and conclusions with the pros and cons of each system as well as adding some comments of the author.

# Chapter 2

# XML and Database

## 2.1 XML and its future

We will begin our journey with a saying of Professor Elliotte Rusty Harold “When your only tool is a hammer, everything looks like a nail. When your only tool is a relational database, everything looks like a table.” Reality, however, is more complicated than that. Data often isn’t tabular and can benefit from a tool that more closely fits its natural structure. In fact, when that data is XML, the appropriate tool for managing it might well be a native XML database.

The term XML stands for “eXtensible Markup Language”, therefore XML is firstly a framework for defining markup languages. XML has all the features of a promising Markup Language that many people believe it will dominate the Web in the future:

- there is no fixed collection of markup tags - we may define our own tags, tailored for our kind of information
- each XML language is targeted at its own application domain, but the languages will share many features
- there is a common set of generic tools for processing documents

Despite many advantages, however, according to my perspective, XML is not a replacement for HTML because of the following reasons:

- XML and HTML were designed with different goals:
  - XML was designed to describe data and to focus on what data is.
  - HTML was designed to display data and to focus on how data looks.
- HTML is about displaying information, while XML is about describing information.
- HTML, or in fact, XHTML is just another Extensive Markup Language for hypertext markup

Further than that, apart from HTML and the task of displaying information, XML is designed to:

- separate syntax from semantics to provide a common framework for structuring information (browser rendering semantics is completely defined by stylesheets)
- allow flexible markup for any imaginable application domain
- support internationalization (Unicode) and platform independence
- be the future of structured information, including databases

## 2.2 XML and Database

A question might arise to many people "Is XML a Database ?". Well, even when we think of a database in its strictest sense of "a collection of data" then a XML document is still definitely a database. Look further in the sense of a data management system, XML and its surrounding technologies have many things required in a DMBS to provide: storage (XML documents), schemas (DTDs, XML Schemas, RELAX NG, and so on), query languages (XQuery, XPath, XQL, XML-QL, QUILT, etc.), programming interfaces (SAX, DOM, JDOM) and so on. However, this doesn't mean XML is perfect for a database, on the other side, there are still some disadvantages and obstacles that XML must over come to find its way to triumph: efficient storage, indexes, security, transactions and data integrity, multi-user access, triggers, queries across multiple documents, and so on. Nevertheless, many currently available XML native database systems have already found its own way to overcome and improve these lacks. XML Database technology is now still being developed, standardized and becoming one of the most important trends in database technologies.

# Chapter 3

# XML database systems

## 3.1 Native XML database

As has been defined by the XML DB consortium, the formal definition of a Native XML Database states that: A Native XML Database ...

- Defines a (logical) model for an XML document – as opposed to the data in that document – and stores and retrieves documents according to that model. At a minimum, the model must include elements, attributes, PCDATA, and document order. Examples of such models are the XPath data model, the XML Infoset, and the models implied by the DOM and the events in SAX 1.0.

- Has an XML document as its fundamental unit of (logical) storage, just as a relational database has a row in a table as its fundamental unit of (logical) storage.

- Is not required to have any particular underlying physical storage model. For example, it can be built on a relational, hierarchical, or object-oriented database, or use a proprietary storage format such as indexed, compressed files.

From this definition we can see that native XML Databases are designed to store XML data with the mechanism of putting XML documents in and taking XML documents out. The storage model for XML Databases can vary differently to support the purpose of querying and retrieving data.

In fact, NXDs are actually quite a new concept in the database arena: XML Databases have only started nearing the mainstream in the past few years. However, the more people used XML for communication and defining file-based documents, the more they found it a pain transforming this structure into the relatively flat relational database model. And so arose the requirement for a database that doesn't require any kind of transformation, it just takes the XML, and stores it in its original structure, and lets you use XML querying languages to get at the data. This is different, though, to storing it in its original format. NXDs generally don't store the data in a text file. Instead, they put it into a binary format that's suitable for random access, performance of updating and therefore data are more compact. They also maintain indices, much like indices in relational databases, except the index is on a path into the XML document rather than on a particular field in a table.

## 3.2 Current native XML database products

There are currently many native XML database systems available on the market, in both OpenSource and Commercial form. The list bellow is taken from the paper "XML Database Product", copyright 2000-2005 by Ronald Bourret at http://www.rpbourret.com/xml/XMLDatabaseProds.

| Product | Developer | License |
|---|---|---|
| ***4Suite, 4Suite Server*** | FourThought | Open Source |
| ***Berkeley DB XML*** | Sleepycat Software | Open Source |
| ***Birdstep RDM XML*** | Birdstep | Commercial |
| ***Centor Interaction Server*** | Centor Software | Commercial |
| ***DBDOM*** | K. Ari Krupnikov | Open Source |
| ***dbXML*** | dbXML Group | Open Source |
| ***DOMSafeXML*** | Ellipsis | Commercial |
| ***eXist*** | Wolfgang Meier | Open Source |
| ***eXtc*** | M/Gateway Ltd. | Commercial |
| ***Extraway*** | 3D Informatica | Commercial |
| ***GoXML DB*** | XML Global | Commercial |
| ***Infonyte DB*** | Infonyte | Commercial |
| ***Ipedo*** | Ipedo | Commercial |
| ***Lore*** | Stanford University | Research |
| ***MarkLogic Server*** | Mark Logic Corp. | Commercial |
| ***myXMLDB*** | Mladen Adamovic | Open Source |
| ***Natix*** | data ex machina | Commercial |
| ***NaX Base*** | Naxoft | Commercial |
| ***Neocore XMS*** | Xpriori | Commercial |
| ***ozone*** | ozone-db.org | Open Source |
| ***Sedna XML DBMS*** | ISP RAS MODIS | Free |
| ***Sekaiju / Yggdrasill*** | Media Fusion | Commercial |
| ***SQL/XML-IMDB*** | QuiLogic | Commercial |
| ***Sonic XML Server*** | Sonic Software | Commercial |
| ***Tamino*** | Software AG | Commercial |
| ***TeraText DBS*** | TeraText Solutions | Commercial |
| ***TEXTML Server*** | IXIASOFT, Inc. | Commercial |
| ***TigerLogic XDMS*** | Raining Data | Commercial |
| ***Timber*** | University of Michigan | Open Source |
| ***TOTAL XML*** | Cincom | Commercial |
| ***Virtuoso*** | OpenLink Software | Commercial |
| ***XDBM*** | M. Parry, P. Sokolovsky | Open Source |
| ***XDB*** | ZVON.org | Open Source |
| ***XediX TeraSolution*** | AM2 Systems | Commercial |
| ***X-Hive/DB*** | X-Hive Corporation | Commercial |
| ***Xindice*** | Apache | Open Source |
| ***XML Transactional DOM*** | Ontonet | Commercial |
| ***XpSQL*** | Makoto Yui | Open Source |

| | | |
|---|---|---|
| ***Xyleme Zone Server*** | Xyleme SA | Commercial |
| ***XStreamDB NXD*** | Bluestream Software | Commercial |

This list enumerated quite adequately all the currently available and worth-to-mention XML native database systems. In general, those Commercial ones seem to be more user-friendly and easier to manipulate but this is not always true. Some of the OpenSource XML native database systems have shown their ultimate values, qualities and advantages as well, and they are still being developed. The difference in the price would make a great deal in the decision of choosing which one for each customer's own purpose.

## 3.3 Remarks

XML native database system technologies seem to be a very interesting field. However, like most other tools, native XML databases have strengths and weaknesses:

- Good for document storage and retrieval if the information being stored is already in XML
- Retrieve documents in a readily usable form
- Most come with a high-quality full text search engine
- Can be tricky when it comes to indexing documents for search
- Don't provide good aggregation functionality

For the purpose of convenience and as a part of the thesis's topic, the author decided to focus on Sedna XML NDS, Ozone and NeoCore XMS. Sedna and Ozone are Opensource projects while NeoCore is a Commercial one, this choice would make the comparison and review more objective and general. The three XML native database systems will be later reviewed and installed, the experiences gained during the whole process will be discussed.

# Chapter 4

# XML Database System Benchmarks

## 4.1 Introduction

XML query processing has taken on considerable importance recently, and many XML databases have been constructed on a variety of platforms. There has naturally been an interest in benchmarking the performance of these systems, and a number of benchmarks have been proposed. The focus of currently proposed benchmarks is to assess the performance of a given XML database in performing a variety of representative tasks. Such benchmarks are valuable to potential users of a database system in providing an indication of the performance that the user can expect on their specific application. The challenge is to devise benchmarks that are sufficiently representative of the requirements of "most" users. The TPC series of benchmarks accomplished this, with reasonable success, for relational database systems. However, no benchmark has been successful in the realm of ORDBMS and OODBMS which have extensibility and user defined functions that lead to great heterogeneity in the nature of their use. It is too soon to say whether any of the current XML benchmarks will be successful in this respect -we certainly hope that they will.

As the purpose of this thesis is not about Benchmarking of XML native database systems, only a brief introduction about each currently available Benchmarking method will be introduced to bring out the basic idea and general technical mechanism behind XML Benchmarking.

## 4.2 The Michigan Benchmark

Being developed at Department of Electrical Engineering and Computer Science, The University of Michigan, the Michigan Benchmark propose a micro-benchmark for XML data management to aid engineers in designing improved XML processing engines. This benchmark is inherently different from application-level benchmarks, which are designed to help users choose between alternative products.

One of the interesting facts I found in this benchmark is that the developers themselves have used the benchmark to analyze the performance of three database systems: two native XML DBMS, and a commercial ORDBMS. The benchmark reveals key strengths and weaknesses of these systems. In the result, they find that commercial relational techniques are effective for XML query processing in many cases, but are sensitive to query rewriting, and require better support for efficiently determining indirect structural containment.

One aspect that current XML benchmarks do not focus on is the performance of the basic query evaluation operations such as selections, joins, and aggregations. A “micro-benchmark” that highlights the performance of these basic operations can be very helpful to a database developer in understanding and evaluating alternatives for implementing these basic operations. A number of questions related to performance may need to be answered: What are the strengths and weaknesses of specific access methods? Which areas should the developer focus attention on? What is the basis to choose between two alternative implementations? Questions of this nature are central to well-engineered systems. Application-level benchmarks, by their nature, are unable to deal with these important issues in detail. For relational systems, the Wisconsin benchmark provided the database community with an invaluable engineering tool to assess the performance of individual operators and access methods. Inspired by the simplicity and the effectiveness of the Wisconsin benchmark for measuring and understanding the performance of relational DBMSs, a comparable benchmark for XML DBMSs is developed. The benchmark that we propose to achieve this goal is called the Michigan benchmark.

A challenging issue in designing any benchmark is the choice of the benchmark’s data set. If the data is specified to represent a particular "real application", it is likely to be quite uncharacteristic for other applications with different data characteristics. Thus, holistic benchmarks can succeed only if they are able to find a real application with data characteristics that are reasonably representative for a large class of different applications.

For a micro-benchmark, the challenges are different. The benchmark data set must be complex enough to incorporate data characteristics that are likely to have an impact on the performance of query operations. However, at the same time, the benchmark data set must be simple so that it is not only easy to pose and understand queries against the data set, but also easy to pinpoint the component of the system that is performing poorly. The Michigan benchmark attempt to achieve this balance by using a data set that has a simple schema but carefully orchestrated structure. In addition, random number generators are used sparingly in generating the benchmark’s data set. The Michigan benchmark uses random generators for only two attribute values, and derives all other data parameters from these two generated values. Furthermore, as in the Wisconsin benchmark, appropriate attribute names are used to reflect the domain and distribution of the attribute values.

Finally, we note that the proposed benchmark meets the key criteria for a successful domain-specific benchmark. These key criteria are: relevant, portable, scalable, and simple. The proposed Michigan benchmark is relevant to testing the performance of XML engines because proposed queries are the core basic components of typical application-level operations of XML application. Michigan benchmark is portable because it is easy to implement the benchmark on many different systems. In fact, the data generator for this benchmark data set is freely available for download from the Michigan benchmark’s web site. It is scalable through the use of a scaling parameter. It is simple since it comprises only one data set and a set of simple queries, each designed to test a distinct functionality.

## 4.3 XMark

Databases are the preferred storage engines for many types of mission critical data. The advent of massive amounts of XML data on the World Wide Web is a challenge to current DBMS (Database Management Systems) as they are originally designed for regular data that nicely fit into tables, which is not the case for many types of XML data.

The database community increasingly devotes attention to practical management of large volumes of XML data. Prototypes of dedicated XML repositories are available and database vendors extend the functionality of their products to meet the XML processing requirements. Alongside the standardization activities of query languages, there is a strong need for a framework to analyze the capabilities and performance of such products as early as possible.

The aim of the XMark project is to provide a benchmark suite that allows users and developers to gain insights into the characteristics of their XML repositories.

The XMark benchmark consists of an application scenario that models an Internet auction site. The ER diagram of the database is shown in Figure 4.1. The attributes are omitted to simplify the diagram. The main entities are: item, person, open auction, close auction, and category. Items are the objects that are on for sale or sold already; person entities contain sub-elements like name, email address, phone number etc.; open auctions are auctions in progress; close auctions are the finished auctions; categories feature a name and a description. The XMark benchmark enriches the references in the data, like the item IDREF in an auction element and the item's ID in an item element. The text data used are the 17000 most frequently occurring words of Shakespeare's plays. The standard data size is 100MB with scaling factor 1.0 and users can change the data size by 10 times from the standard data (the initial data) each time.

## 4.4 XMach-1

XMach-1 is a benchmark for XML data management, proposed by the Database group within Department of computer science at Universitat Leipzig, Germany.

The XMach-1 benchmark is based on a web application in order to model a typical use case of a XML data management system. The system architecture consists of four parts: the XML database, application servers, loaders and browser clients (Figure 4.2). Similar to TPC-W, the System under Test (SUT) for which response time and throughput performance is determined includes the database and application servers. Including both server types is necessary since XML database processing is typically spread across the database backend and application servers. The application servers are essential for improved throughput, scalability, load balancing and caching. The number of database and application servers is not predetermined but can be chosen according to the throughput goals.

The database contains a single directory structure and a certain number of text-oriented XML documents. The directory contains metadata about all other XML documents. It represents structured data as it is schema-based and holds element content only. The XML documents are generated synthetically to support almost arbitrary numbers of documents with well-defined contents and structure. Document size and structure are variable and skewed; the average size is about 18 KB. To support scaling to different system configurations, four database sizes are possible with an initial document number of 10.000, 100.000, 1.000.000 or 10.000.000. Due to insert operations the number of documents increases during benchmark execution. Two benchmark variants are distinguished depending on whether the XML documents are schema-less or conform to schemas or DTDs. This allows us to run the benchmark with systems only supporting one of the two cases. If both variants are possible, we can evaluate the performance impact of having schema support. The workload is defined by a mix of operations from 8 query types and 3 update types. These operations

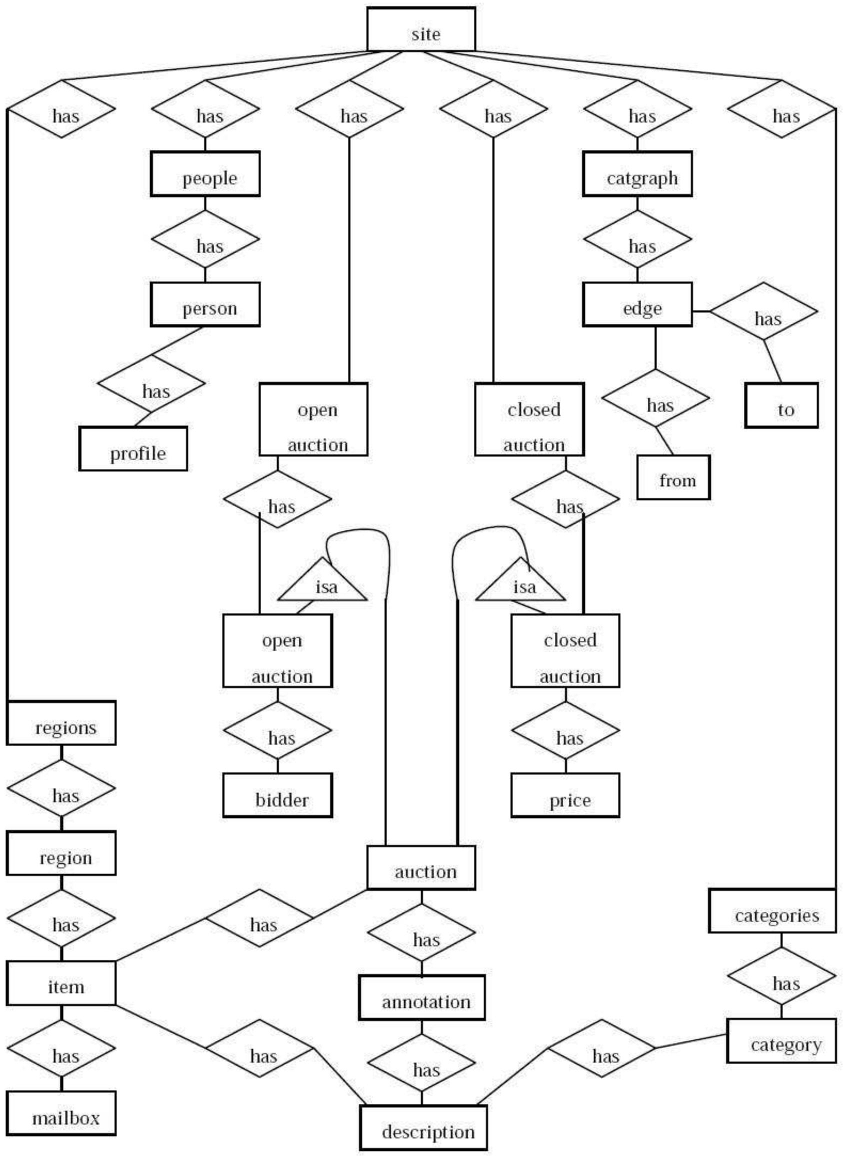


Figure 4.1: ERD of XMark Database

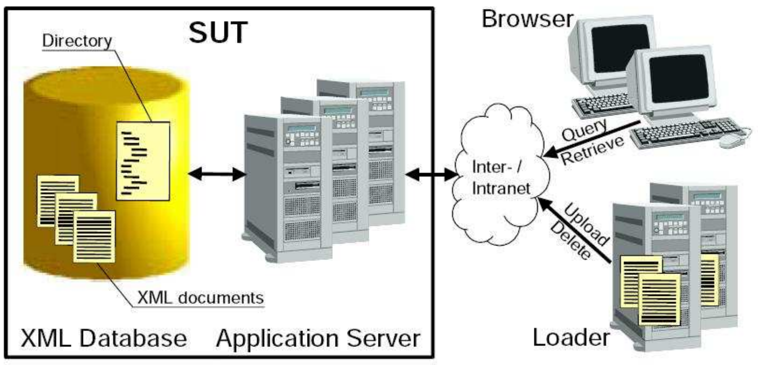


Figure 4.2: Components of benchmark architecture

cover typical database functionality (join, aggregation, sort) as well as information retrieval and XML-specific features (document assembly, navigation, element access). All operations can be expressed by XML language proposals such as Quilt or XQuery. The query and update workload is generated by emulated browsers and loaders (Fig. 4.2). The number of these clients is not predetermined but can be chosen according to the throughput goals. Interaction with the application server is via a HTTP interface.

