A TOOLKIT FOR MANAGING XML DATA WITH A RELATIONAL DATABASE MANAGEMENT SYSTEM

By

RAMASUBRAMANIAN RAMANI

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2001

To my parents, Yamuna and Ramani, who have given me the best values in life.

ACKNOWLEDGMENTS

This thesis is a result of the motivation and support provided by many individuals. Firstly, I would like to thank Dr. Joachim Hammer who has always remained a constant source of inspiration and technical expertise. His enthusiasm for the subject has been a driving force, channeling my efforts. I am also thankful to Dr. Douglas Dankel and Dr. Herman Lam, who kindly agreed to participate in my supervisory committee. It has been a great honor to be a part of the IWiz development team and to work with my colleagues Anna Teterovskaya, Amit Shah, Charnyote Pluempitiwiriyawej and Rajesh Kanna. I would like to thank Sharon Grant and Mathew Belcher, who deserve a special mention for their support and help in the lab. Finally, I would like to acknowledge the support given by my family members, back in India.

TABLE OF CONTENTS

page

CHAPTERS

LIST OF FIGURES

Figure Page

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

A TOOLKIT FOR MANAGING XML DATA WITH A RELATIONAL DATABASE MANAGEMENT SYSTEM

By

Ramasubramanian Ramani

August 2001

Chairman: Joachim Hammer
Major Department: Computer and Information Science and Engineering

This thesis presents the underlying research, design and implementation of our XML Data Management Toolkit (XML toolkit), which provides the core functionality for storing, querying, and managing XML data using a relational database management system (RDBMS). The XML toolkit is an integral part of the Information Integration Wizard (IWiz) system that is currently under development in the Database Research and Development Center at the University of Florida. IWiz enables the querying of multiple semistructured information sources through one integrated view, thereby removing existing heterogeneities at the structural and semantic levels. IWiz uses a combined mediation/data warehousing approach to retrieve and manage information from the data sources which are represented as semistructured data in IWiz; the internal data model is based on XML and the document object model (DOM). The XML toolkit is part of the Data Warehouse Manager (WHM), which is responsible for caching the results of

frequently accessed queries in the IWiz warehouse for faster response and increased efficiency.

IWiz has two major phases of operation: A *built-time phase* during which the schema creator module of the XML toolkit creates the relational schema for the data warehouse using the DTD description of the global IWiz schema as input. This is followed by the *run-time* or *query phase* during which the warehouse accepts and processes XML-QL queries against the underlying relational database. Note the XML-QL to SQL conversion is part of another ongoing research project in the center. During run-time, the Relational-to-XML-Engine component of the XML toolkit is used to convert relational results from the warehouse into an equivalent XML document that has the same structure as the global IWiz schema. The initial query may also be sent to the mediator in case the contents of the data warehouse are not up-to-date. The loader component of the XML toolkit is used to convert and store XML data from the sources via the mediator into the underlying relational format during warehouse maintenance.

We have implemented a fully functional version of the XML toolkit, which uses Oracle 8*i* as the underlying relational data warehouse engine. The XML toolkit is integrated into the IWiz testbed and is currently undergoing extensive testing.

CHAPTER 1
INTRODUCTION

1.1. Using XML to Represent Semistructured Data

The Web is a vast data store for information and is growing at a fast rate. This information can originate from a variety of sources, such as email, HTML files, unstructured text as well as structured databases. These sources make the Web a dynamic and heterogeneous environment, in which interpretation of information is difficult and error prone [1]. Much research has been undertaken to provide an integrated view of the Web by using a computerized approach. However the identification, querying and merging of data from heterogeneous sources is difficult.

A considerable amount of information available on the Web today is semistructured [2]. Semistructured data can be defined as data that has structure that may be irregular and incomplete and need not conform to a fixed schema. There has been a lot of research in the past in developing data models, query languages and systems to manage semistructured data. One such model is the Object Exchange Model (OEM) that was explicitly defined to represent semistructured data in heterogeneous systems in the Tsimmis system [3]. A variant of this data model has been used in the development of Lore [4]. The recent emergence of the Extensible Markup Language (XML) from the World Wide Web Consortium [5] has kindled a lot of interest in using it to model semistructured data [6-7]. XML is well suited to model semistructured data because it makes no restrictions on the tags and relationships used to represent the data. XML also provides advanced features to model constraints on the data, using an XML schema or a Document Type Definition (DTD). However, XML does have some differences with the other semistructured data models: (1) XML has ordered collections while semistructured data are unordered, (2) Attributes in XML can be unordered and (3) XML allows usage of references to associate unique identifiers for elements; this is absent in most other data models. Despite these differences, XML is a popular data model to represent semistructured data, mainly due to the close relationship to HTML as well as the emergence of standards and tools for creating and viewing XML. However, to the best of our knowledge not much progress has been made in the development of techniques and tools for storing and managing XML for rapid querying.

1.2. Goals of This Research

The goal of the thesis is to analyze the problems of XML data management and implement a toolkit that can be used to provide a persistent storage, retrieval and query component for XML data. We have developed such a toolkit as part of the Warehouse Manager (WHM) component in the IWiz prototype system in the Database and Research Center, University of Florida [8].

We rephrase the overall problem statement for this thesis as follows: Given the need to manage semistructured data in general and XML data in particular we need a system for managing this data efficiently. There are a wide variety of management systems, ranging from native XML databases to XML-enabled databases. Among the alternatives, we found it very compelling to choose the relational DBMS because of its wide spread popularity, robustness and performance. Since relational databases are already used to store information for most web sites and since XML is becoming the standard to represent this information, it is of the utmost importance that these two technologies be integrated [9]. So, in our system we have an underlying relational database for storing XML data and an interface to transform XML data to relational and vice-versa. Several major database vendors like Oracle are working on tools for managing XML data. We have summarized the limitations of these products in the related research section.

1.2.1. Challenges

To address the problem raised above, we have identified the following three challenges. (1) Automatic creation of the underlying relational schema based on the schema for the XML data that must be managed. This problem is further complicated when using DTDs to specify the structure of XML data; DTDs provide only a loose description of the structure of an XML document and does not contain any type information. (2) The loading of a single XML document into an equivalent relational schema may trigger the insertion of tuples into several tables. (3) Creation of a well-structured XML document with nested tags requires additional input and processing [10]. Existing methods in converting relational results into equivalent XML documents, use simple techniques where by the resulting document has tags derived from the metadata and values from the relational results. XML is a constantly evolving data model. Thus the solution to XML data management is not permanent and needs to be enhanced with the progress made in related fields like new query languages, more persistent storage options and new grammar definitions like XML Schema.

1.2.2. Contributions

Upon the conclusion of this research we will have contributed to the state-of-the-art in XML data management in several important ways. (1) Automatic schema generation: XML uses hierarchical representation of data. This native nesting in XML has to be translated to the relational schema that is flat in structure. The schema created has to preserve the relationships expressed in XML and map them to relational constraints. (2) Loading of XML data into a relational data warehouse: The loading operation will have to adhere to the constraints in the relational schema. The data in the XML data could contain extraneous characters like quotation marks that need to be removed before loading into the relational tables. (3) Automatic creation of nested XML documents: A structured XML document has to be recreated from the relational data obtained as a result of a SQL query. To achieve nesting in the created XML document would involve additional processing.

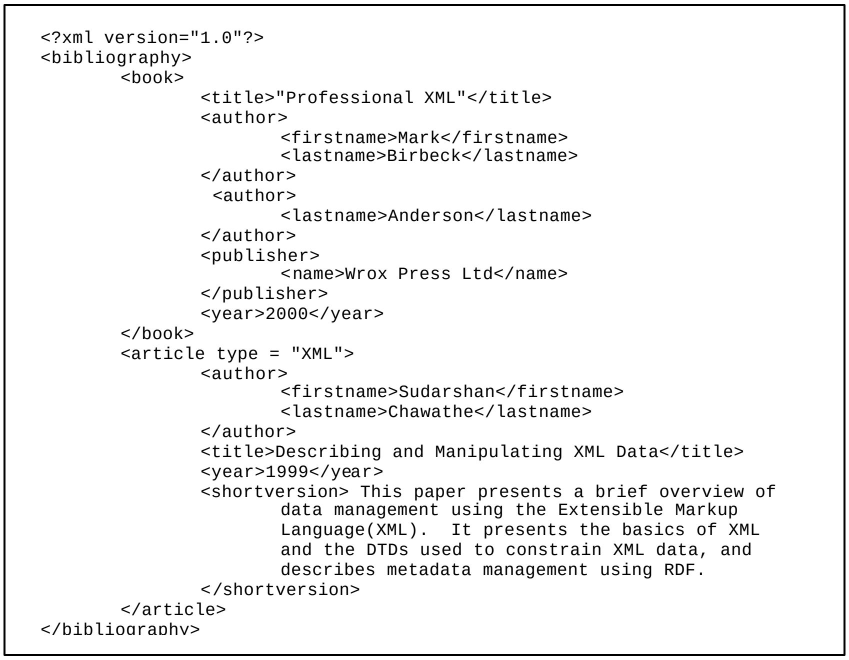
The rest of the thesis is composed as follows. Chapter 2 provides an overview of XML and related technologies. Chapter 3 describes the IWiz architecture and in particular the warehouse manager component. Chapter 4 concentrates on our implementation of the XML toolkit and its integration in the IWiz system. Chapter 5 performs an analysis of the implementation, and Chapter 6 concludes the thesis with the summary of our accomplishments and issues to be considered in future releases.

CHAPTER 2
RELATED RESEARCH

2.1. XML

Among the various representations to model semistructured data, XML has clearly emerged as the frontrunner. XML started as a language to represent hierarchical semantics of text data, but is now enriched with extensive APIs, tools such as parsers, and presentation mechanisms, making it into an ideal data model for semistructured data. XML consists of a set of tags and declarations, but rather than being concerned with formatting information like HTML, it focuses on the data and its relations to other data. Some important features of XML that are making it popular are the following [11]:

- XML is a plain ASCII text file making it platform independent.
- XML is self-describing: Each data element has a descriptive tag. Using these tags, the document structure can be extracted without knowledge of the domain or a document description.
- XML is extensible by allowing the creation of new tags. This supports new customized applications such as MathML, ChemicalML, etc.
- XML can represent relationships between concepts and maintain them in a hierarchical fashion.
- XML allows recursive definitions, as well as multiple occurrences of an element.
- The structure of an XML document can be described using DTD or XML schema.



```
<?xml version="1.0"?>
<bibliography>
        <book>
                <title>"Professional XML"</title>
                <author>
                        <firstname>Mark</firstname>
                        <lastname>Birbeck</lastname>
                </author>
                 <author>
                        <lastname>Anderson</lastname>
                </author>
                <publisher>
                        <name>Wrox Press Ltd</name>
                </publisher>
                <year>2000</year>
        </book>
        <article type = "XML">
                <author>
                        <firstname>Sudarshan</firstname>
                        <lastname>Chawathe</lastname>
                </author>
                <title>Describing and Manipulating XML Data</title>
                <year>1999</year>
                <shortversion> This paper presents a brief overview of
                        data management using the Extensible Markup
                        Language(XML).  It presents the basics of XML
                        and the DTDs used to constrain XML data, and
                        describes metadata management using RDF.
                </shortversion>
        </article>
</bibliography>
```

Figure 2.1: Example of an XML document.

2.1.1. Basics

The Extensible Markup Language (XML) is a subset of SGML [12]. XML is a markup language. Markup tags can convey semantics of the data included between the tags, special processing instructions for applications and references to other data elements either internal or external; nested markup, in the form of tags, describes the structure of an XML document.

The XML document in Figure 2.1 illustrates a set of bibliographic information consisting of books and articles, each with its own specific structure. Tags define the semantic information and the data is enclosed between them. For example in Figure 2.1, `<year>` represents the tag information and "`2000`" denotes the data value.

The fundamental structure composing an XML document is the *element*. A document has a root element that can contain other elements. Elements can contain character data and auxiliary structures or they can be empty. All XML data must be contained within elements. Examples of elements in Figure 2.1 are `<bibliography>`, `<title>` and `<lastname>`. Attributes can be used to represent simple information about elements, which are name-value pairs attached to an element. Attributes are often used to store the element's metadata. Attributes are not allowed to be nested, they can be only be simple character strings. The element `<article>` in our example has an attribute "`type`" with an associated data value "`XML.`"

2.1.2. DTDs

To specify the structure and permissible values in XML documents, a Document Type Definition (DTD) is used. Thus the DTD in XML is very similar to a schema in a relational database. It describes a formal grammar for the XML document. Elements are defined using the `<!ELEMENT>` tag, attributes are defined using the `<!ATTLIST>` tag.

```
<?xml version="1.0"?>
<!DOCTYPE bibliography [
<!ELEMENT bibliography (book|article)*>
<!ELEMENT book (title, author+, editor?, publisher?, year)>
<!ELEMENT article (author+, title, year ,(shortversion|longversion)?)>
<!ATTLIST article type CDATA #REQUIRED
                 month CDATA #IMPLIED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (firstname?, lastname)>
<!ELEMENT editor (#PCDATA)>
<!ELEMENT publisher (name, address?)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT shortversion (#PCDATA)>
<!ELEMENT longversion (#PCDATA)>
]>
```

Figure 2.2: A sample DTD representing bibliographic information

When a well-formed XML document conforms to a DTD, the document is called *valid* with respect to that DTD. Figure 2.2 presents a DTD that can be used to validate the XML document in Figure 2.1.