XMach-1 can be implemented for different XML data management systems. The implementation effort depends on the supported functionality. For currently available systems performance and scalability in a multi-user environment and with data corresponding to different DTDs are unsatisfying. Sufficient performance can be achieved for XML-formatted relational data or small data sets but not for large collections with millions of XML documents.

## 4.5 XOO7

XOO7 is a research project with members from National University of Singapore, Arizona State University, USA and University of Auckland, New Zealand. The XOO7 benchmark is an XML version of the OO7 benchmark enriched with relational, document and navigational queries that are specific and critical for XML databases. The XOO7 benchmark meets the four criteria: relevance, portability, scalability and simplicity.

- XOO7 Data Set: The XOO7 data set contains both data-centric and document-centric XML data. The data structure is easy to understand. Data sets with various sizes are provided.

- XOO7 Queries: The XOO7 queries are designed to capture data-centric, document-centric and mixed characteristics of XML data.

The basic data structure in our proposed XOO7 benchmark comes from the OO7 benchmark. Figure 1 shows the conceptual schema of the database modeled using the ER diagram. This schema is translated into the corresponding DTD as shown in Figure 4.3. We note that there may be other ways to translate the ER diagram to a DTD, which are beyond the scope of this paper. Nevertheless we outline our main translation decisions.

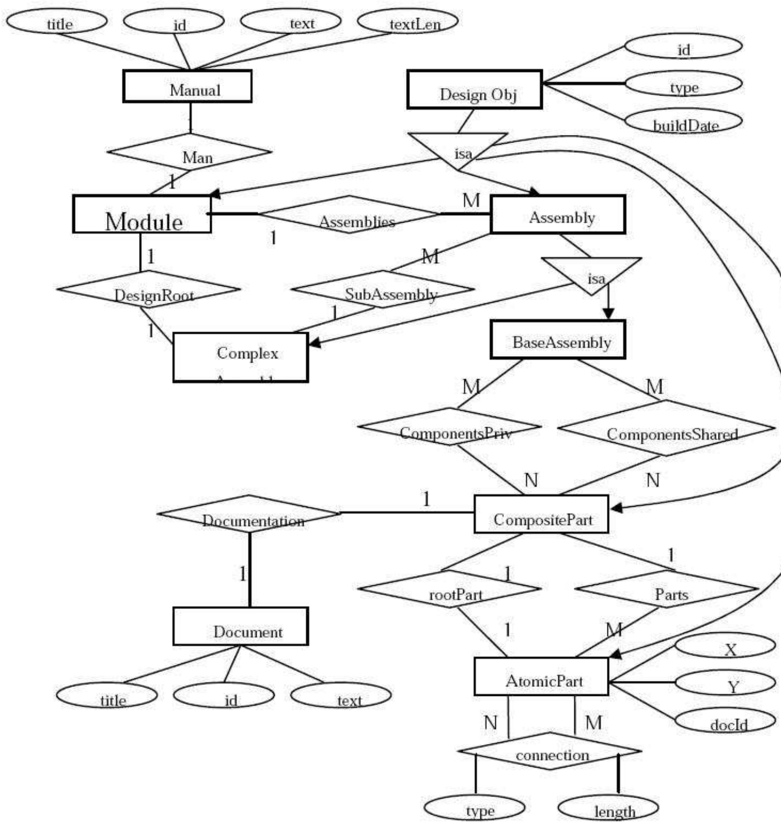


Figure 4.3: Entity relationship diagram for the OO7 benchmark

The DTD and data set of XOO7 are directly obtained by mapping the OO7 schema and data set to XML. OO7 does not model any specific application, but it intends to capture the characteristics of an object-oriented database. Additionally, in order to cater for the document centric view of XML, the document object of OO7 is extended to contain sub-elements mixed with text data. Thus, the Document element provides for a liberal use of free-form text that is "marked up" with elements. Therefore the XOO7 data set can capture all the characteristics of typical XML database applications. It provides a parameterized program to generate XML databases of various sizes and characteristics. The XOO7 benchmark also extends and modifies the eight OO7 queries with a set of twenty-three queries. XOO7 provides relational, document and navigational queries that are specific and critical for XML database applications. The queries test the primitive features and each query covers only a few features. XOO7 queries are defined to express the requirements published by the W3C XML Query Language working group. XOO7 queries are therefore supported by most XML Management Systems, which makes them very portable. Users can choose a subset of queries: data centric, document or navigational, to test on the features required in their applications.

The XOO7 benchmark has been implemented and used to test 4 XML management systems: LORE, XENA, Kweelt, and DOM-XPath. The first two systems are XML-Enabled management systems while the last two systems are Native-XML management systems.

## 4.6 Comparison

The XOO7, XMach-1, and XMark benchmarks have been designed to investigate the performance of different XMLMS. In terms of database description, XMach-1 is the most straightforward. It models a document, which is easy for users to understand. XMark simulate an auction scenario, which is a very specialized interest area, containing elements and attribute that may be difficult for users to understand. XOO7 adapts and extends the OO7 database on describing a Module, which can be understood easily.

XOO7 and XMach-1 have similar database structure. Both of them contain recursive elements, section in XMach-1 and ComplexAssembly in XOO7. However, the structure of XMach-1 is not deeply nested compared to XOO7. Although the XMark database does not contain recursive elements, it is nevertheless more complex than both XOO7 and XMach-1, containing many elements.

All the three benchmark datasets can be scaled by varying the number of elements in the XML files. In fact, XOO7 allows users to change the file size both depth-wise and breadth-wise. In addition, XMach-1 controls the database size by changing the number of XML files.

In terms of queries used by the benchmarks, the queries set in the XOO7 and XMach-1 benchmarks are straightforward and each query covers only a few functionalities. Hence, even if an XML management system does not support some features, most of queries are still executable on the system. In contrast, XMark proposes many complex queries that cover more than one functionalities. It is clear that XMach-1 and XOO7 have better portability. On the other hand, the Michigan benchmark also meets the key criteria of “relevant, portable, scalable, and simple” for a successful domain-specific benchmark.

# Chapter 5

# Sedna XML DBMS

## 5.1 Introduction

### 5.1.1 Overview

Sedna is a XML database system being developed by the MODIS team at the Institute for System Programming of the Russian Academy of Sciences. Sedna implements XQuery and its data model exploiting techniques developed specially for this language.

Sedna is designed with having two main goals:

- First, it should be the full-featured database system. That requires support for all traditional database services such as external memory management, query and update facilities, concurrency control, query optimization, etc.

- Second, it should provide a run-time environment for XML-data intensive applications. That involves tight integration of database management functionality and that of a programming language.

From the developers: "Developing Sedna, we decided not to adopt any existing database system. Instead of building a superstructure upon an existing database system, we have developed a native system from scratch. It took more time and effort but gave us more freedom in making design decisions and allowed avoiding undesirable run-time overheads resulting from interfacing with the data model of the underlying database system."

### 5.1.2 Architecture

Sedna DBMS has the following components:

- The **governor** serves as the "control center" of the system. All other components register at the governor. The governor knows which databases and transactions are running in the system and controls them.

- The **listener** creates an instance of the connection component for each client and sets up the direct connection between the client and the connection component. The connection component encapsulates the client's session. It creates an instance of the transaction component for each "begin transaction" client request.

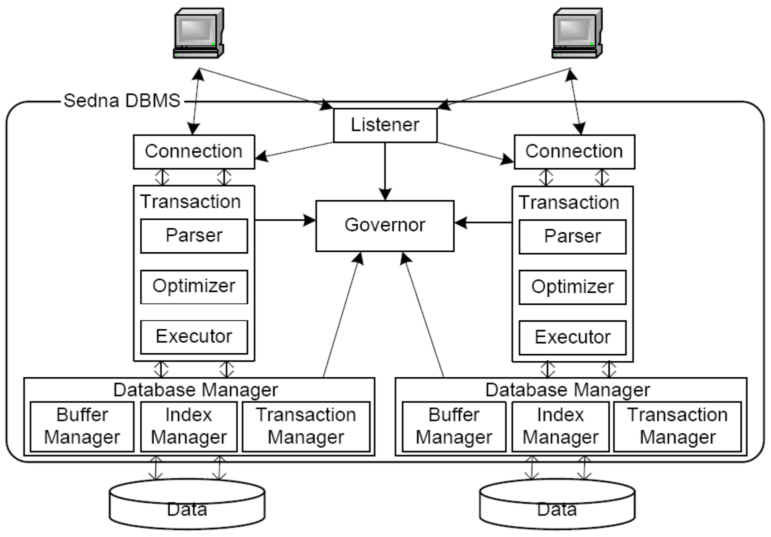


Figure 5.1: Sedna Architecture

- The **transaction component** encapsulates the following query execution components: parser, optimizer, and executor.
  - The *parser* translates the query into its logical representation, which is a tree of operations close to the XQuery core.
  - The *optimizer* takes the query logical representation and produces the optimized query execution plan which is a tree of low level operations over physical data structures.
  - The execution plan is interpreted by the *executor*. Each instance of the database manager encapsulates a single database and consists of database management services such as the index manager that keeps track of indices built on the database, the buffer manager that is responsible for the interaction between disk and main memory, and the transaction manager that provides concurrency control facilities.

### 5.1.3 Optimization

In Sedna, a wide set of rule-based query optimization techniques for XQuery is implemented. Function in-lining technique allows replacing calls to user-defined functions with their bodies. Function in-lining eliminates function call overhead and allows us to optimize in static the inlined function body together with the rest of the query. This essentially facilitates the application of the other optimization techniques. The developers have implemented an in-lining algorithm that can properly handle non-recursive functions and structurally recursive functions. The algorithm reasonably terminates innite inlining for recursive functions of any kind that makes the algorithm applicable to any XQuery query.

Predicate push down XML element constructors changes the order of operations to apply predicates before XML element constructors. It allows reducing the size of intermediate computation results to which XML element constructors are applied. This kind of transformation is of great importance because XML element constructor is an expensive operation, the evaluation of which requires deep copy of the element content. Projection of transformation statically applies XPath expressions to element constructors. It allows avoiding redundant computation of costly XML element constructors.

Query simplication using schema is useful when a query is written by the user that has vague notion about XML document schema. Making query formulation more accurate allows avoiding redundant data scanning that is peculiar to such queries. This optimization technique is based on the XQuery static type inference. Making query formulation as declarative as possible allows the optimizer to widen search space with optimal execution plans. The technique is the adaptation of analogous SQL-oriented technique that is aimed at unnesting subexpressions into joins. Join predicates normalization consists in rewriting joins predicates into conjunctive normal form to take the advantage of using dierent join algorithms, but not only nested loop join. To achieve this goal we extract subexpressions like XPath statements from join predicates and place them outside join operations, where they are evaluated only once. Identifying iterator-free subexpressions in the body of iterator-operations reduces the computation complexity of the query by putting subexpressions which do not contain free occurrences of iterator outside the body of the iterator-operation.

### 5.1.4 Concurrency control

Sedna supports multiple users concurrently accessing data. It uses a locking protocol to solve various synchronization problems, both at the logical level of objects like documents and nodes and at the physical level of pages. To ensure serializability of transactions Sedna use a well-known strict two phase locking protocol. For now, the locking granule is the whole XML document. In many cases locking of the whole XML document is not needed and this leads to the decreasing of concurrency. This is the reason why Sedna is developed with a new fine-granularity locking method, which achieves high degree of concurrency. The main idea behind our method is using numbering scheme for locking nodes and entire subtrees of XML document. The locking of entire subtree is implemented by means of locking the interval of numbering scheme labels, which includes all node labels in this subtree. This interval can be calculated by the label of the subtree root. In this approach the locking of the subtree does not lead to the locking of ancestors of the subtree root in intention mode.

## 5.2 Storage system

### 5.2.1 Data organization

Designing data organization, Sedna's makes it efficient for both queries and updates. Designing data organization in Sedna, following main decisions have been made to speed up query processing.

- First, direct pointers are used to represent relationships between nodes of an XML document such as parent, child, and sibling relationships.
- Second, a descriptive schema driven storage strategy have been developed, it consists in clustering nodes of an XML document according to their positions in the descriptive schema of the document. In contrast to prescriptive schema that is known in advance

and is usually specified in DTD and XML Schema, descriptive schema is dynamically generated (and increasingly maintained) from the data and presents a concise and accurate structure summary of these data. Formally speaking, every path of the document has exactly one path in the descriptive schema, and every path of the descriptive schema is a path of the document. As it follows from the definition, descriptive schema for XML is always a tree. Using descriptive schema instead of prescriptive one gives the following advantages:

– (1) descriptive schema is more accurate than prescriptive one.
– (2) it allows us to apply this storage strategy for XML documents which come with no prescriptive schema.

The overall principles of the data organization are illustrated in Figure 5.2. The central component is the descriptive schema that is presented as a tree of schema nodes. Each schema node is labeled with an XML node kind name (e.g. element, attribute, text, etc.) and has a pointer to data blocks where nodes corresponding to the schema node are stored. Some schema nodes depending on their node kinds are also labeled with names. Data blocks belonging to one schema node are linked via pointers into a bidirectional list. Node descriptors in a list of blocks are partly ordered according to document order. It means that every node descriptor in the block i precedes every node descriptor in the block j in document order, if and only if i < j (i.e. the block i precedes the block j in the list). Nodes are ordered between blocks in the same list in document order. Within a block nodes are unordered that reduces overheads on maintaining document order in case of updates. The text value is the content of a text node or the value of an attribute node, etc. The essence of text value is that it is of variable length. Text values are stored in blocks according to the well-known slotted-page structure method developed specifically for data of variable length. The structural part of a node reflects its relationship to other nodes (i.e. parent, children, sibling nodes) and is presented in the form of node descriptor :

The common structure of node descriptors for all node kinds is shown in Figure 5.3. The label field contains a label of numbering scheme. The meaning of the left-sibling and right-sibling pointers is straightforward. The next-in-block and prev-in-block pointers are used to link nodes within the block to allow reconstructing document order as was mentioned above. The next-in-block and prev-in-block pointers allow reconstructing document order among nodes corresponding to the same schema node, whereas the left-sibling and right-sibling pointers are used to support document order between sibling nodes.

Storing pointers to all children in the node descriptor may result in the node descriptor the size of which exceeds the size of block. To avoid this, Sedna stores only pointers to the first children by the descriptive schema. This is illustrated by the example of the library element in Figure 5.2. It has two books and one paper as child elements. In the descriptive scheme the element library schema node has only two children. In spite of the actual number of books and papers, the node descriptor for the library element has exactly two children pointers. These are pointers to the first book element and the first paper element. Having only pointers to the first children by schema allows Sedna to save up storage space and, that is more important, to make node descriptors a fixed size.