The DTD can also be used to specify the cardinality of the elements. The following explicit cardinality operators are available: “?” stands for "zero-or-one," “*” for "zero-or-more" and “+” for "one-or-more." The default cardinality of one is assumed when none of these operators are used. The operator “|” between elements is used to denote the appearance of one of the elements in the document. In our example in Figure 2.1, a `book` can contain one or more `author` child elements, must have a child element named `title`, and the `publisher` information can be missing. Order is an important consideration in XML documents; the child elements in the document must be present in the order specified in the DTD for this document. For example, a `book` element with a `year` child element as the first child will not be considered a part of a valid XML document conforming to the DTD in Figure 2.2.

The entire DTD structure can be placed in the beginning of the associated XML document or in a separate location, in which case the document contains only a `<!DOCTYPE>` tag followed by the root element name and the location of the DTD file in form of a URI. Separation of a schema and data permits multiple XML documents to refer to the same DTD.

At the moment of writing, a DTD is the only officially approved mechanism to express and restrict the structure of XML documents. There are obvious drawbacks to DTDs. Their syntax is different from the XML syntax (this is one reason why most parsers do not provide programmatical access to DTD structure). In addition, DTDs do not provide any inherent support for datatypes or inheritance. Finally, the format of cardinality declarations permits only coarse-grained specifications.

```
<schema ...>
      <element name = "bibliography"
                type = "string"
                minOccurs = "0"
                maxOccurs = "unbounded">
             <type>
                    <group order = choice>
                          <element type = "book">
                          ...
                          </element>
                          <element type = "article">
                                 <attribute name = "type" type ="string">
                                 <attribute name = "month"
                                            type =  "integer"
                                            default = "1">
                          ...
                          </element>
                    </group>
             </type>
      </element>
</schema>
```

Figure 2.3: An XML Schema representing the bibliographic information in the sample DTD.

W3C has recognized these existing problems with DTDs and has been working on new specifications called XML Schema since 1999 [13-14]. In March 2001, XML schema has been advanced to the proposed recommendation status. Eventually, this new data definition mechanism will have features like strong typing and support for data types. Proposed data types include types currently present in XML 1.0 and additional data types such as boolean, float, double, integer, URI and date types. In future systems, XML schema will provide a better integration of XML and existing persistent storage data models.

2.1.3. APIs for Processing XML Documents

The two alternative ways to access contents of an XML document from a program are the tree-based approach and the event-based approach. In the tree-based approach, an internal tree structure is created that contains the entire XML document in memory. An application program can now freely manipulate any part of the document. In case of the event-based approach, an XML document is scanned, and the programmer is notified about any significant events such as start or end of a particular tag that are encountered during scanning. The realizations of these approaches that have gained widespread popularity are the Document Object Model (implementing the tree-based model) and the Simple API for XML (in case of the event-based model).

The Document Object Model (DOM) specifications are produced by W3C like most of the XML-related technologies. The DOM Level 1 Recommendation dates back to October 1, 1998 [15]. The W3C has also come up with a Level 2 Recommendation for the DOM model [16]. DOM is a language- and platform-neutral definition and specifies the APIs for the objects participating in the tree model.

The Simple API for XML (SAX) represents a different approach to parsing XML documents. A SAX parser does not create a data structure for the parsed XML file. Instead, a SAX parser gives the programmer the freedom to interpret the information from the parser, as it becomes available. The parser notifies the program when a document starts and ends, an element starts and ends and when a text portion of a document starts. The programmer is free to build his/her own data structure for the information encountered or to process the information in some other ways.

As we have seen, both approaches have their own benefits and drawbacks. The decision to use one or the other should be based on a thorough assessment of application and system requirements.

2.2. XML Query Languages

The W3 consortium is in the process of standardizing a query language for XML based on the XML query algebra. From the semistructured community, three languages have emerged for querying XML data: XML-QL [17], YATL [18] and Lorel [19]. The document processing community has developed XQL [20], which is more suitable for querying documents and searching for text. For the IWiz system, we use an implementation of XML-QL by AT&T Labs. The following section discusses the syntax and features provided by the XML-QL language.

XML-QL has several notable features [21]. It can extract data from the existing XML documents and construct new documents. XML-QL is "relational complete"; i.e., it can express joins. Also, database techniques for query optimization, cost estimation and query rewriting could be extended to XML-QL. Transformation of data from one DTD to a different DTD can be easily achieved. Finally, it can be used for integration of multiple XML data sources.

In XML-QL, all the conditions are specified using a <WHERE> clause and the format of the resulting document is obtained from the <CONSTRUCT> clause. The structure specified in the <WHERE> clause must conform to the structure of the XML document that is queried. Tag-elements are bound using the "$" symbol to distinguish them from string literals and can be used in the <CONSTRUCT> clause or in conditional filters. Join conditions can be specified implicitly or explicitly. New tags can be created in the resulting document by using them in the <CONSTRUCT> clause. XML-QL uses element patterns to match data in an XML document, using the structure in the <WHERE> clause. There is considerable amount of similarity between XML-QL and other query languages. In particular, considering SQL, one can notice that the "WHERE" clause specifying the condition in SQL has the same functionality as the <WHERE> clause in XML-QL. Just like "AS" can be used to rename results in SQL, the <CONSTRUCT> clause can be used to create new tags and rename results. The XML document specified using the "IN" clause in XML-QL is like the set of tables represented using the "FROM" clause in SQL.

2.3. Data Warehousing

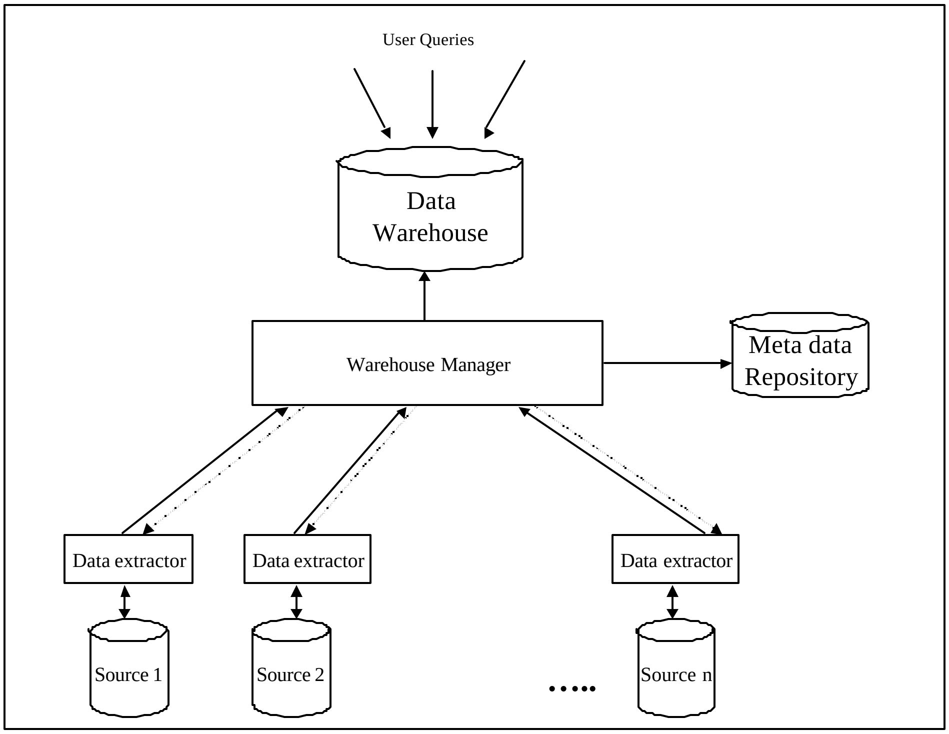


Figure 2.4: Generic warehousing architecture

Another technology related to this research is data warehousing. A data warehouse is a repository of integrated information from distributed, autonomous and possibly heterogeneous, sources. In the case of data warehousing systems [22-23], the warehouse manager loads and maintains the data warehouse, which is usually, a relational database, in advance using the metadata repository. Figure 2.4 shows the generic architecture of such a system. This is also commonly referred to as the *eager* or *in-advance* approach to data integration. Each source has a data extractor wrapped around it. This data warehouse is then queried and results are returned to the user. It represents a large volume of data that is stored in a single repository. The data warehouse can be optimized for storage depending on the transactions. Usually most of the loading into the data warehouse will involve appending of new information and fewer updates.

Thus the data warehouse serves as a cache with high query performance. The toolkit uses the data warehousing approach. Some of the key issues in this approach are the schema creation of the data warehouse, data loading and maintenance.

2.4. Mapping DTDs into Relational Schemas

The input queries and maintenance of data influence the schema for the data warehouse. The schema creation plays a big role in the efficiency of the data warehouse. The stored data can then be mined for information. Some of the algorithms for schema creation are as follows:

Edge Approach [24]: The XML document is viewed as a graph with no distinction between attributes and subelements. A table "*Edge*" is created with this schema (edge source, ordinal, name, flag, target). This table stores the tag information of the XML document. A separate "*Values*" table is created that has the schema as (target, value) to store the data contained in the XML document. This method proposes a simple scheme for translating an XML document to a relational table but is inefficient due to the redundancy in the schema creation.

Basic-Inlining [25]: Every element in the DTD is mapped to a relation table and elements mapped to a separate table inline as many of their descendants into the same table. In such a scheme, a particular element may be present in several tables. While loading the data element from an XML document, several tables have to be loaded with the data value. Also, in this schema creation scheme, a simple query would require several join conditions. Due to these inefficiencies, this approach is not suitable.

Shared-Inlining [25]: This approach tries to solve the problems in the "*Basic-Inlining*" approach by sharing relational tables. The principal idea in this method is to create a DTD graph and create separate relation tables for nodes that have an in-degree either equal to zero or greater than one. Elements being involved in a one-many relationship, which can be known by the presence of a “*” or a “+ are also mapped to a separate table. Thus in this scheme, a particular data item will be loaded only into a single relational table. But this scheme may not be appropriate when you consider data maintenance for the following reason. Since data elements have been inlined, when maintenance queries are generated, there is an overlap of concepts from the XML domain. Thus a table representing book information could have possibly a field for the author’s name. Hence, this approach is not used.

Hybrid-Inlining [25]: This method is a slight modification of the *“Shared-Inlining”* approach. In this scheme, elements with degree greater than one are also inlined as relational attributes in the table created for the parent. This does reduce the join conditions but has similar maintenance problems as *“Shared-Inlining”* scheme.

Our approach uses inlining of child nodes that do not have children or attributes, without considering the degree of the node. As relational tables contain only related fields, such a relational mapping provides more clarity to the system.

All the constraints expressed explicitly and derivable have to be translated to the data warehouse schema [26]. The data warehouse schema creation algorithm can have additional features to incorporate incremental changes to the schema of the underlying sources.

2.5. Data Loading and Maintenance

An important concern in using the warehousing approach is dealing with updates. The warehouse data has to be refreshed so that it can be consistent with the sources. Some simple techniques propose that the system goes off-line so that the entire warehouse can be refreshed with new data. This is obviously very inefficient for large data warehouses. There are other algorithms proposed to detect changes in the sources, escalate them to the integrator, which reflects it in the data warehouse [27-28]. Some of the loading schemes are as follows:

In the *load-append* strategy, the input data is loaded into the various relational tables without checking if the same data actually exists in the data warehouse. This is a simple technique but redundant tuples can be created. This is just like the "insert" operation in relational databases.

The *load-merge* strategy involves merging of the input data along with the existing data in the warehouse minimizing the redundancy. This operation is comparable to the "update" operation in relational databases.

In the *load-erase* strategy the content of the data warehouse is removed and then loaded with the incoming data. Thus the older content is totally removed and fresh data is loaded. Such an operation could be useless if the incoming data set is much lesser than the contents of the data warehouse. The operation is analogous to a "delete" operation followed by an "insert" operation.