In conclusion of this section, I would like to sum up the main features of Sedna data organization that are designed to improve update operations (can be found in Sedna Documentation):

```
<library>
  <book>
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vianu</author>
  </book>
  <book>
    <title>An Introduction to Database
           Systems</title>
    <author>Date</author>
    <issue>
      <publisher>Addison-Wesley</publisher>
      <year>2004</year>
    </issue>
  </book>
  . . .
  <paper>
    <title>A Relational Model for
           Large Shared Data Banks</title>
    <author>Codd</author>
  </paper>
</library>
```

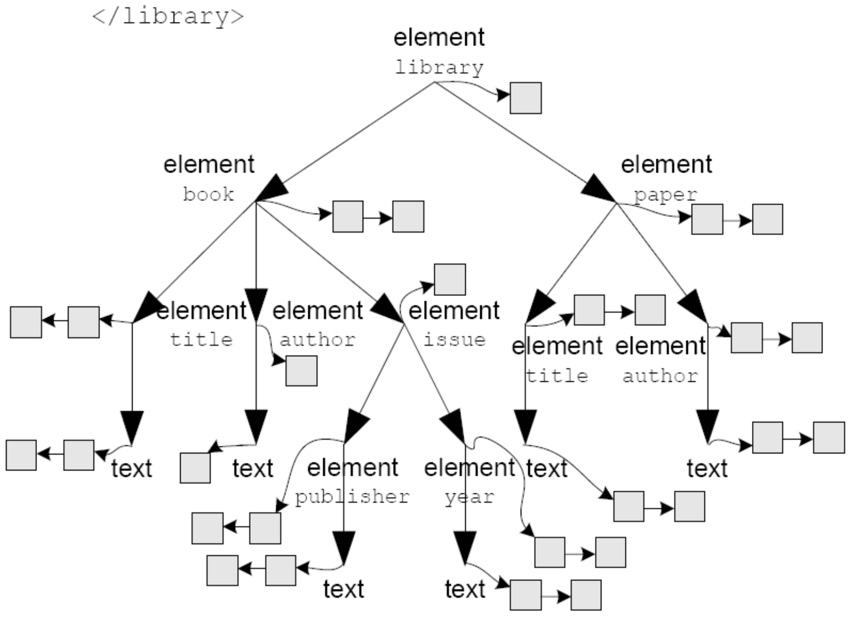


Figure 5.2: Data Organization

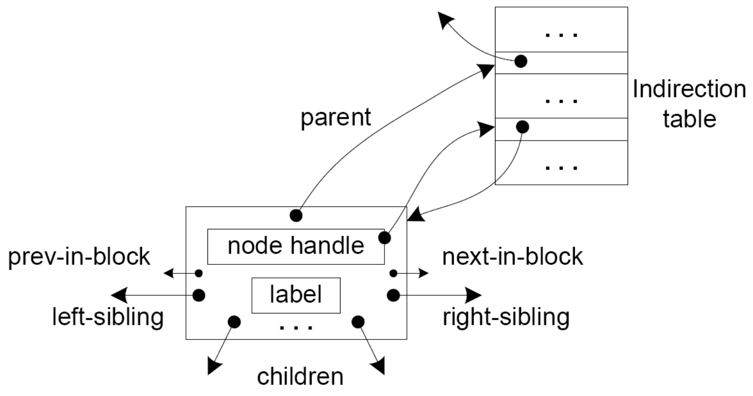


Figure 5.3: Common structure of node descriptor

- Node descriptors have a fixed size within a block
- Node descriptors are partly ordered
- The parent pointer of node descriptor is indirect

### 5.2.2 Memory management

As discussed in Section 5.2.1, one of the key design choices concerning the data organization in Sedna is to use direct pointers to present relationships between nodes. Therefore, traversing the nodes during query execution results in intensive pointer dereferencing. Making dereferencing as fast as possible is of crucial importance to achieve high performance. Although using ordinary pointers of programming languages powered by the OS virtual memory management mechanism is perfect for performance and programming effort required, this solution is inadequate for the following two reasons:

- First, it imposes restrictions on the address space size caused by the standard 32-bit architecture that still prevails nowadays.
- Second, we cannot rely on the virtual memory mechanism provided by OS in case of processing queries over large amounts of data because we need to control the page replacement (swapping) procedure to force pages to disk according to the query execution logic.

To solve these problems Sedna has been implemented with its own memory management mechanism that supports 64-bit ad- dress space (refer to it as Sedna Address Space - SAS for short) and manages page replacement. It is supported by mapping it onto the process virtual address space (PVAS for short) in order to use ordinary pointers for the mapped part. The mapping is carried out as follows:

- SAS is divided into layers of the equal size. The size of layer has been chosen so that the layer fits PVAS. The layer consists of pages (that are those in which XML data are stored as described in Section 5.2.1). All pages are of the equal size so that they can be efficiently handled by the buffer manager.

- In the header of each page there is the number of the layer the page belongs to. The address of an object in SAS (that is 64-bit long) consists of the layer number (the first 32 bits) and the address within the layer (the second 32 bits). The address within the layer is mapped to the address in PVAS on an equality basis.
- The address range of PVAS (to which the layers of SAS are mapped) is in turn mapped onto main memory by the Sedna buffer manager using memory management facilities provided by OS.

Dereferencing a pointer to an object in SAS (*layer_num, addr*) is performed as follows. *addr* is dereferenced as an ordinary pointer to an object in PVAS. This may result in a memory fault that means there is no page in main memory by this address of PVAS. In this case buffer manager reads the required page from disk. Otherwise the system checks whether the page that is currently in main memory belongs to the layer_num layer. If it is not so, the buffer manager replaces the page with the required one. It is worth mentioning that the unit of interaction with disk is not a layer but a page so that main memory may contain pages from dierent layers at the same time.

The main advantages of the memory management mechanism used in Sedna are as follows:

- Emulating 64-bit virtual address space on the standard 32-bit architecture allows removing restrictions on the size of database;
- Overheads for dereferencing are not much more than for dereferencing ordinary pointers because we map the layer to PVAS addresses on an equality basis;
- The same pointer representation in main and secondary memory is used that allows avoiding costly pointer swizzling (i.e. the process of transformation of a pointer in secondary memory to the pointer that can be used directly in main memory is called).

## 5.3 Query execution

In this section we discuss XQuery query execution over the storage system described in the previous section. The *executor* operates with the query execution plan that is a tree of physical operations which consume data from one another in a cascading fashion. Sedna's set of physical operations covers the functionality of the XQuery library and XQuery Core. It turns out that operating only in terms of the XQuery data model does not allow Sedna to support the full variety of possible query execution plans. In addition to the physical counterparts of the XQuery data model notions, tuples are used. Tuple is an ordered set of atomic values or pointers to node descriptors. Using tuples allows Sedna to extend the set of physical operations with relational-like operations such as join and group by. The set of physical operations also provides support for updates. The statement of XUpdate is represented as an execution plan which consists of two parts. The first part selects nodes that are target for the update, and the second part perform the update of the selected nodes. The selected nodes as well as intermediate result of any query expression are represented by direct pointers. The update switches to indirect pointers presented as node handles. It is necessary because the sequential updating of the selected nodes might invalidate pointers to them by performing a number of move operations. Switching to node handles fully avoids this problem.

## 5.4 Installation

### 5.4.1 Quick Installation guide

#### 5.4.1.1 Sedna Installed from RPM or DEB Packages

If you install Sedna from RPM or DEB packages on Linux, the directory structure is as follows. Below PREFIX refers to the directory where Sedna is installed.

- PREFIX/bin command line utilities for managing Sedna and databases, running queries, etc.
- PREFIX/cfg database configuration files named <db_name>_cfg.xml, where <db_name> is the name of the corresponding database
- PREFIX/data database data stored in subdirectories named <db_name>_files, where <db_name> is the name of the corresponding database
- PREFIX/lib database-independent libraries of external functions
- PREFIX/data/<db_name>_files/lib database-specific libraries of external functions
- PREFIX/doc Sedna documentation
- PREFIX/driver API drivers for various programming languages
- PREFIX/include include files
- PREFIX/etc Sedna config file sednaconf.xml (optional)
- /etc Sedna config file sednaconf.xml if it is not found in PREFIX/etc (also optional)
- PREFIX/examples example databases and applications for the Sedna APIs

To add Sedna API driver for Java we will need to point the CLASSPATH (Unix environment) to the location of *sednadriver.jar*, we could also copy *sednadriver.jar* straightly to the library location of Java Runtime Environment. This location would probably be in *...jdk.version.release/jre/lib/ext/*. After having the file copied, we now can easily use any Sedna API functions by adding *"**import** ru.ispras.sedna.driver.*;"* at the beginning of any Java code file.

#### 5.4.1.2 Sedna Installed from Source Code

If you install Sedna from source code, the directory structure after installation will still be the same as described in Section 5.4.1.1 above. All we have to do is to download the source from the developers website, using "make install", make sure you have root privilege and gcc compiler properly installed.

#### 5.4.1.3 Sedna installation under MS Windows

As Sedna does not require any particular step of setting up, installing Sedna under MS Windows is just a piece of cake, the installation file "sedna-*version*-bin-win.tar.gz" can be easily downloaded from the developer's website. Extract the downloaded file to a given directory (C:\Sedna for instant) then we are done. All the executable file will be located under Sedna\bin, API libraries can be found in Sedna\driver, databases will be stored in Sedna\data, examples can be found in Sedna\examples.

#### 5.4.1.4 Remarks

I have Sedna installed on both platforms Windows XP and Linux. Under MS Windows, we can just simply extract the installation archive to any folder, if you wish to execute it anywhere, a path can also be added. Under linux, an installation script is provided, namely *"sedna-version-bin-linux.sh"*, run this script with root privilege and the installation process will be accomplished. After installation, Sedna will be located at */usr/local/sedna* by default.

One of the interesting facts that I found when running se_sm on my laptop (Intel Centrino 1.73 Ghz, 512 MB of RAM) is that it takes about 20% of physical memory (total amount of 110MB), this is quite a great amount for a sample database. Therefore, for the sake of performance, a computer with 1 GB of RAM is recommended.

Sedna comes with a adequately comprehensive documentation. This includes the Programming guide, Administration guide, Quick start with a running example and Client-Server Protocol description.

### 5.4.2 Running Example

Running an example is a quick way to get used to and understand the organization of Sedna. The example is located in the directory:

[under Windows:] *INSTALL_DIR\examples\databases\auction* .

[under Linux:] *INSTALL_DIR/examples/databases/auction.*

To run the example, type the following commands in the command prompt:

- Change the current directory to the directory where the example is located by typing in a command line:
  - [win:] cd INSTALL_DIR\examples\databases\auction
  - [linux:] cd INSTALL_DIR/examples/databases/auction
- Start Sedna by running the following command: se_gov. If Sedna is started successfully it prints "GOVERNOR has been started in the background mode".
- Create a new database auction by running the following command: se_cdb auction. If the database is created successfully it prints "The database 'auction' has been created successfully".
- Start the auction database by running the following command: se_sm auction. If the database is started successfully it prints "SM has been started in the background mode".

- Load the sample XML document into the auction database by typing the command: se_trn auction load_data.xquery. If the document is loaded successfully it prints "UPDATE is executed successfully".
- Now you can execute the sample queries by typing the command: se_trn auction <query_name>.xquery where <query_name>.xquery is *the* name of a file with the sample query. For instance, to execute sample01.xquery you should type se_trn auction sample01.xquery It prints the query result.
- Stop Sedna by running the following command: se_stop

## 5.5 Managing databases

## 5.6 Application

I have designed a Sedna client with graphical user interface, codes are written in Java using NetBeans IDE 4.1. The screen-shot of the application is illustrated in Figure 5.4.

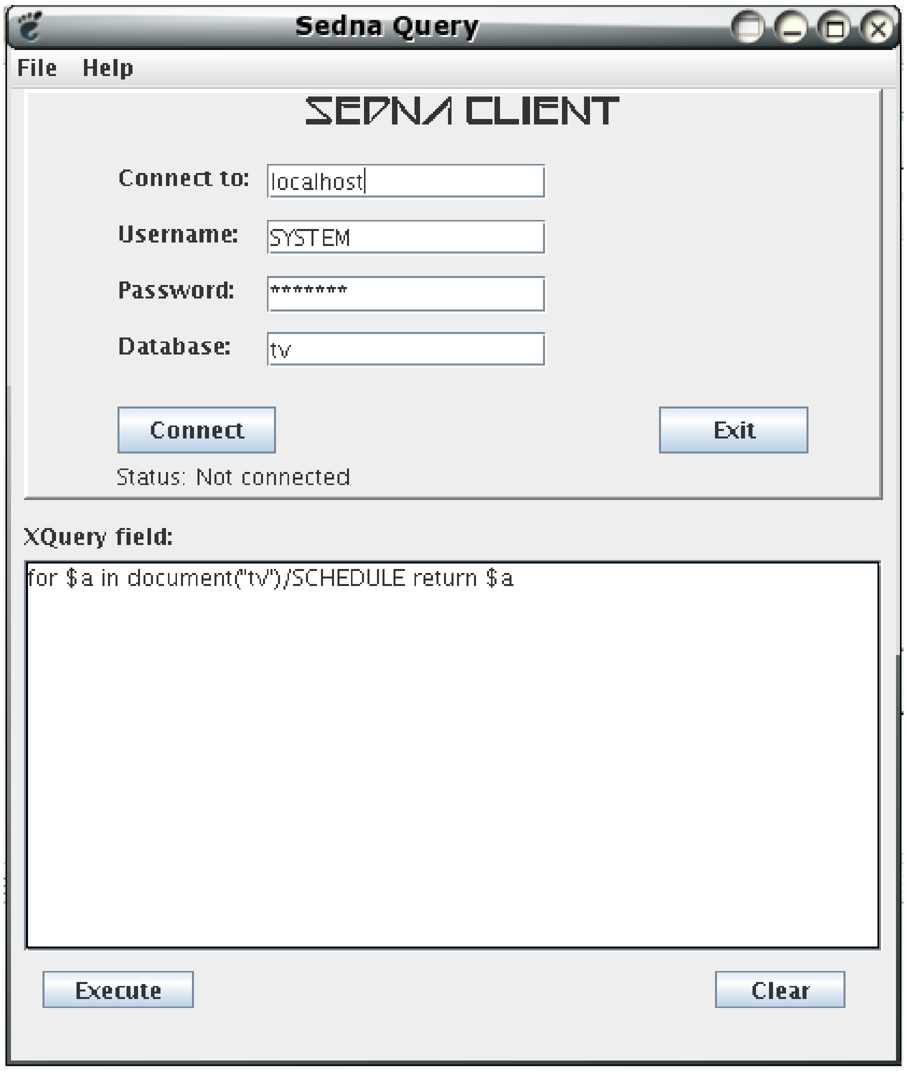
To use the provided Java API there are two possibilities:

- Copy the Java API *"sednadriver.jar"* provided in the directory *{sednapath}/driver* into library location of Java Runtime Environment. This location would most probably be in *...jdk.version.release/jre/lib/ext/*. After having the file copied, we now can easily use any Sedna API functions by adding ***"import** ru.ispras.sedna.driver.*;"* at the beginning of any Java code file.
- In NetBeans IDE 4.1, go to Project Properties, there we have the options for adding additional libraries. Click on Add Jar file and browse to the location of *"sednadriver.jar"*, press OK. After this step ***"import** ru.ispras.sedna.driver.*;"* would definitely work and now we can use any function from the Sedna API library.

My Sedna Client Application is written in Java, using Netbeans IDE 4.1. The purpose of the application is to build a client model with graphical user interface, get connected to Sedna server and perform queries on the XML database. The application consists of a connection field, where we can establish a connection to Sedna server using username, password, hostname and database name, a Query field where we can edit and execute any XQuery statement, display the result according to that.

To begin our application, we first need a running server with a database. I had Sedna running on my localhost, this can be done by executing the two commands *se_gov* and *se_sm <databasename>* (see Figure 5.5).

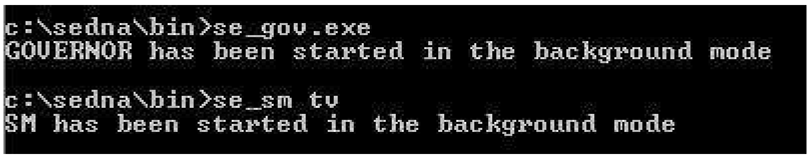
Under Linux, things work in the same way, we have to start our database before any further command can be executed. To firstly create a database, we use the command *se_cdb <database_name>.* This will take a while for Sedna to create and prepare the directory structure for the new database, after that an announcement about the successfully created database will be shown (Figure 5.6).