2.6. XML Management Systems

There are several commercial products available to manipulate XML such as XML parsers, XML editors and other tools. We will briefly highlight the features of a few commercially available and research oriented XML management systems; thus, laying a foundation to the set of functions we want to provide in our toolkit.

2.6.1. Oracle XSU

The Oracle XML-SQL Utility (XSU) is an XML application that can be used for XML content and data management. The underlying persistent storage could be an object-relational or a relational database like Oracle 8*i*. XML data is stored as LOB (Large unstructured object) in relational tables and XML documents are stored as CLOB (Character Large Objects). XSU can also be accessed from a servlet. Some of the features are as follows [29]:

- Oracle XSU can generate an XML document from SQL results.
- It can store SQL results from XML inserts, updates and deletes.
- There are three different interfaces provided to access XSU: command line front end, Java API and PL/SQL API.

However, there are a few shortcomings that need to be addressed. The database schema has to be defined manually. The data loading assumes that the elements and attributes in the document are columns in a single table. To load multiple tables, the input document has to be translated into several documents, using XSL or any other language, and individually loaded into the various tables. Also, the data in the document has to be stored in child nodes and not as attributes. In the XML document generated, tags are not automatically nested.

2.6.2. GMD-IPSI XQL Engine

This is a Java based storage and query application that uses two major technologies: persistent implementation of the DOM objects and XQL language [30]. Some of the features are as follows:

- XML documents are parsed once and stored as persistent DOM (PDOM).
- The implementation can swap DOM nodes to disk, while handling large DOM trees and hence main memory is not a limit to file size.
- There are built-in caching and garbage collection mechanisms.
- Multi-threaded access of the PDOM file is allowed.

However the implementation has limitations. A PDOM structure is created for each incoming XML document. The XML document to be queried has to be specified. An update operation would increase the PDOM file size and requires a de-fragmentation operation to be initiated. Similarly, a delete operation creates wasted space in the file, which has to be reclaimed using the garbage collection operation.

2.6.3. LORE

LORE is a DBMS for managing semistructured data developed in the Stanford University. It was initially developed for the OEM data model to manage semistructured data but later migrated to XML. A few of the mention-able features are as follows:

A query language LOREL with a cost-based optimizer is used. The prototype is complete with indexing techniques, multi-user support, logging and recovery. Dataguides, a structural summary of all database paths, is generated; thus allowing free form input data.

As mentioned in the web site, LORE needs some more development in the areas of storage schemes and comparison operators. The LORE system currently does not use DTDs and does not encode sub-elements ordering.

Two of the other prominent management systems that are available are Microsoft SQL Server [31] and Strudel Web site management system [32].

CHAPTER 3
THE IWIZ PROJECT

The XML toolkit is an integral part of the IWiz system. IWiz has the following main modules as shown in the Figure 3.1: A Query Browsing Interface that serves as the user interface. The Warehouse Manager as shown in the top tier of the architecture maintains the data warehouse. A Mediator rewrites the user query to source-specific terms and resolves conflicts in the returned results. Each source is connected to IWiz via a wrapper (DRE) for querying the data and restructuring the results. The interactions between the various modules are as shown in Figure 3.1. A brief description of the modules in the IWiz prototype with their functionality and inputs is provided below.

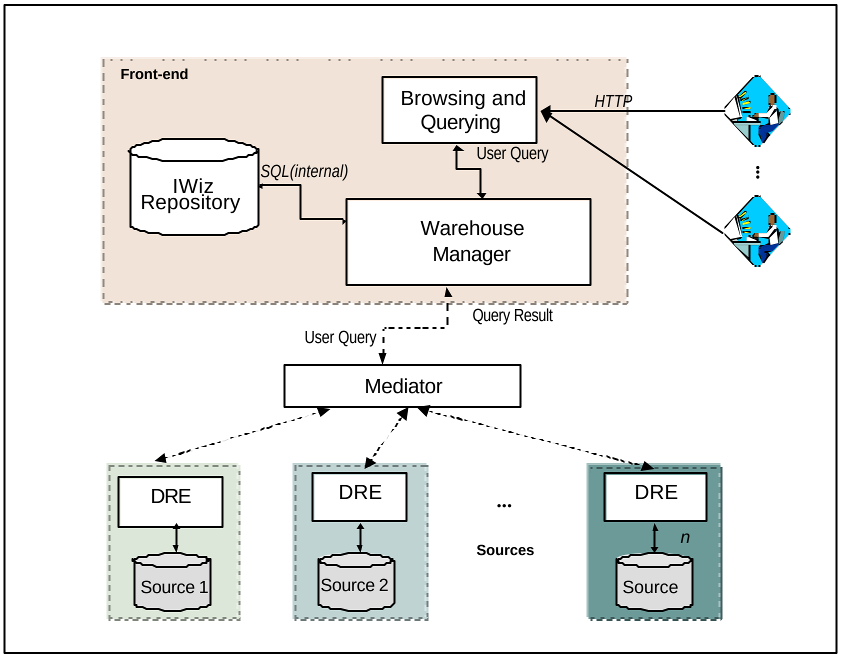


Figure 3.1: IWiz Architecture

The Query Browsing interface (QBI) presents an integrated view of the IWiz global schema. The QBI is used to generate a user query, which is then sent to the warehouse manager component.

The Warehouse Manager (WHM) component, as shown in Figure 3.2, maintains the IWiz Repository which is an Oracle 8i database. The WHM has two major phases of operation: a *built-time phase* during which the schema creator module of the XML toolkit creates the script file to create the relational schema for the data warehouse using the DTD description of the global IWiz schema as input. The Database Connection Engine (DBCE) executes the script file to complete the schema generation process. This is followed by the *run-time* or *query phase* during which WHM accepts and processes XML-QL queries against the underlying relational database; the same query may also be sent to the mediator in case the contents of the data warehouse are not up-to-date.