Connect to: localhost
Username: SYSTEM
Password: ********
Database: tv
Status: Not connected

```
for $a in document("tv")/SCHEDULE return $a
```

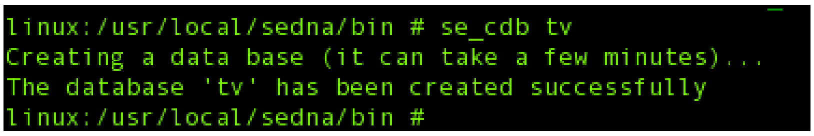
Figure 5.4: My Sedna Client under Linux



```
c:\sedna\bin>se_gov.exe
GOVERNOR has been started in the background mode

c:\sedna\bin>se_sm tv
SM has been started in the background mode
```

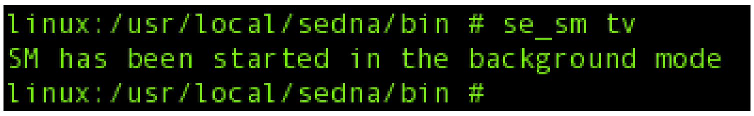
Figure 5.5: Running Sedna server under Windows



```
linux:/usr/local/sedna/bin # se_cdb tv
Creating a data base (it can take a few minutes)...
The database 'tv' has been created successfully
linux:/usr/local/sedna/bin #
```

Figure 5.6: Create a new database

To start the database, all we have to do is to run the command *"se_sm tv"* where *"tv"* is the name of the newly created database:

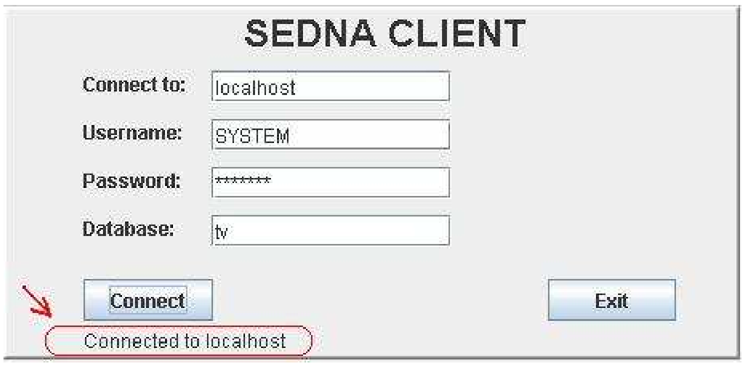


```
linux:/usr/local/sedna/bin # se_sm tv
SM has been started in the background mode
linux:/usr/local/sedna/bin #
```

Figure 5.7: Starting database under Linux

Now that we have the server running, we can start the client application, let's fill in hostname (as *localhost*), username and password, the database used is the tv database. Press Connect, if all the information provided are correct, a connection will be established and a status "Connected to localhost" will be shown (see Figure 5.6). The establishment of a connection is done by the following codes using function *getConnection* provided in the API Java Driver :

*SednaConnection con = DatabaseManager.getConnection(url, dbname, user,password);*



SEDNA CLIENT

Connect to: localhost

Username: SYSTEM

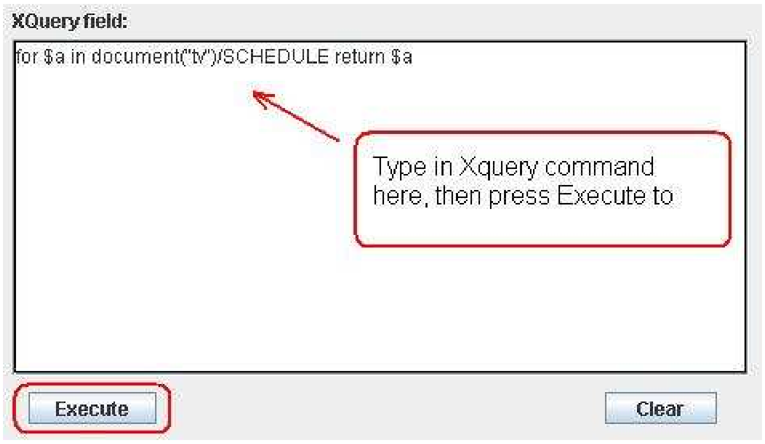
Password: *******

Database: tv

Connected to localhost

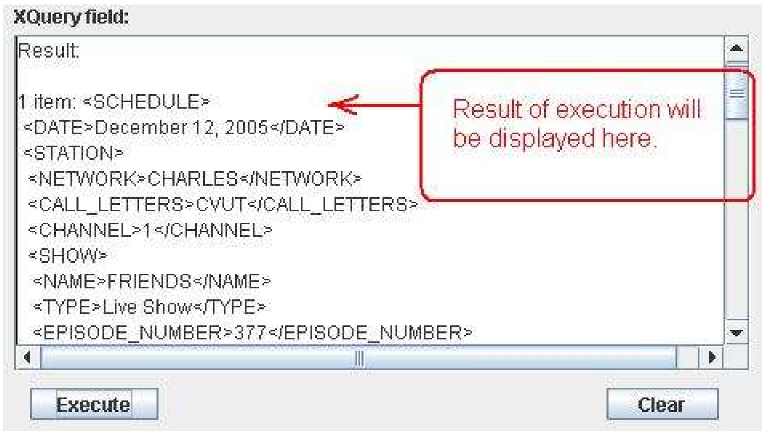
Figure 5.8: Connected to localhost

To load a document into the database, just simply use the command: *LOAD "document_name.xml" "database_name"*. In our case, we can use *LOAD "friends.xml" "tv"* to load the document "friends.xml" into the database "tv". After getting connected, we now can



```
for $a in document("tv")/SCHEDULE return $a
```

Figure 5.10: Sedna XQuery field



```
Result:

1 item: <SCHEDULE>
 <DATE>December 12, 2005</DATE>
 <STATION>
  <NETWORK>CHARLES</NETWORK>
  <CALL_LETTERS>CVUT</CALL_LETTERS>
  <CHANNEL>1</CHANNEL>
  <SHOW>
   <NAME>FRIENDS</NAME>
   <TYPE>Live Show</TYPE>
   <EPISODE_NUMBER>377</EPISODE_NUMBER>
```

Figure 5.11: Result of XQuery execution

try several Xquery statements on the database. Type in XQuery command in XQuery field and press Execute to run. If the query is correctly provided, the result we be display and the status will show "Statement succeeded" (Figure 5.9)

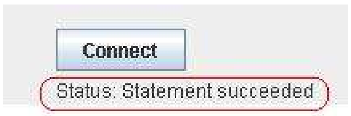
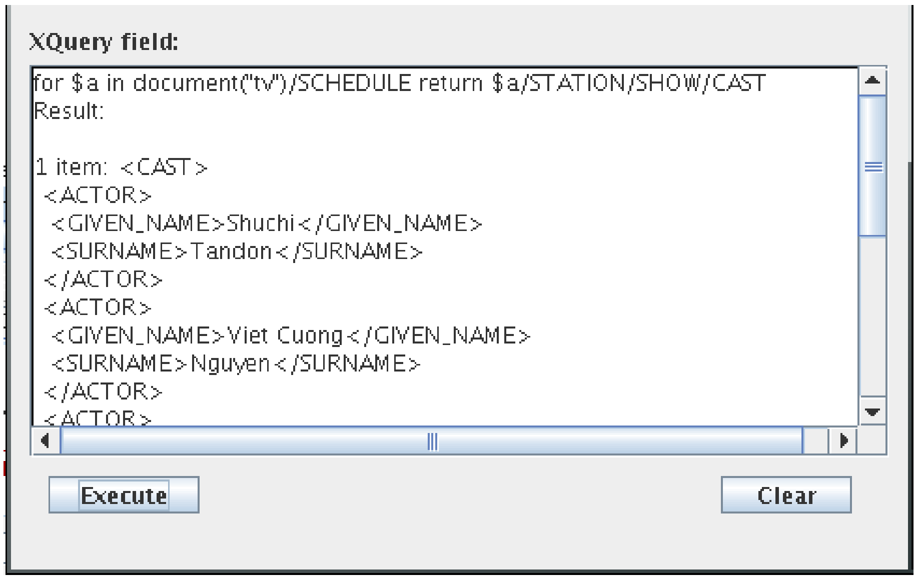


Figure 5.9: Statement succeeded status

Using the database that we created, we can see the result of the XQuery queries being displayed in XQuery field, if any error is detected, the error message will be shown here as well.

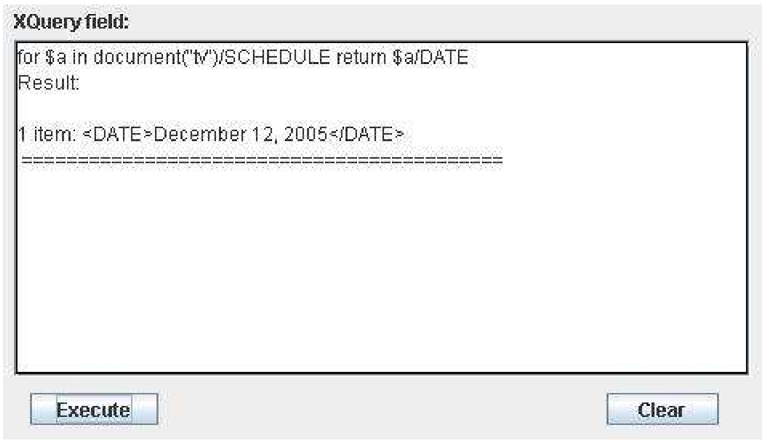
Let us now start with some simple XQuery commands, say we want to view the date of the tv program, a simple query could solve this:



```
for $a in document("tv")/SCHEDULE return $a/STATION/SHOW/CAST
Result:

1 item: <CAST>
 <ACTOR>
  <GIVEN_NAME>Shuchi</GIVEN_NAME>
  <SURNAME>Tandon</SURNAME>
 </ACTOR>
 <ACTOR>
  <GIVEN_NAME>Viet Cuong</GIVEN_NAME>
  <SURNAME>Nguyen</SURNAME>
 </ACTOR>
 <ACTOR>
```

Figure 5.13: Display CAST in the show



```
for $a in document("tv")/SCHEDULE return $a/DATE
Result:

1 item: <DATE>December 12, 2005</DATE>
========================================
```

Figure 5.12: Result of first query

In fact, the previous query can be replaced just by an XPath syntax: *document("tv")/SCHEDULE/DATE*. Now we want to print out all the casts participated in the show, the XPath syntax for this would be:

*document("tv")/SCHEDULE/STATION/SHOW/CAST*.

For some more complex XQueries, let's try to count number of actors participated in the show:

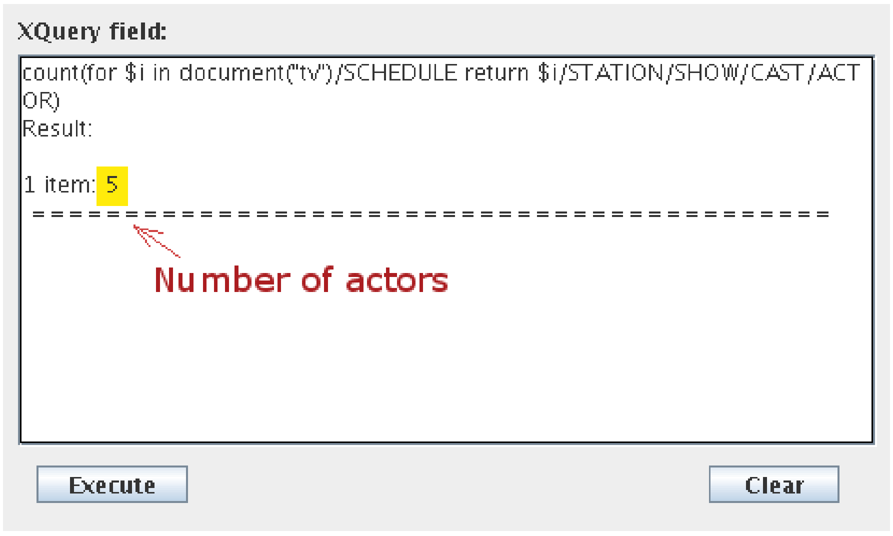
*count(for $i in document("tv")/SCHEDULE/STATION/SHOW/CAST return $i/ACTOR)*

For number of actors with surname "Nguyen" participated in the show:

*count(for $i in document("tv")/SCHEDULE/STATION/SHOW/CAST*
*where $i/ACTOR/SURNAME="Nguyen" return $i/ACTOR)*

Now let's try some XUpdate syntaxes to update some of the data in the document.

- First, let's try to insert into the database another actor with the name "John Smith" to



```
count(for $i in document("tv")/SCHEDULE return $i/STATION/SHOW/CAST/ACT
OR)
Result:

1 item: 5
 ==========================================
```

Figure 5.14: Number of actors participated in the show

the cast lists. This is done by the following query :

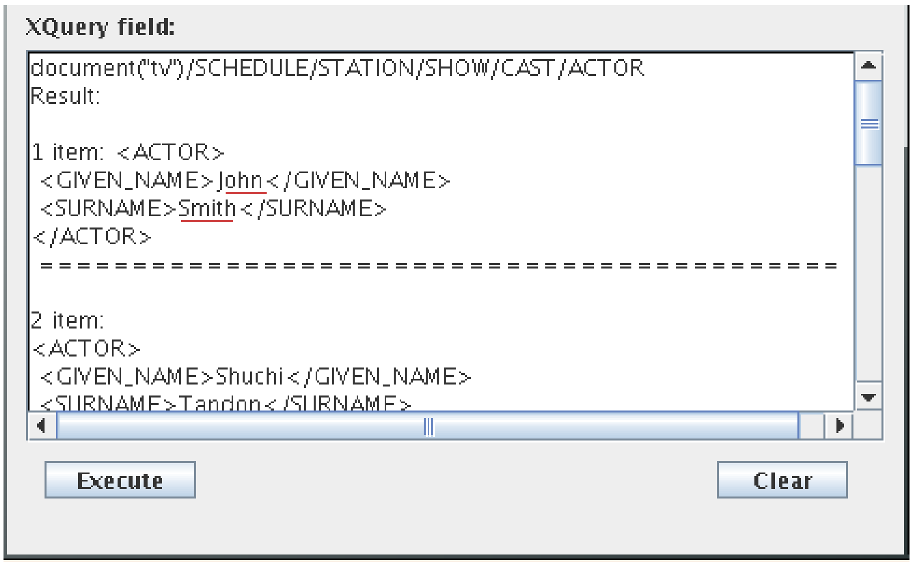
*UPDATE insert <ACTOR>*

*<GIVEN_NAME>John</GIVEN_NAME>*

*<SURNAME>Smith</SURNAME>*

*</ACTOR> into document("tv")/SCHEDULE/STATION/SHOW/CAST*

To check if the query has been executed correctly, we will display all the actors name in the cast list by executing of "*document("tv")/SCHEDULE/STATION/SHOW/CAST* / ACTOR". The result of this query is displayed showing that "John Smith" has already been added to the CAST section (Figure 5.12).



```
document("tv")/SCHEDULE/STATION/SHOW/CAST/ACTOR
Result:

1 item: <ACTOR>
 <GIVEN_NAME>John</GIVEN_NAME>
 <SURNAME>Smith</SURNAME>
</ACTOR>
 ==========================================

2 item:
<ACTOR>
 <GIVEN_NAME>Shuchi</GIVEN_NAME>
 <SURNAME>Tandon</SURNAME>
```

Figure 5.15: "John Smith" is successfully inserted

- Now, on the other hand, if we want to delete "John Smith" from CAST lists, the following query will do:

  *UPDATE delete*

  *document("tv")/SCHEDULE/STATION/SHOW/CAST/ACTOR[SURNAME/text()="Smith"]*

## 5.7 Conclusion

Sedna shows a lot of advantages as an XML native database system and without any doubts, it can be called a native full-featured data management system. Sedna is developed from scratch in C/C++ and Scheme. Some of the key features can be summarized as follow:

- Support for all traditional DBMS features (such as update and query languages, query optimization, concurrency control, various indexing techniques, recovery and security)
- Efficient support for unlimited volumes of document-centric and data-centric XML documents that may have a complex and irregular structure,
- Full support for the W3C XQuery language in such a way that the system can be efficiently used for solving problems from different domains such as XML data querying, XML data transformations and even business logic computation (in this case XQuery is regarded as a general-purpose functional programming language).
- Having a well-organized documentation and more importantly Sedna is available for free in open source form under Apache License 2.0

# Chapter 6

# Ozone XML DBMS

## 6.1 Introduction

The Ozone Database Project is a java based Object Server that makes it possible to execute persistent Java objects in a transactional environment. Ozone is an Open Source project distributed under the using the LGPL license (GNU Library General Public License). The Ozone Core however, is under the GPL license ensuring that the Ozone product itself is not used as base for a commercial closed source OODB based on Ozone.

The main goal of Ozone is to add persistence and transactions to the java object model in a transparent manner. Ozone applications should look and work like ordinary java applications. Methods on these persistent Objects should be able to be invoked on the server side making Ozone an application server, not just an object storage.

- Ozone does not introduce a new type system, object or computation model, or query language like relational DBMSs and ODMG/OQL does. Ozone just lets you work with persistent Java objects in a transactional environment. Ozone is Java! No other types than Java types, no other query/update language than Java, no impedance mismatch!
- The Ozone project aims to evolve an object server that allows developers to build pure object-oriented, Java based, persistent applications. I.e. we can create Java objects and let them run in a transactional database environment without any split between data model and object model.

Besides Java, Ozone also have an extended XML support making it an excellent XML data repository, especially when the XML manipulation we want to do can be created as Ozone objects inside the server. Ozone includes a fully W3C compliant DOM implementation that allows storing of XML data.

## 6.2 Architecture

### 6.2.1 General view

The Ozone architecture, very generally represented by the diagram further below, has four main layers:

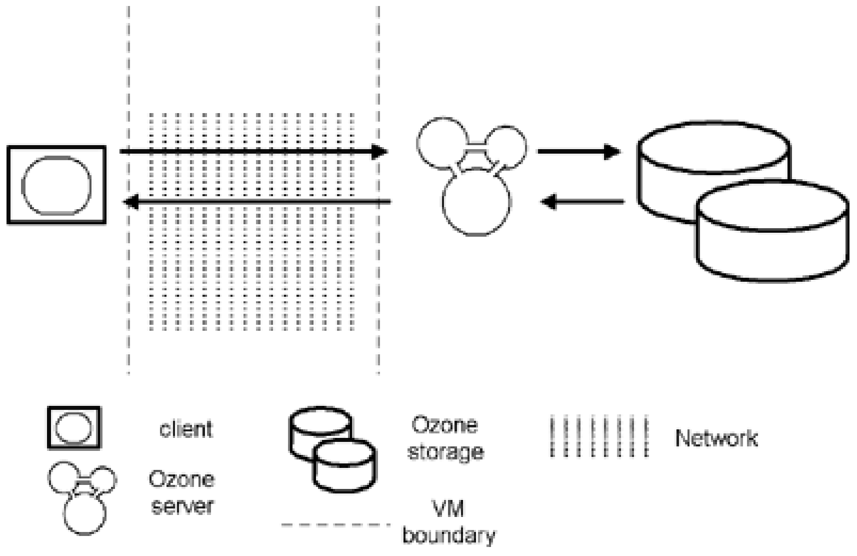


Figure 6.1: High-level view of Ozone's architecture

- Client

  This is the client application area; the client obtains a connection to an Ozone server, connection that can be shared by many threads. The client application interacts with the database API to create, delete, update and search persistent objects in the underlying Ozone storage

- Network

  The network layer is where the Ozone protocol plays a role similar to RMI. It carries method invocation information targeted at persistent objects, in addition to all other commands relayed to the Ozone server.

- Server

  The server manages client connections, security, transactions, and incoming method invocations from the clients. If required, it is in charge of invoking methods on persistent objects, therefore tightly interacting with the underlying object storage facility. The server maintains a transactionally safe environment for multiple clients that access persistent objects through a remote proxy.

- Storage

  The storage system is always accessed through an Ozone server. The storage is responsible for object persistence, clustering, object identification, and other task pertaining to low-level database-like operations.

The above architecture illustrate the most obvious use of Ozone, which acts as a remote server; but a client application can instantiate an Ozone server as part of its VM, therefore removing the network layer. Such a scheme would be convenient for standalone use. In such a case though, this behavior is implemented through a proxy pattern, which makes client code portable to both configurations (local and remote). We will see further below what this proxy scheme consists of.

### 6.2.2 Ozone Proxies

Database objects are the persistent objects designed by developers to fulfill their application logic needs. Database objects implement a given interface (in more concrete terms, a Java interface that extends org.ozoneDB.OzoneRemote), and this interface is the "visible" side of database objects. There is only one instance of a database object, which lives inside the database server. This database object is controlled via proxy objects.

A given proxy object represents its corresponding database object - inside the client applications and inside other database objects. A proxy object can be seen as a persistent reference. Proxy classes are automatically generated out of the database classes by the Ozone post-processor and implement the same public interface as their respective database object counterpart - which means that they also implement the OzoneRemote interface that their corresponding database object implements.

All ozone API methods return proxies for the actual database object inside the database. Therefore, the client deals with proxies only. However, this is transparent to the client: proxies can be used as if they were the actual database objects, since they implement the same interface.

Database objects are different from ordinary Java objects (other systems and specs, like JDO, respectively call them "primary" and "secondary", or "first-class" and "second-class"). Only one instance of a given database object reference exists in the database, as opposed to standard Java objects, which are treated in a "by-copy" fashion each time they are serialized. By analogy, database objects are a bit like rows in a relational database table, and members of these database objects that are standard Java objects correspond to the columns in the row - database object members would correspond to links to other tables, if we push the analogy.

Standard Java objects that are part of a database object can be directly used from the client application and from other database objects. In fact, database objects are sets of ordinary objects. For example the class "Car" contains several String members which are objects. One object of type Car and the dependent String members is a database object. In this regard modeling an ozone database is comparable to modeling a relational database.

Database objects in the server are loaded into memory as a whole when someone invokes a method on the object. Since the client actually uses a proxy to the server object, it gets loaded when a method on the proxy is invoked. This is a very important feature as it ensures that objects in the server are only loaded when they are needed. As an example, if the proxy you are working with contains a collection, the collection and the proxies in the collection will be available on the client but none of the corresponding objects in the server will be loaded. The object server loads objects based on client use of the proxy objects which helps reduce the load and memory usage in the server by only loading objects when needed.

### 6.2.3 Ozone storage architecture

The storage is responsible for object persistence, clustering, object identification, and other task pertaining to low-level database-like operations.

The storage API consist of a few interfaces: org.ozoneDB.core.StoreManager and org.ozoneDB.core.Objec
StoreManager manages ObjectContainers which, in turn, is just a wrapper (Decorator) around an OzoneObject (an OzoneCompatible to be exact).

There are two implementations of the Storage right now: ClassicStore and WizardStore. Both bundle several OzoneObjects into a cluster in order to decrease the overhead in serializing and deserializing objects. Clusters are thus aggregations of OzoneObjects (including the objects they reference directly). The Storage has to decide when a cluster should be considered "full" and create a new one. If the cluster is too small, there is a large overhead because for each OzoneObject, a cluster file has to be opened, unpacked, loaded, etc. If the cluster is too large, there is also a large overhead, because if only one OzoneObject is referenced, all objects in the same cluster are to be unpacked, loaded, etc. so the optimum is somewhere inbetween.

### 6.2.4 The Ozone Garbage Collector

#### 6.2.4.1 Why Garbage Collector?

An automatic memory management structure (i.e. Garbage Collection) would simplify development of systems using complex data structures (such as circular referenced structures). This is in line with the Java style of memory management and as Ozone is works as a persistent Java heap it makes sense to not force users to do manual memory management for their persistent objects when they do not have to do it for other objects.

A GarbageCollector-enabled system may run faster, because at peak uses, object deletions may be delayed until the GarbageCollector is run.Development times will be faster because users do not have to do memory management manually.

#### 6.2.4.2 Design

The persistent garbage collector is a mark-sweep-garbage collector which acts transparently in the background. Basically, it works as follows:

- There are three disjoint virtual sets of objects (possiblyReachable, surelyReachable and doneReachable). All objects initially belong to possiblyReachable. The root objects are moved to surelyReachable. Then, every surelyReachable object is processed for references it has, and these references are added to the surelyReachable set. The processed objects is moved to the doneReachable set.

- The process has ended if the surelyReachable set is empty. The objects in the doneReachable set are reachable and the objects remaining in the possiblyReachable set are not. These objects are then reclaimed. The process is done at object granularity so that there is no need to stop database operations for running the garbage collector.

The GarbageCollector considers following database objects as reachable (which will not be garbage collected):

- Named object

- Object whose proxy was sent from the server to the client during the current session and this session was not already closed

- Objects which are referenced by clients during a database session. I.e. It is an object which is on the stack of a current transaction thread

- It is an object which is reachable by a reachable object

Other objects are considered as "unreachable" and will be reclaimed once a GarbageCollector run has finished. Not only are objects reachable which have a name, also objects are reachable which are reachable by reachable objects. These "indirectly reachable" objects do not have names and still may not be reclaimed.

The GarbageCollector runs at object granularity, thus it does not block the rest of the database users. (It still disturbs because it should cause heavy IO-traffic).

#### 6.2.4.3 Referential integrity

Referential integrity is maintained. Since unreachable OzoneObjects are unreachable, they no longer need to support referential integrity. The other objects which are reachable do not loose their referential integrity property by being reachable (reachable objects are not touched). Objects, which are reclaimed, will be called via their "onDelete()" method. Thus, even the unreachable object set maintains referential integrity.

#### 6.2.4.4 Transactions

The trick to not reclaim fresh objects which are only held by the current transaction is:

- At garbageCollect start, first let all transaction finish which currently run before walking the object graph. For new transactions, new objects are marked as reachable at their creation time.
- We wait until all objects which are held by current transactions on their call stack will go reachable by other means.

#### 6.2.4.5 Remaining questions and problems

The only issue which seems to remain to impact the theoretical implementability of such a garbage-collector is the way transaction support is implemented. The reason is, that transactions, which are eligible to rollback, may recreate objects on rollback which may be handled as dead by the garbage-collector if the structure of the implementation of transaction support is not carefully observed. Then, how does it take into account that proxies (object references) can not only exist in other objects but also on the client side? Another remaining problem is with Referential Integrity: One issue we might need to investigate is the case where calling the onDelete() method actually makes that object reachable again: Is this handled by the garbage collector properly? This is a fine example for a sophisticated garbage collector test case.

## 6.3 Administration

### 6.3.1 Overview

The *'ozoneAdmin'* and *'ozoneInst'* are the commands that assist in the administration of the Ozone database. They are *'ozoneAdmin'* or *'ozoneAdmin.bat'* and *'ozoneInst'* or *'ozoneInst.bat'* depending on if the system is Windows or Unix. Just running the *'ozoneAdmin'* or *'ozoneInst'* without any arguments will display a full list of options. Note that on Windows systems any -name=value must be in double quotes. For example *'ozoneAdmin "-dburl=ozonedb:remote://localhost:3333"'*.

### 6.3.2 Build a New Database

The command *'ozoneInst -dDirectory'* will create a new empty ozone database. The database will be created in the specified directory.

### 6.3.3 Starting up and shutting down Ozone

To startup ozone use the command *'ozone -dDirectory -uUser'*. This will display information about the running ozone database to the screen. Typing the letter 'q' followed by Enter will stop the database. The specified User will be created as a user if it has not already been created.

The command *'ozoneAdmin shutdown -dburl=URL'* will shutdown a remote database. Note that on Windows systems the -dburl=URL must be in double quotes. The default if -dburl is not specified is '-dburl=ozonedb:remote://localhost:3333'.

### 6.3.4 Backing up an Ozone Database

When the database was first created, using the *'ozoneInst -dDirectory'* command, a set of directories and files were created in specified directory. This represents all of the 'persistents' of any Ozone database. Making a copy of these files, while the database is not running, can be used to backup the database.

Be sure to backup any .java files that describe the classes that are within the Ozone database. If you get a mismatch between the .java files that make up the Ozone database and the Ozone database files, you might not be able to access the data.

To restore the database, just copy the directories and files back to their original place.

There is also the *'ozoneAdmin backup > filename'* command that will backup a running Ozone database. The resulting file is an XML representation of the entire Ozone database. Again, remember to backup the .java files that make up the classes when doing a backup of the database.

To restore, use the 'ozoneAdmin restore < filename' command.

### 6.3.5 Creating Users

*'ozoneAdmin newuser -name=USERNAME -id=ID'* is the command that will add the new user USERNAME with the specified ID. The ID must be a number >= 100 since zero to 99 are reserved by Ozone. Initially, the newly created user will not exist in any groups. It is not a requirement that the user be part of any group.

### 6.3.6 Removing Users

*'ozoneAdmin removeuser -name=USERNAME'* will remove the specified USERNAME.

### 6.3.7 Listing All Users

*'ozoneAdmin allusers'* will list all users that have been created.

### 6.3.8 Creating Groups

*'ozoneAdmin newgroup -name=GROUPNAME -id=ID'* will create a new group with the specified GROUPNAME and ID. Note that the ID must be >= 100 since zero to 99 are reserved by Ozone. When a group is created it will not contain any users initially.

### 6.3.9 Removing Groups

*'ozoneAdmin removegroup -name=GROUPNAME'* will remove the specified GROUPNAME.

### 6.3.10 Adding Users to Groups

*'ozoneAdmin user2group -username=USERNAME -groupname=GROUPNAME'* will add the specified USERNAME to the specified GROUPNAME.

### 6.3.11 Listing All Groups

*'ozoneAdmin allgroups'* will list all groups that have been created as well as showing, by id, which users are in each group.

### 6.3.12 User Control Notes

When the database is created, there will be an administrator user in the admin group. When the database is started with the -uUSER, then the specified USER will be created if it does not already exist.

By utilizing the org.ozoneDB.ExternalDatabase.openDatabase(URL) without the username or password specified will give access to the database under the administrator user and admin group.

The org.ozoneDB.ExternalDatabase.openDatabase(URL,user,password) must be used if the objects need to be segregated by user or group.

The password parameter in the openDatabase method does not seem to do anything. Any password can be used as long as the user name matches.

The access parameter in the ExternalDatabase.createObject method can be used to specify read or update access to the object created. The org.ozoneDB.OzoneInterface class has the access bits. These bits can be combined (or them together) to give additional access.

#### 6.3.12.1 Access Bits

- *org.ozoneDB.OzoneInterface.Private* Will specify access that allows read and update access only to the user who created the object.
- *org.ozoneDB.OzoneInterface.AllRead* Will specify access that allows read access to all users. Only the user who created the object can update it.
- *org.ozoneDB.OzoneInterface.AllLock* Will specify access that allows update access to all users. Only the user who created the object can read it.

- *org.ozoneDB.OzoneInterface.GroupRead* Will specify access that allows read access to all users within the group of the user who created the object. Only the user who created the object can update it.
- *org.ozoneDB.OzoneInterface.GroupLock* Will specify access that allows update access to all users within the group of the user who created the object. Only the user who created the object can read it.
- *org.ozoneDB.OzoneInterface.Public* Will specify access that allows read and update to all users. This is the same as 'AllRead | AllLock'.

### 6.3.13 Configuration

#### 6.3.13.1 Setting Parameters

Parameters are set by adding parameters to the '*OZconfig.properties*'. This file contains all of the parameters that were used in the creation of the Ozone database. If any parameters are changed, this file is overwritten. There are some parameters (like clusterSize) that can be changed after the database has been created, but there are others (like compressClusters) that if changed will result in a database that will not startup properly.

#### 6.3.13.2 Classic Store vs. Wizzard Store

The following is from the Ozone developers news group written by Falco as a description of the difference between the original Classic Store and the new Wizard Store.

In short: we started out with classic store where we tried to achieve maximum flexibility by serializing each object separately and storing those binary objects into clusters. This allowed us to build and rebuild clusters whenever needed. Later we focused on XML and tried to support this better. XML brings a lot of very small objects. They are highly connected and it is very likely that the access is very "local". And, many if not all of the objects have the same access rights. So we tried to support this in the wizard store. The wizard store serializes the entire cluster together which saves a lot of memory. And the wizard store is optimized for in-cluster object-to-object accesses, which is characteristic for XML data (DOM).

## 6.4 Installation

### 6.4.1 First steps

To install Ozone, we first have do download the releases from Ozone's developer site. There are two choices, we can download the binary file for installation or building it from the source code. For the first case, we then just have to extract the downloaded file to a specific directory, for the second one, the following commands will do:

```
~/ozone> cd server
~/ozone/server> build.sh
```

This is the way to compile under Linux, under windows, things work also in the same way with the equivalent *build.bat* file.

### 6.4.2 Setting environment variables

To get Ozone ready, we will have to set some environment variables. There are a number of environment variables that control ozone compilation and runtime properties. On Linux (bash) you may set variables via:
*export <variable>=<value>*
on windows this is done via
*set <variable>=<value>*
Those environment are:

- OZONE_HOME : The home directory of your ozone installation. This variable must be set
- PATH : The PATH variable is not ozone specific but helps you starting the ozone server and tools. Add <OZONE_HOME>/bin to your PATH.
- OZONE_JVM : This is the JVM which is used to run the server, install and all other tools. It defaults to 'java'.
- OZONE_JAVAC : OPP (the Ozone Post Processor) uses this compiler to compile generated source files. It defaults to 'javac'.

### 6.4.3 Running the ozone server

- Initialize database directory.

  A database directory holds all files of an ozone database. To initialize an empty database start the ozone install tool:

  *ozoneInst -d<data_directory>*.To see all possible parameters of ozoneInst type '*ozoneInst -h*'.

- Start the server.

  *ozone -d<data_directory> -u<username>*

  After the initialization ozone is up and running, some status information will be shown. We can setup as many databases and we can run as many servers as we need on one system.

To define users and to do other administration things use the admin tool:
*ozoneAdmin*
To see all possible commands and parameters of ozoneAdmin type 'ozoneAdmin -h'.
You can also try the adminGui which is a Swing based admin tool
*ozoneAdminGui*

### 6.4.4 Trying out the ozone samples

There is one important thing that I think we must keep in mind before running Ozone : the VM of the ozone server must be able to access the class files of the application. To run the ozone samples, we may add "." to our CLASSPATH and start the ozone server INSIDE THE SAMPLE DIRECTORY.