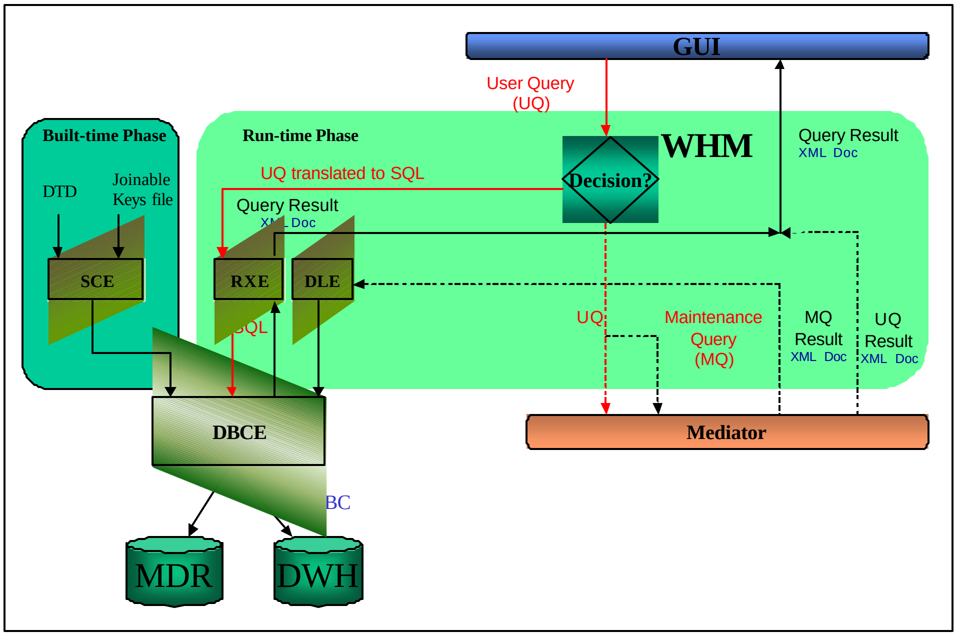


Figure 3.2: WHM Architecture

The WHM server also provides the Ontology contents to the Mediator and the Wrapper. When the XML-QL query is provided to the WHM, the query is checked if it could be satisfied from the contents of the data warehouse. The XML-QL is then translated to SQL and executed against the warehouse by the DBCE. The XML-QL to SQL conversion is part of another ongoing research project in the Database Research and Development Center at the University of Florida. The Relational-to-XML-Engine component of the toolkit could be used to translate the relational result set to an XML document. In case the user query cannot be satisfied from the contents of the data warehouse, the XML-QL query and a maintenance query are sent to the Mediator component. After the Data Merge Engine returns the resulting document generated by merging the information from the various sources, the WHM presents it to the QBI that displays it to the user. The XML Loader (DLE) is invoked if the merged document is an effect of a maintenance query. The DLE parses the document and generates the insert commands. The DBCE is invoked and the data is loaded into the warehouse. All the interactions with the data warehouse are interfaced through the DBCE.

The Mediator component has two modules: the Query Restructuring Engine (QRE) and the Data Merge Engine (DME). The QRE during the *built-time* phase gets the knowledge about the data existing in the various data sources. The QRE, using this information, then splits the input query into source specific queries in terms of the global schema terms appending the source names to the query id. It also generates the query plan that is used to merge the data from the sources by the DME. The DME merges the results from the various sources removing duplicates and transforms them to a single XML document. The DME returns the merged document to the WHM.

Each source in the system has a Data Restructuring Engine (DRE) wrapped around it. The DRE is responsible for translating the input query in terms of the global schema terms to the source specific terms and converting the source data returned from the sources to the global schema terms. The DRE returns the results from each source to the DME.

CHAPTER 4
XML TOOLKIT: ARCHITECTURE AND IMPLEMENTATION

This chapter discusses about the relational approach, the advantages and dis-advantages of this approach, the architecture and implementation details of the XML toolkit. It describes the algorithm implemented in the various modules of the toolkit. The toolkit is implemented using Java (SDK 1.3) from Sun Microsystems. Some of the other software tools and packages used in the implementation are the XML Parser from Oracle version 2.0.2.9, Oracle 8*i* and the Oracle JDBC driver version 2.

4.1. Managing XML Data in IWiz

The popularity of XML as a new standard for data representation and exchange on the Web necessitates the development of an XML management system. XML management systems can be broadly classified as XML document management systems and XML data management systems [33]. In the former case, the structure is very irregular and is usually difficult for a machine to interpret the data. Some of the examples

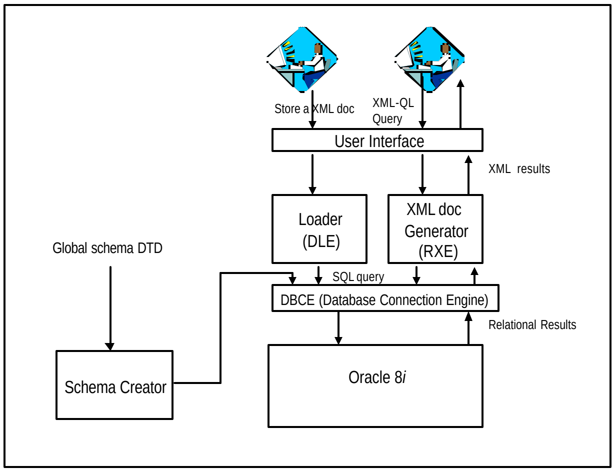


Figure 4.1: Proposed Architecture of XML data management in IWiz.

could be advertisements and HTML documents. Systems to handle XML documents are also known as content management systems [34]. The focus of this approach is however on XML data management. In this data-centric approach XML is used as the representation format. Many of the documents created in real-world applications such as flight schedules and sales order are examples of this classification. The XML data management system should be able to provide a persistent store for XML data using a relational, object database or an object relational database. The system should be able to provide an interface to transfer data between the database and XML. All the data can be stored in a single centralized repository, thus having controlled redundancy. The data can be queried and a merged view can be created.

The conceptual architecture for our proposed XML data management system is shown in Figure 4.1. If a global schema exists it can be used to create the schema for the persistent store. An additional feature of such a system will be to automate schema creation for the persistent storage. There is usually a user interface that is used to either load an XML document into the data store or query the stored data. The loader component stores the data contained in the incoming document in the underlying data store. The data maintenance is generally built as part of the loader. The XML document generator module accepts the user query, transforms it to the language of the underlying data store, executes it and then formats the relational results as an XML document and presents it to the user.

4.2. Rational for Using an RDBMS as Our Storage Management

In this toolkit, the persistent storage is achieved using the relational Oracle 8*i* database. While there are systems that use other techniques such as using semi-structured data stores, for example in Lore, the question of whether which of them is the best approach remains. The downside of using these other techniques is that they turn their back on several years of work invested in relational database technology. When semi-structured data becomes more widely popular and is machine processed, the management systems will require efficient query processing and storage features. Relational systems currently are the best in providing these features. Specifically, they have the following advantages:

- Centralized merged data: The data warehouse is a single repository containing the data due to several loaded documents and is not specific to any input XML document.
- Scalability: With the increase in contents RDBMS provide one of the best scaling.
- Standard query languages: Using worldwide accepted query languages with efficient querying capabilities.
- Concurrency Control, Data recovery and management of secondary storage features.