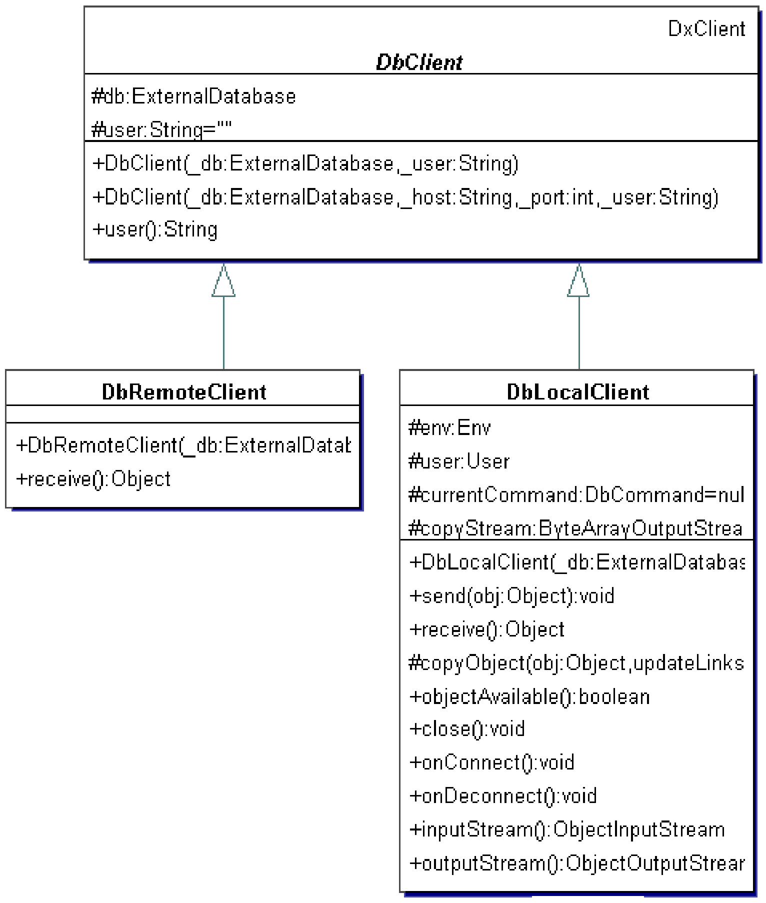

DxClient
DbClient
#db:ExternalDatabase
#user:String=""
+DbClient(_db:ExternalDatabase,_user:String)
+DbClient(_db:ExternalDatabase,_host:String,_port:int,_user:String)
+user():String
DbRemoteClient
+DbRemoteClient(_db:ExternalDatat
+receive():Object
DbLocalClient
#env:Env
#user:User
#currentCommand:DbCommand=nul
#copyStream:ByteArrayOutputStrea
+DbLocalClient(_db:ExternalDatabas
+send(obj:Object):void
+receive():Object
#copyObject(obj:Object,updateLinks
+objectAvailable():boolean
+close():void
+onConnect():void
+onDeconnect():void
+inputStream():ObjectInputStream
+outputStream():ObjectOutputStrear

Figure 6.2: Ozone Class hierarchy has been used to deploy my application

- Start the server.

  Open a new console. Change directory to *<OZONE_HOME>/samples/simple*. Install a new database via *'ozoneInst -d<db_dir>'*. Start the server via *'ozone -d<db_dir> -u<username>'*.

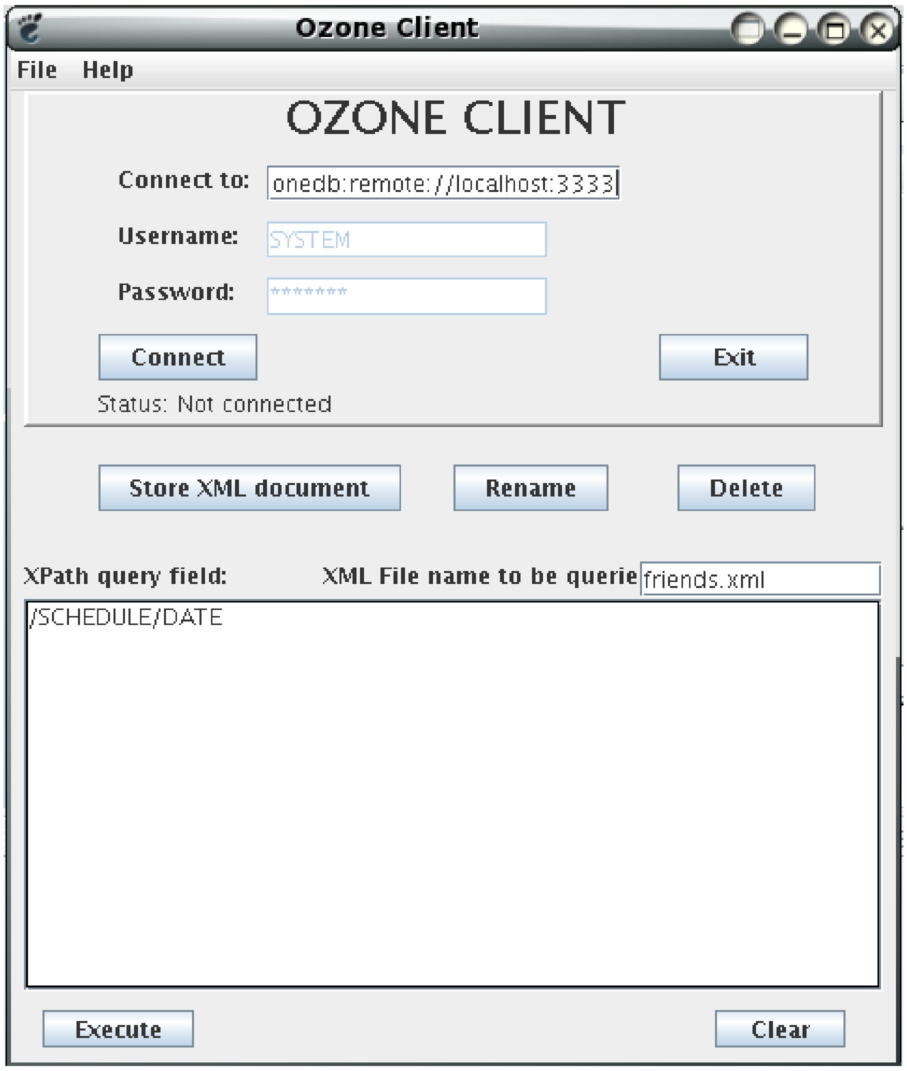
- Start the Client.

  Open a new console. Change directory to *<OZONE_HOME>/samples/simple*. Build the client via *'../../build.[sh | bat]'* or *'make'*. Start the client via *'ojvm Client'*.

That's it!

## 6.5 Application

The application that I designed is an Ozone XML Client, this program establishes a connection to Ozone server then performs XML manipulation on some provided database (Figure 6.3 ). Xpath statements can be executed and the result will be shown directly on the same window. The program consists of of a connection panel and a query panel where we can test several commands and XPath syntaxes.



Connect to: onedb:remote://localhost:3333

Username: SYSTEM

Password: ********

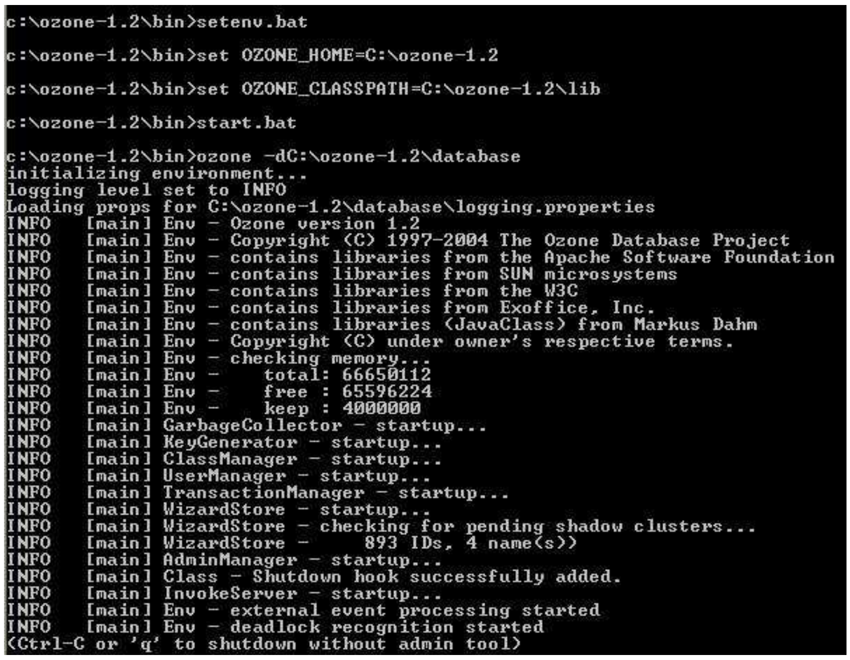
Status: Not connected

XPath query field:

XML File name to be querie friends.xml

/SCHEDULE/DATE

Figure 6.3: My Ozone XML Client



```
c:\ozone-1.2\bin>setenv.bat

c:\ozone-1.2\bin>set OZONE_HOME=C:\ozone-1.2

c:\ozone-1.2\bin>set OZONE_CLASSPATH=C:\ozone-1.2\lib

c:\ozone-1.2\bin>start.bat

c:\ozone-1.2\bin>ozone -dC:\ozone-1.2\database
initializing environment...
logging level set to INFO
Loading props for C:\ozone-1.2\database\logging.properties
INFO   [main] Env - Ozone version 1.2
INFO   [main] Env - Copyright (C) 1997-2004 The Ozone Database Project
INFO   [main] Env - contains libraries from the Apache Software Foundation
INFO   [main] Env - contains libraries from SUN microsystems
INFO   [main] Env - contains libraries from the W3C
INFO   [main] Env - contains libraries from Exoffice, Inc.
INFO   [main] Env - contains libraries (JavaClass) from Markus Dahm
INFO   [main] Env - Copyright (C) under owner's respective terms.
INFO   [main] Env - checking memory...
INFO   [main] Env -     total: 66650112
INFO   [main] Env -     free : 65596224
INFO   [main] Env -     keep : 4000000
INFO   [main] GarbageCollector - startup...
INFO   [main] KeyGenerator - startup...
INFO   [main] ClassManager - startup...
INFO   [main] UserManager - startup...
INFO   [main] TransactionManager - startup...
INFO   [main] WizardStore - startup...
INFO   [main] WizardStore - checking for pending shadow clusters...
INFO   [main] WizardStore -      893 IDs, 4 name(s))
INFO   [main] AdminManager - startup...
INFO   [main] Class - Shutdown hook successfully added.
INFO   [main] InvokeServer - startup...
INFO   [main] Env - external event processing started
INFO   [main] Env - deadlock recognition started
(Ctrl-C or 'q' to shutdown without admin tool)
```

Figure 6.4: Running Ozone server

To run the application, first we have to start the server. Before this, we will have to set the environment variables, under Windows, for the purpose of convenience I created a setenv.bat file which consists of these lines to set two environment variables:

set OZONE_HOME=C:\ozone-1.2

set OZONE_CLASSPATH=C:\ozone-1.2\lib

After that, a start.bat file is created. The command to run the created database that has already been stored in the directory C:\ozone-1.2\database is *ozone -dC:\ozone-1.2\database*

-d parameter is to specify the directory location where I created the tv database. After this, at the command prompt , we just have to run the two .bat file to start Ozone server.

Now that the server is ready to begin, by calling *start.bat* we just in other way run *ozone -dC:\ozone-1.2\database.* If everything works fine, the status information will be shown (Figure 6.4).

The server is now ready, we can start our application and establish a connection to the running server. Launch the application, after filling the hostname, port and username, password, (by default I used SYSTEM as username, MANAGER as password), hit Connect, this will establish the connection to server by calling the function *openDatabase(dbURL)* of class *ExternalDatabase*:

*ExternalDatabase db = ExternalDatabase.openDatabase(dbURL);*

where dbURL = "ozonedb:remote://localhost:3333" by default.

If successful, the Status will display: Connected to .. otherwise an Exception will be thrown. (Figure 6.5)

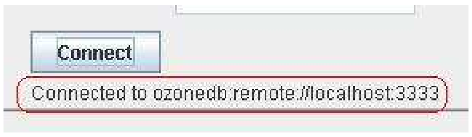


Figure 6.5: Connected to Ozone server

After got connected, we can try to add a new XML document into the database. This can simply be done by clicking on “Store XML document” button.

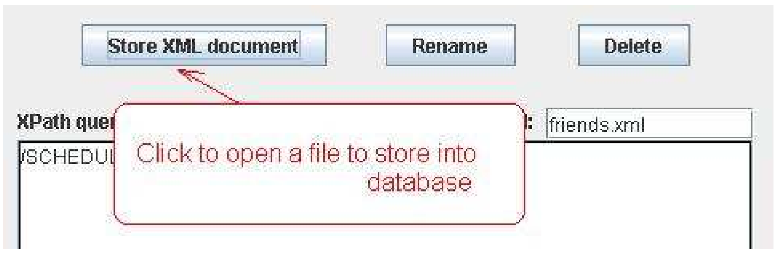
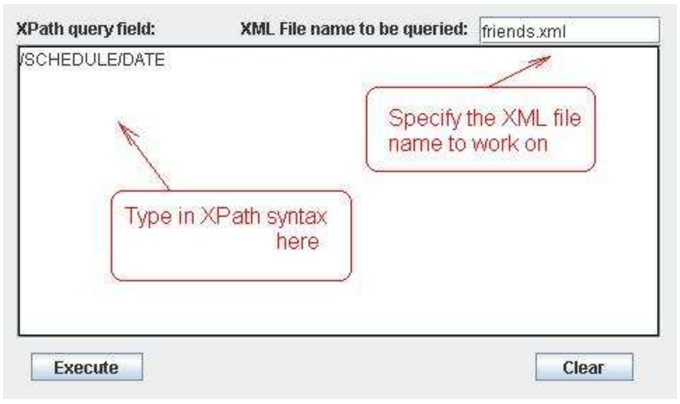


Figure 6.6: Store XML document

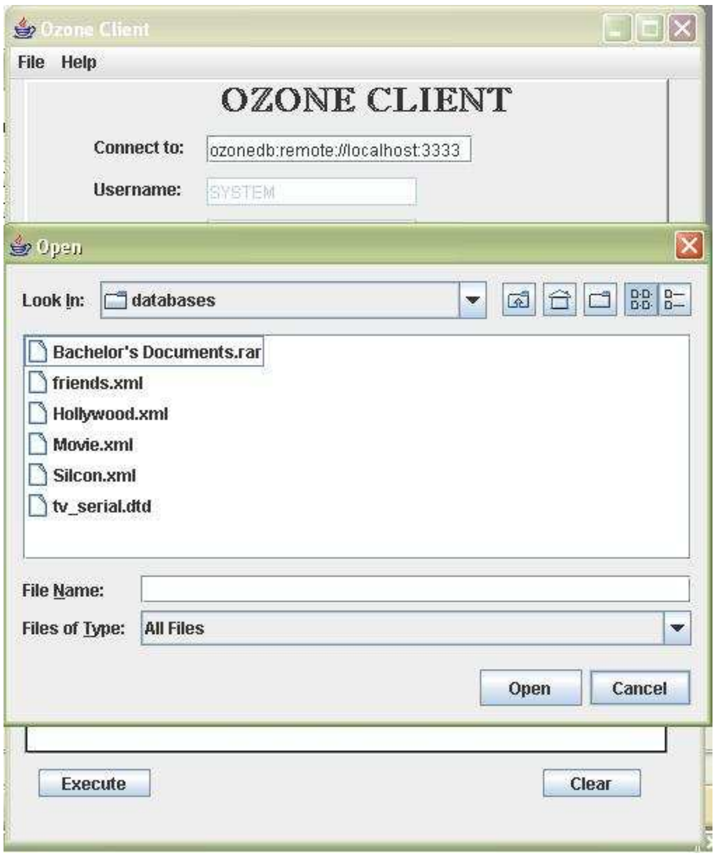
An open dialog will appear to choose the file that we want to store. Choose an XML file, “friends.xml” in our case then click on “Open”, the chosen XML file will be stored into the database. In the source code, this is done in the function *protected static void* ***domStore****( String filename ) throws Exception.*


XPath query field:
XML File name to be queried:
Specify the XML file name to work on
Type in XPath syntax here
Execute
Clear

XML File name to be queried: friends.xml

/SCHEDULE/DATE

Figure 6.8: Xpath field



Connect to: ozonedb:remote://localhost:3333

Username: SYSTEM

Look In: databases

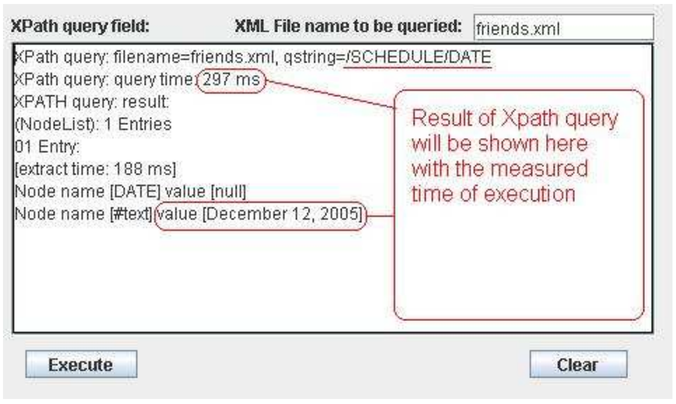
Bachelor's Documents.rar
friends.xml
Hollywood.xml
Movie.xml
Silcon.xml
tv_serial.dtd

Files of Type: All Files

Figure 6.7: Choosing an XML file to store

In the Xpath field, we can try several Xpath syntaxes to be executed. We can simply do that by specify the file name that we want to query and type our Xpath syntax in the Xpath field. Hit "Execute" when ready, the result of given Xpath syntax will be shown. Functions *public void* ***xpathQuery***( *String filename,String qstring* ) and *public void* ***printNodeTree***( *Node node, StringBuffer indent* ) in my source is the ones responsible for this.

With the given XML file, let's try to print out the date of the schedule of the series store


XPath query field:
XML File name to be queried:
friends.xml
Result of Xpath query will be shown here with the measured time of execution
Execute
Clear

```
XPath query: filename=friends.xml, qstring=/SCHEDULE/DATE
XPath query: query time: 297 ms
XPATH query: result:
(NodeList): 1 Entries
01 Entry:
[extract time: 188 ms]
Node name [DATE] value [null]
Node name [#text] value [December 12, 2005]
```

Figure 6.9: Result of Xpath query

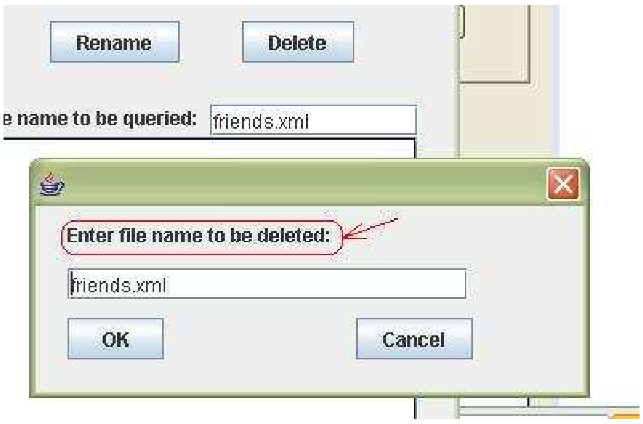


Figure 6.10: Deleting a file

in our tv database under the file "friends.xml". The Xpath syntax for this is "/SCHEDULE/DATE". After execution, the displayed result contains information about the nodes specified by the given Xpath plus information about execution time of the query and extracting time on the server.

Ozone is good for XML documents manipulation, we can rename of delete our stored document in the database. Let's try to delete the "friends.xml" document that we have just stored. Simply click on "Delete", a dialog will appear asking for the name of the file that we want to delete. Type in "friends.xml" click ok, then the file will be deleted from our database (Figure 6.10). This is done by the function *protected static void **delete**( String filename ) throws Exception* in my source code. Similarly with the renaming process, we can just simply click on "Rename" then provide the new name for the file we want to change. This is done by the function:

*protected static void **rename**( String filename ) throws Exception.*

## 6.6 Conclusion

The Ozone Database Project is a open initiative for the creation of an open source, Java based, object-oriented database management system. Ozone is a fully featured, object-oriented database management system completely implemented in Java and distributed under an open source license. The ozone project aims to evolve a database system that allows developers to build pure object-oriented, pure Java database applications. Ozone deals with Java objects and let them run in a transactional database environment.

Besides the native API, Ozone also provides a ODMG 3.0 interface. Ozone does not depend on any back-end database or mapping technology to actually save objects. It contains its own clustered storage and cache system to handle persistent Java objects.

Ozone includes a fully W3C compliant DOM implementation that allows us to store XML data. However, according to my point of view, Ozone is not really as good as an XML database management system. Ozone is strong in XML document manipulation, including functionalities of Renaming, Deleting, Storing. Ozone treats XML document as an object, each file will be stored and manipulated in a fully object-oriented model. Ozone shows a lot of lacks for becoming a real strong XML DBMS especially the lack of support for Xquery, currently only Xpath is supported.

# Chapter 7

# NeoCore XML Management System

## 7.1 Introduction

NeoCoreTM XML Information Management System (or shortly XMS) is a high-speed, extremely flexible tool for managing information. NeoCore Inc. has developed NeoCore XMS to store and retrieve Extensible Markup Language (XML) data and its context. Using an XML database removes many of the impediments that accompany other solutions.

XMS includes the Server Administration Console, the NeoServer, and the APIs.

More than this, NeoCore XMS is a powerful XML information management system, with many features and functions. NeoCore XMS documentation set includes two guides that cover specific tasks. The System Administration Guide provides a functional overview and detailed instruction on installing, configuring, using, and maintaining NeoCore XMS, as well as preserving data integrity. The System Programming Guide provides more detailed information on NeoCore XMS best practices, architecture, using XQuery for XML manipulation, the APIs for Java, C++, Component Object Model (COM) Wrapper, and Hypertext Transfer Protocol (HTTP), as well as application notes.

## 7.2 Architecture

NeoCore XMS is a high-performance XML database management system that easily and dynamically adapts to heterogeneous and continually changing information and business requirements. This system is powered by the company's Digital Pattern Processing (DPP) engine, a series of algorithms (Xpriori has six issued patents and 20 pending) that are designed to recognize patterns within data, and rapidly accelerate the processing of that data by looking for the resulting data pattern vs. the full data string.

Xpriori has combined the extensibility of XML with its DPP core technology to create the NeoCore XML Information Management System (XMS). Through its flexible architectural core, XMS delivers all of the benefits of a traditional relational database management system (RDMS), including advanced querying and multi-dimensional data analysis, without the traditional structural and processing constraints or compromises inherent in such traditional database architectures. XMS users realize a compelling ROI through their ability to rapidly deploy high-performing and easily maintained and customized applications (including some that would not be possible to develop using existing technologies or products) that fully leverage their business information.

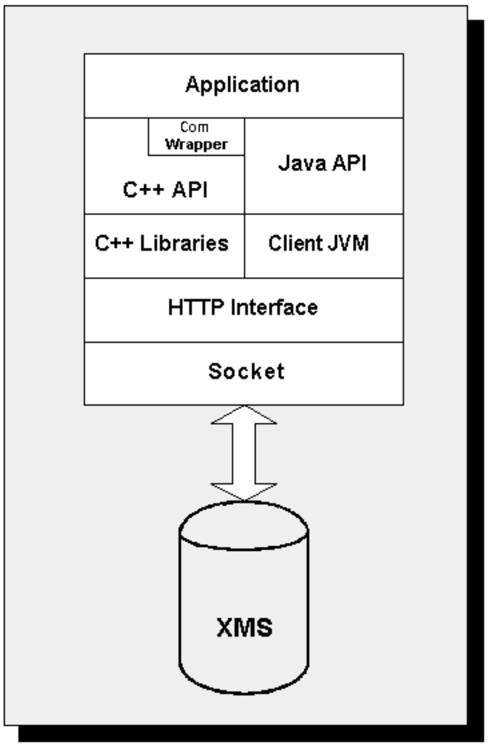


Figure 7.1: Neocore Client Side view

### 7.2.1 Client-Side View

The protocol level interface to NeoCore XMS is HTTP over TCP. The application may either generate http directly, or use the client-side APIs to generate the http on its behalf. Currently, Java and C++ APIs are provided. In addition, the C++ API has a COM wrapper to support VB and other Microsoft programmers (see Figure 7.1).

### 7.2.2 Server-Side View

This figure illustrates the high level architecture of the XMS server. The Server Process controls the execution of the server and relies on the Server Component manager to load the right components for execution. Currently, XMS only supports an HTTP component. The HTTP Server Component processes the HTTP commands and sends them to the proper subcomponent (see Figure 7.2).

The Admin Module is responsible for processing administration commands, including statistics and server administration. These commands do not affect the transactional processing of the database. These commands are accessible via all APIs, but must be accessed through the administration port.

The XMS module processes commands from the HTTP interface. These commands are predominantly transactional commands that query, modify, and store XML into the XMS. These commands can be accessed via the normal database port or via the Administration port.

XMS includes a server side JVM to handle extended server side processing. Currently, the

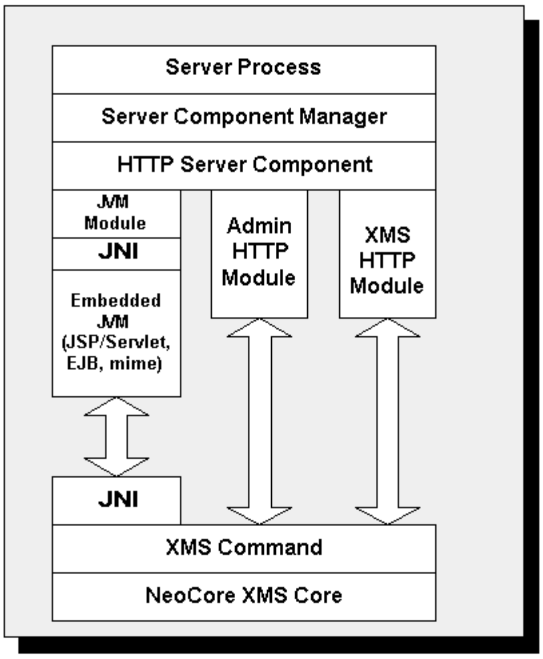


Figure 7.2: Neocore Sever-side View

XSLT transform engine runs in this JVM. Other applications can extend server functionality in this embedded environment. Note, however, that as of Version 2.7, the interface to this JVMS is not exposed to end users, but is used by Xpriori alone to deploy additional server-side functionality. Later releases may expose this interface.

XMS Command is the layer of the XMS that handles client requests. This includes the normal database commands and the administration commands, but not extended server side processing, which is mediated through a JNI layer.

The server core actually executes client requests, manipulating the internal data structures, enforcing access control, and maintaining transactional integrity.

## 7.3 Installation

XMS comes with the possibility of running on both platforms Linux and Windows, this is a real advantage for a commercial product. For the installation process, we can download both installation files for Linux or Windows. Under Linux, a binary file is provided, namely "*xms-version-dev-lin32.bin*", the latest version is 3.2.1.22.

Executing the setup file with root privileges, a graphical user interface of the installation process is displayed, this makes it very easy for any user to follow and configure the setup parameters. Setup requires to create a *neoadmin* user and add it to the group named *neocore.* After that you will be asked to provide username and password for log in to console as well as password for Administrator.

## 7.4 Administration

### 7.4.1 Starting and stopping Neocore

- The preferred way to start the NeoCore Server under Linux is to use the /etc/init.d/neocore script. This file is modified to correctly run the server at install time. As root or neoadmin, type:

  */etc/init.d/neocore start*

- Recommended stop method: Use the /etc/init.d/neocore script. As root, type:

  */etc/init.d/neocore stop*

  or we can stop the server using Administration Console:

  1. Go to the Server Administration Console select the Manage Server tab.

  2. Select the ShutDown button.

  3. Select the ShutDownXMS Server button.

### 7.4.2 Launching the console

We can launch the console one of three ways:

- Select Launch Console in Start/Programs/NeoCore XMS/ (use this to access the server when both are on the same machine) if we installed NeoCore under Windows.

- Start the web browser and select: *http://<servername>:port/console* (use this to access a server that is on a different machine than the console). Examples: *http://localhost:7701/console*

Once we have started the console, we are prompted to log in. Use the Server Administrator username and password selected during installation. The server administrator username and password can be different from the database administrator password used to create the database.

Keep in mind that NeoCore XMS uses two ports. Port 7700 is the default for the database, and port 7701 is the default for the administrator console. These ports are set up during installation but can be modified in the server configuration file called NeoServer.xml.

As a commercial product, Neocore comes with a great Server Administration Console, here we can totally manipulate the whole Neocore sever with a lot of possibilities:

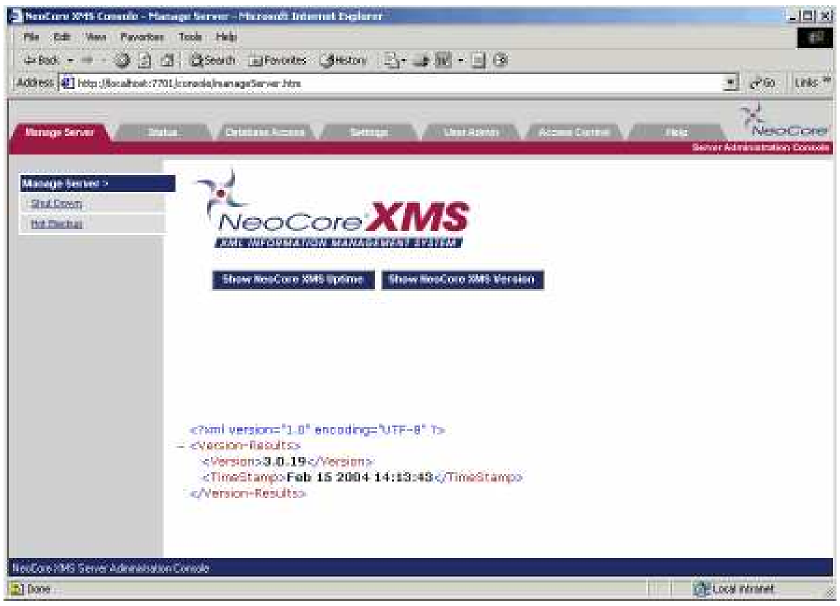
- *Manage Server*, the Manage Server tab is used to view server uptime and version, to shut down the database and server, and to perform Hot Backup.

  – by selecting the Show XMS Uptime button we can view the server uptime.
  – by selecting the Show XMS Version button we can view the server Version.

*Status*, the Status tab has the following options:

- Server Status (view status of the server)

  – XMS Status (view status of the database)


NeoCore XMS Console - Manage Server - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:7701/console/manageServer.htm
Manage Server
Status
Database Access
Settings
User Admin
Access Control
Help
NeoCore
Server Administration Console
Manage Server >
Shut Down
Hot Backup
NeoCore XMS
XML INFORMATION MANAGEMENT SYSTEM
Show NeoCore XMS Uptime
Show NeoCore XMS Version
NeoCore XMS Server Administration Console
Done
Local intranet

```
<?xml version="1.0" encoding="UTF-8" ?>
- <Version-Results>
    <Version>3.0.19</Version>
    <TimeStamp>Feb 15 2004 14:13:43</TimeStamp>
  </Version-Results>
```

Figure 7.3: Administration console

  - Server Log (view the current database log file)
  - Access Log (view the server access log file [if enabled])

- *Database Access*, this tab allows us to log in to the database and use the following functions:

  - Store (store an XML document)
  - Copy (copy an XML document)
  - Query (use XPath to query the server)
  - Insert (insert XML into an existing document)
  - Modify (modify an element or attribute in an XML document)
  - Delete (delete XML in the database)

- *Settings*, the Settings tab includes the following options:

  - Server Settings (view the server settings)
  - XMS Runtime Settings (view the database settings that can be changed at database startup)
  - XMS Static Settings (view the database that can only be changed when the server is created or rebuilt)

- *User Admin*, the User Admin tab has the following options:

  - Manage Users
  - Add New Users

  – Manage Groups
  – Add New Group

- *Access Control*, the Access Control tab lets us select Access Control for Users or Groups.