On the other hand, using a relational database management system has the following drawbacks:

- In order to satisfy a query, the join sequences could be an n-way join leading to an inefficient execution.
- An RDBMD requires a rigid schema definition.

4.3. Functional Specifications

Next we proceed to derive the set of functional specifications that the toolkit must satisfy. Firstly, the toolkit must assist in the relational schema creation. Ideally this process must be automated. The schema creation should be derived from a global schema that binds the incoming XML documents. For example, a DTD can be used to describe the global schema definition. The relational schema created must include the cardinality constraints specified in the DTD. There should be an interface to accept an XML document as input and load the data contained into the various relational tables. Similarly, the toolkit should have the capability to execute queries and wrap the results as XML documents. Nesting of tags and creating XML documents conforming to the global schema would provide additional features.

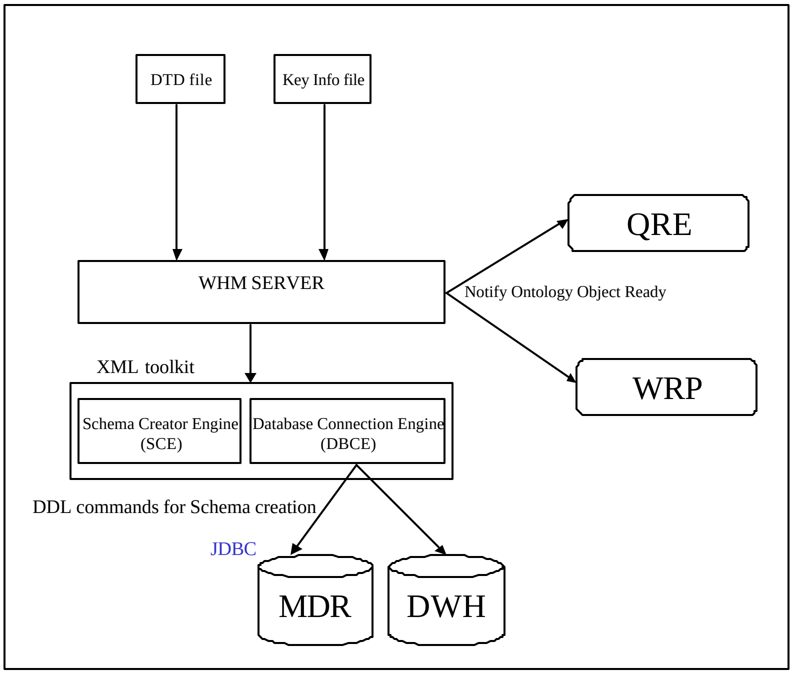

DTD file
Key Info file
QRE
WHM SERVER
Notify Ontology Object Ready
XML toolkit
WRP
Schema Creator Engine (SCE)
Database Connection Engine (DBCE)
DDL commands for Schema creation
JDBC
MDR
DWH

Figure 4.2: Built-time architecture of the XML toolkit

4.4. Architecture Overview

From the set of functional specifications above we can derive the architecture of the XML toolkit which is divided into a *built-time* and *run-time* phase. The *built-time* phase creates the preliminary steps- setting up the schema and the server so that in the WHM can accept and process user queries during the *run-time* phase.

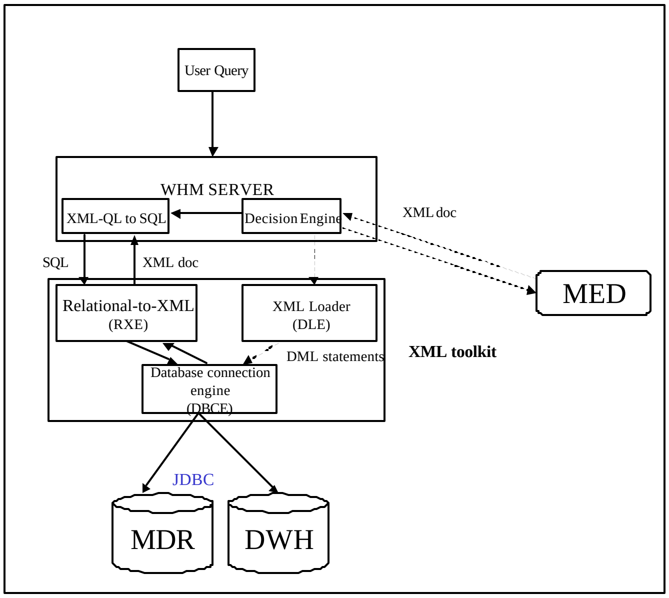


Figure 4.3: Run-time architecture of the XML toolkit

During the *built-time* phase of the WHM, as shown in Figure 4.2, the Schema creator engine (SCE) and the database connection engine (DBCE) of the toolkit are used. The input DTD file representing the IWiz global schema is parsed and the script file to generate the relational tables is created. The script file is then passed on to the DBCE, which executes it, and the relational schema is setup. The Joinable Keys file, which is the other input to this module, is stored in a separate system table. The WHM is implemented as a server that can accept XML-QL user queries, translate them to SQL, run them against the warehouse and return the results as an XML document to the user. This also serves as an Ontology server that provides the Ontology as a DOM object when invoked by the Mediator and the Wrapper. After parsing the DTD, the Ontology server notifies the Wrapper and the Mediator. After being notified, the wrapper and mediator request for the ontology DOM object to begin their *built-time* phase.