## 7.5 Application

### 7.5.1 NeoCore APIs

The NeoCore XMS application programming interfaces (APIs) are designed to support rapid development and integration of customer-specific applications with NeoCore XMS. The APIs are delivered as:

- Windows DLL - C++
- Solaris shared library - C++
- Java Proxy Classes - Java
- Component Object Model (COM) Wrapper
- HTTP

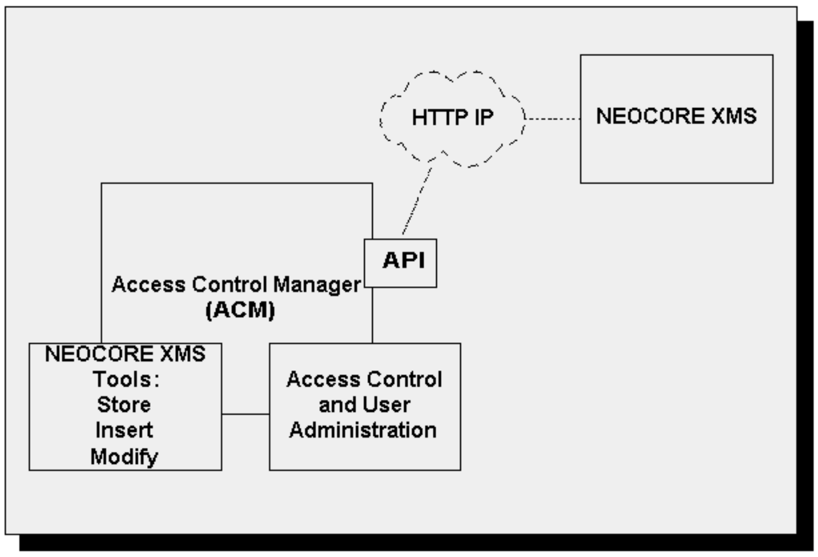


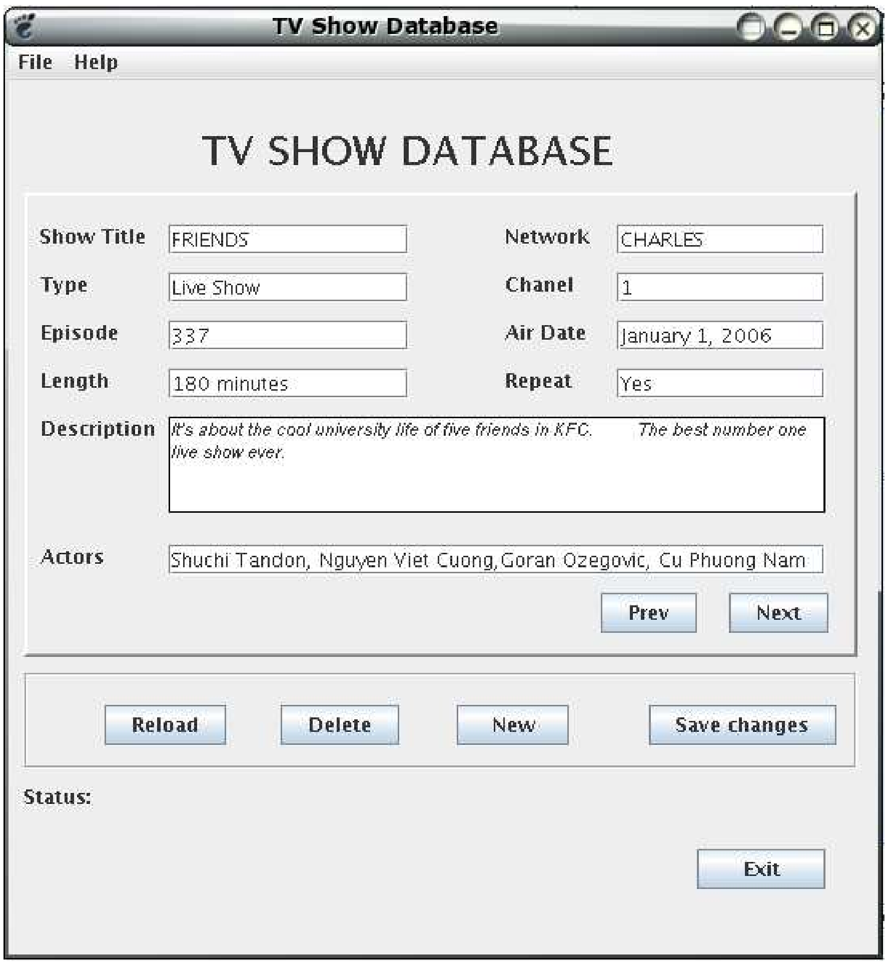
Figure 7.4: Necore API

### 7.5.2 My Application

My application is a TV database program, written in Java using Neocore APIs provided for Java. The application will firstly connect to the server, using its own username and password to login and access its XML database. After getting connected, the information about the TV shows will be displayed with the possibilities of modifying, deleting, inserting new entries.

As shown in figure 7.5, the application consists of a data presentation panel, where information about Show title, Type, Episode, Actors, etc. are displayed. We can move back and forth to view next and previous show by pressing “Prev” and “Next”. Each entry is a TextBox, we can directly edit each entry and press “Save changes” to save the modified field. We can also create a new show database by press “New” button, after that we can fill all the information about the show and press “Save changes” to store. Similarly, we can delete or just simple reload the show database by pressing “Delete” and “Reload”.

The application works in the following way: firstly, it will try to connect to NeoCore database server at localhost using username=”*cuong*”, password=”*tv*”. If the server is ready and username and password are correctly provided, it will display the status: “Connected”. Once we got the connection, database about shows will be read by execution of XPath queries using the appropriate API functions provided by the developers. From this point on, we can perform any action of deleting, creating or modifying the data.



TV SHOW DATABASE

| | | | |
|---|---|---|---|
| Show Title | FRIENDS | Network | CHARLES |
| Type | Live Show | Chanel | 1 |
| Episode | 337 | Air Date | January 1, 2006 |
| Length | 180 minutes | Repeat | Yes |

Description: *It's about the cool university life of five friends in KFC. The best number one live show ever.*

Actors: Shuchi Tandon, Nguyen Viet Cuong,Goran Ozegovic, Cu Phuong Nam

Status:

Figure 7.5: TV show database application

## 7.6 Conclusion

Being entirely self-constructing, the native XML database eliminates database programming and design and frees application developers to concentrate all their efforts on application development. Changes are supported seamlessly, requiring no database redesign or re-indexing instructions. XMS is uniquely flexible and can support all types of content - data, office documents, forms, multimedia content, anything expressed in XML - in an unified, fully transactional and access controlled environment. XMS is open standards based (HTTP, XML, XSLT, XPATH, XQUERY, WebDAV), insuring compatibility with all popular development platforms.

NeoCore XML Management System (XMS) is a fully transactional native XML database system that serves as a bi-directional web server, accepting and returning XML documents and fragments via HTTP(S). It supports all basic database functions, including storage, delete, copy, and query for XML documents, and insert, modify, and query for XML data elements. NeoCore XMS is schema-independent. Specific features of NeoCore XMS include XPath-based query support, access control, user-defined document data management, GUI-based administration, and session control. NeoCore XMS uses a variant of XPath as its query language; query responses return elements, document fragments, full documents, and multiple documents. NeoCore XMS has built-in access control to set permissions at the document or fragment level, and at user or group levels. NeoCore XMS also supports access control by specifying IP addresses and supporting X.509 certificates via Netscape Server. Interfaces to NeoCore XMS include HTTP(S)/SSL, Java, C++, and Microsoft COM. However, as a commercial product, NeoCore XMS still need improvements to continuously optimize its efficiency to become more competitive in the market.

# Chapter 8

# Reviews and Comparisons

One of the most difficult aspects of selecting a technology is getting an unbiased view of the pros and cons. Anyone who has found themselves mired in the marketing gloss applied to many white papers while questing for concrete technical information will be familiar with this problem.

This is where high quality user communities, like XML-DEV, can have their greatest benefit: providing a forum for objective discussion on the relative merits of particular technologies. Armed with facts, and aware of the trade-offs, the plucky developer is better able to make an informed decision. This process can be facilitated if others are willing to summarize these findings and report them to the community. In the world of databases and content management where attention has been drawn lately to XML and, specifically, XML databases, things work in the same way. It is best to get a good indication of the state of XML-based content management technology by examining developments at the ground floor: the XML native database systems that form a base for databases and larger content management applications. Back to our process, to evaluate the advantages and disadvantages of some native database systems, it requires the involvement and experience of users. Each may have different judgements on the same aspect and may come up with different conclusions.

In general, all the three chosen XML native database systems have their own qualities. Ozone in fact is an object-oriented database management system based on Java and XML database with a fast and scalable solution to store large XML datasets is just one of its features. Sedna also comes with great functionalities and supports which can be considered as a fully-featured XML native database system. On the other hand, NeoCore XMS is good at query optimization and user-friendly while still maintains the flexibility of deploying applications. Let us now sum up all the worth-to-mention key features of Sedna, Ozone and NeoCore XMS:

| **Sedna** | **Ozone** | **Neocore** |
|---|---|---|
| • Available for free in open source form under Apache License 2.0<br>• Support for the W3C XQuery language<br>• Support for a declarative update language<br>• Support for SQL connection from XQuery<br>• Native XML data storage structures designed for efficient support for both queries and updates<br>• Support for transactions (rollback and concurrency control)<br>• Support for database security (users and privileges)<br>• Support for regular expressions | • Available for free in open source form under GNU Library General Public License and GNU General Public License<br>• Multi-user, multi-thread support<br>• Object level access rights<br>• Fully transaction based<br>• JTA/XA support<br>• Deadlock recognition<br>• BLOB support<br>• XML (DOM) support<br>• ODMG 3.0 support<br>• Garbage collection<br>• Enhancements to object lifecycle<br>• Enhanced exception handling | • NeoCore XMS comes with 3 different versions: Enterprise Server, Workgroup Server and Development Sever. We may use NeoCore® XMS-Developer as long as we need for the purposes of internal development and test<br>• Supports all basic database functions, including storage, delete, copy, and query for XML documents, and insert, modify, and query for XML data elements<br>• open standards based (HTTP, XML, XSLT, XPATH, XQUERY, WebDAV)XPath-based query support<br>• Access control user-defined document data management |

| **Sedna** | **Ozone** | **NeoCore** |
|---|---|---|
| • Support for Unicode (utf8)<br>• Java API, Scheme API and C API for application development<br>• Open client/server protocol over sockets that allows implementing APIs for other programming languages<br>• Integration with Apache HTTP server<br>• Supplied with extensive documentation<br>• Support both Windows and Linux platforms | • OPP generates a factory which simplifies object creation<br>• New Module - Persistent Collections<br>• AdminGUI - a swing client alternative to the command line interface<br>• Integration with the JBoss application server<br>• Support for Windows and Linux platforms | • GUI-based administration, and session control.<br>• Built-in access control to set permissions at the document or fragment level, and at user or group levels.<br>• Supports access control by specifying IP addresses<br>• Interfaces include HTTP(S)/SSL, Java, C++, .NET and Microsoft COM.<br>• Runs on Windows, Solaris and Linux platforms |

# Chapter 9

# Conclusions

XML features are popping up in applications like flowers after a spring rain. And it hasn't taken long for a new problem to blossom-managing XML data in an organized way. The native XML database has jumped ahead to simplify the problem. Any company with an XML strategy has good reason to use a native XML database. XML and SQL just don't match on a number of levels. They use different data types, and XML's ability to change structure in the middle of a document does not mesh with a relational database's rigid table structures. Also, when an XML document is stored in a relational table, information can be lost, such as element ordering and the distinction between attributes and elements.

In fact, all the giant database vendors of the world like IBMs, Microsofts, Oracles, and Sybases are not ignoring the XML trend. They already believe in XML as the future of data management and are leaning hard on their development teams to implement strong XML capabilities in the next releases of their products. Some content management systems are being built on native XML databases rather than the file system or relational databases.

Focus on the scope of this thesis, which is mainly to discover the myth of XML native database, three XML native database systems were installed and reviewed. Two of the chosen systems are in the opensource form, the other is a commercial product. The comparisons base on the metric of Supported Platforms, License, Query languages supports, Access Control, Database type, Storage limits, Concurrent access, API supports, Documentation and the rating of the author himself is briefly summarized by the following table:

| | **Sedna** | **Ozone** | **NeoCore XMS** |
|---|---|---|---|
| Supported Platforms | *Windows, Linux* | *Windows, Linux* | *Windows, Linux, Solaris* |
| License | *Open source* | *Open source* | *Commercial* |
| Query lang. support | *Xpath, Xquery* | *Xpath* | *Xpath, Xquery* |
| Access Control | *Yes* | *Yes* | *Yes* |
| Database type | *Proprietary* | *Object-oriented* | *Proprietary* |
| Storage limits | *Unlimited* | *Unlimited* | *10 Million nodes** |
| Concurrent access | *2-phase locking protocol* | *Deadlock recognition* | *Only 2 sessions at a time** |
| APIs support | *Java, C/C++, Scheme, .NET, Python, HTTP* | *Java, DOM support* | *.NET, HTTP, Java, C++, Eiffel, WebDAV* |
| Documentation | *Good* | *Poor* | *Good* |
| Back up strategy | | | *Yes* |
| My rating | *8/10* | *6/10* | *7/10* |

⋆ NeoCore XMS has 3 different versions: Enterprise Server, Workgroup Server and Development Sever[1].

After going through the process of gaining idea and knowledge about XML native database systems, I would like to sum up my views and experience about the three chosen ones here. In general, Ozone is strong in XML document manipulation, including the functionalities of Renaming, Deleting and Storing. Ozone treats XML document as an object, each file will be stored and treated in a fully object-oriented way. However, one of the serious lacks of Ozone is its ability of query languages especially the lack of support for Xquery, currently just Xpath queries can be accepted. Another disadvantage of Ozone is the documentation, although the developers have provided a system of documentation including Administration and Configuration, Ozone Developers' Guide, Ozone Users Guide, it's still not a comprehensive and transparent cook-book for normal user to understand and easily work with Ozone.

Neocore XMS is very user-friendly, easy to use and manipulate with a great administration GUI, I must say. NeoCore Administration Console categorizes the functionalities according to purpose and properties, this make normal users feel more comfortable and reduce the amount of configuration work . The installation of Neocore is simple since it's done just by clicking and following the graphical user interface of the installation program. NeoCore supports both platforms Windows and Linux with great amount of examples and tutorials. Even being a commercial product, NeoCore XMS-Developer is still available as long as it is used for the purposes of internal development and evaluation, while almost all other XML commercial products just let us use them for an evaluation time of about 30 days.

Sedna is a multiple-platform XML native database system, as an opensource project, Sedna allows us to freely obtain and use for any non-commercial purpose with no restriction under Apache License 2.0. Sedna fully supports W3C compliant XQuery and has a mechanism for a declarative update language. The provided APIs are comprehensive and easy to use. Sedna comes with efficient support for stand-alone XML documents with arbitrary structures and collections of XML documents with similar structures. Working with Sedna is comfortable since the directory structure are well organized. Besides, a great advantage of

[1]The attributes marked with * is valid for a free Developer version of NeoCore XMS. We may use NeoCore ® XMS-Developer as long as we need for the purposes of internal development and test. In the Enterprise version, NeoCore supports *unlimited concurrent sessions* and *unlimited storage*.

Sedna lies in the documentation. With several guides, I found Sedna's documentation really helpful and coherent. However, nothing is perfect, Sedna still has some drawbacks that the developers must improve to make Sedna and more competitive. One of the drawback is the optimization of XQuery and resource management. Sedna runs a bit slowly on my laptop, using a great deal of physical memory while NeoCore for instance, works adequately well even under huge amount of processing. Despite these drawbacks, in general, Sedna is still a promising full-featured XML native database management system.

Finally, according to my thoughts, if there was a rating in the scale of 10, I would rate Sedna for 8/10, NeoCore XMS for 7/10 and Ozone 6/10. If anybody happened to choose one of these XML Native Database Systems, I would recommend Sedna for the sake of convenience, finance-free and great functionalities.

# Bibliography

[1] Kimbro Staken, *Introduction to Native XML Databases* , October 31, 2001, http://www.xml.com/pub/a/2001/10/31/nativexmldb.html

[2] W3C Working Draft, *XQuery 1.0 and XPath 2.0 Data Model*, 12 November 2003, http://www.w3.org/TR/2003/WD-xpath-datamodel- 20031112/

[3] Ronald Bourret, *XML Database Products: Native XML Databases*, Copyright 2000-2005, http://www.rpbourret.com/xml/ProdsNative.htm

[4] Modis group, *Sedna XML DBMS*, http://modis.ispras.ru/Development/sedna.htm

[5] *XMach-1: A Benchmark for XML Data Management* http://dbs.uni-leipzig.de/en/projekte/XML/XmlBenchmarking.html

[6] *The Michigan Benchmark for XML Data Management*, http://www.eecs.umich.edu/db/mbench/

[7] *XMark - An XML Benchmark Project*, http://monetdb.cwi.nl/xml/index.html

[8] *The XOO7 Benchmark*, http://www.comp.nus.edu.sg/~ebh/XOO7.html

[9] Elliotte Rusty Harold, *Managing XML data: Native XML databases*, Polytechnic University, Jun 06 2005. http://www-128.ibm.com/developerworks/xml/library/x-mxd4.html?ca=dnt-623

[10] John Wood, *Native XML Databases, SQL Server and the future of databases* http://www.dotnetjunkies.com/WebLog/johnwood/archive/2005/04/30/72480.aspx

[11] Anders Mueller & Michael I. Schwartzbach BRICS, *The XML Revolution Technologies for the future Web*, , University of Aarhus, http://www.brics.dk/~amoeller/XML/

[12] T. Bohme, E. Rahm, *Benchmarking XML Database Systems – First Experiences*, Position Paper, Ninth International Workshop on High Performance Transaction Systems (HPTS), Pacific Grove, California, 14.-17. October, 2001

[13] Stephane Bressan, Mong Li Lee, Ying Guang Li, Z. Lacroix and Ullas Nambiar: *The XOO7 XML Management System Benchmark*, NUS CS Dept Technical Report TR21/00, November, 2001.

[14] Rob Lapensee, *Ozone Administration and Configuration Manual*, Copyright © 2002 SMB GmbH. http://sourceforge.net/docman/display_doc.php?docid=10744&group_id=3969

[15] Falko Braeutigam, Gerd Mueller, Per Nyfelt, Leo Mekenkamp, *Ozone Users Guide*, Copyright © 2002-2003 SMB GmbH, http://sourceforge.net/docman/display_doc.php?docid=10396&group_id=39695

[16] Yanick Duchesne and Per Nyfelt, *Ozone Developers' Guide*, Copyright © 2001 SMB GmbH

[17] Xpriori LLC, *NeoCore XMS Quick Start guide*, Copyright © 2004 Xpriori LLC, http://www.xpriori.com

[18] Xpriori LLC, *XMS System Administration Guide* 3.2, Copyright © 2004 Xpriori LLC, http://www.xpriori.com

[19] Jon Udell, *The future of XML documents and relational databases,* July 25, 2003, http://www.infoworld.com/article/03/07/25/29FEdocs_1.html

[20] Timothy Dyck, *Going Native: XML Databases*, June 30 2002, http://www.pcmag.com/article2/0,1895,7122,00.asp