In the *run-time* phase, as shown in Figure 4.3, the Relational-to-XML-engine (RXE), XML data loader engine (DLE) and the DBCE components of the toolkit are used. The decision engine module of the WHM analyzes the input XML-QL user query. If the input query could be satisfied in the warehouse, the user query is translated into SQL by the XML-QL to SQL module of the WHM. The SQL query generated is then executed in the data warehouse using the DBCE of the toolkit. The relational results are then converted to an XML document object by the RXE and returned to the QBI interface. If the query cannot be satisfied in the data warehouse, due to the absence or staleness of data then a maintenance query is generated and both the user query and the maintenance query are sent to the mediator. The maintenance query generates results that are used to load the data warehouse so that in future, similar queries could be satisfied directly from the warehouse. The DLE component of the toolkit is used to load the data. The XML document generated due to the user query is returned to the QBI interface, which presents it to the user.

```
<!ELEMENT Bib (Book,Article )>
<!ELEMENT Book (Author+,Title,Year,Editor*,ISBN)>
<!ELEMENT Article (Author*,Title,Year,Editor?)>
<!ELEMENT Author (firstname?,lastname,address)>
<!ELEMENT Editor (lastname)>
<!ELEMENT Title ( #PCDATA) >
<!ELEMENT ISBN ( #PCDATA) >
<!ELEMENT Year (#PCDATA)>
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT address  (#PCDATA) >
```

Figure 4.4: Input DTD to the Schema creator engine (SCE)

4.5. Schema Creator Engine (SCE)

The input to this module is the DTD file and Joinable Keys file. The algorithm for schema creation views the DTD as a graph and nodes are distinguished based on the path from the parent node and not based on the tag names. For example, the paths of the “*Year*” element in Figure 4.4; “*Bib/Article/Year*” and “*Bib/Book/Year*” are different. The DTD file is parsed by the Oracle XML parser and an n-ary tree is created. The module begins traversing this tree beginning at the root node identifying the various ‘*CONCEPT*” nodes. An element in the DTD is a “*CONCEPT*” if it satisfies one or more of the following conditions:

- It has one or more attributes or
- It has one or more children or
- It is involved in a one-many relationship with some other element in the DTD, which can be inferred by the presence of the cardinality operators, “*” or “+”, following this node when it appears as a child node.

Every ‘*CONCEPT*” in the DTD is mapped to a relational table and all child elements that are not “*CONCEPT*”(leaf), are mapped as relational attributes in the table. A hash table containing type-information about each element and attribute is created while processing the DTD. The type can be a ‘*CONCEPT*”, ‘*Inlined-Attribute*” or a ‘*Inlined-Child*”. This information is also persistently stored in a system table, which is later retrieved and used by the loader module. The script to create a relational table with the same name as the “*CONCEPT*” node is generated and appended to a global script file. Every table created has a primary key. The name of every table created is stored in a hash table. A table is created only if it is absent in this hash table. This process is recursively performed on all the children of this element passing the Parent-element name. All the parent-child relationships between the elements in the DTD are mapped as a general m:n relation and a separate table is created to store this information. The relation tables contain two fields with reference to the corresponding field in the parent and child table. The format of naming the table is "<Parent-element> _ <Child_element>". The foreign key references are appended to the global script file. This script file is finally executed, using the DBCE, creating the various tables and the foreign key constraints. The other input to this module is the Joinable Keys file. This is used by the QRE module of the mediator to detect join sequences for the various elements in the global Ontology DTD.

```
Bib.Book      \t        1        \t        ISBN
Bib.Book      \t        2        \t        Title
Bib.Article   \t        1        \t        Title
```

Figure 4.5: Joinable Keys file format

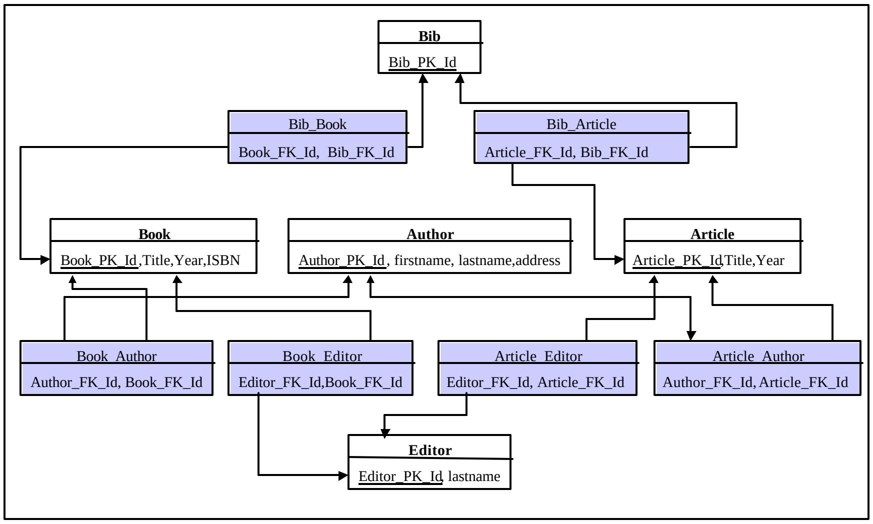
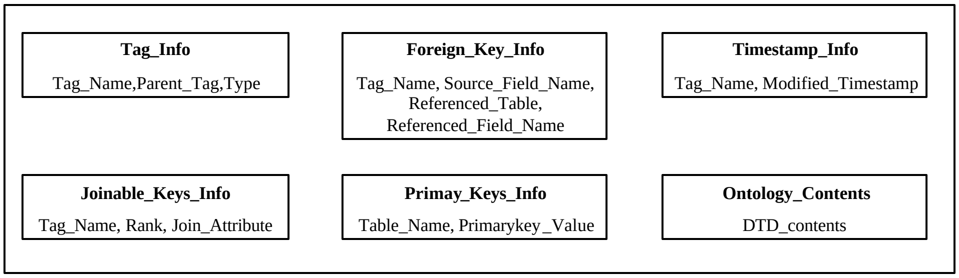


Figure 4.6: Tables created by the SCE for the input DTD in Figure 4.4.

The format of this file is as shown in Figure 4.5. It contains the path and joinable key for the "*CONCEPT*" elements in the DTD.



Tag_Info
Tag_Name,Parent_Tag,Type

Foreign_Key_Info
Tag_Name, Source_Field_Name, Referenced_Table, Referenced_Field_Name

Timestamp_Info
Tag_Name, Modified_Timestamp

Joinable_Keys_Info
Tag_Name, Rank, Join_Attribute

Primay_Keys_Info
Table_Name, Primarykey_Value

Ontology_Contents
DTD_contents

Figure 4.7: System tables created by the SCE.

Figure 4.6 gives the set of tables created for the DTD in Figure 4.4. The system tables, as shown in Figure 4.7, are also created by the SCE. The Ontology file contents are stored in the “Ontology_Contents” system table. It is later retrieved, parsed into a DTD object and provided to the QRE and the WRP modules. The Joinable Keys file is stored in the “Joinable_Keys_Info” system table. The “Primary_Keys_Info” and the “Tag_Info” are used by the loader module.

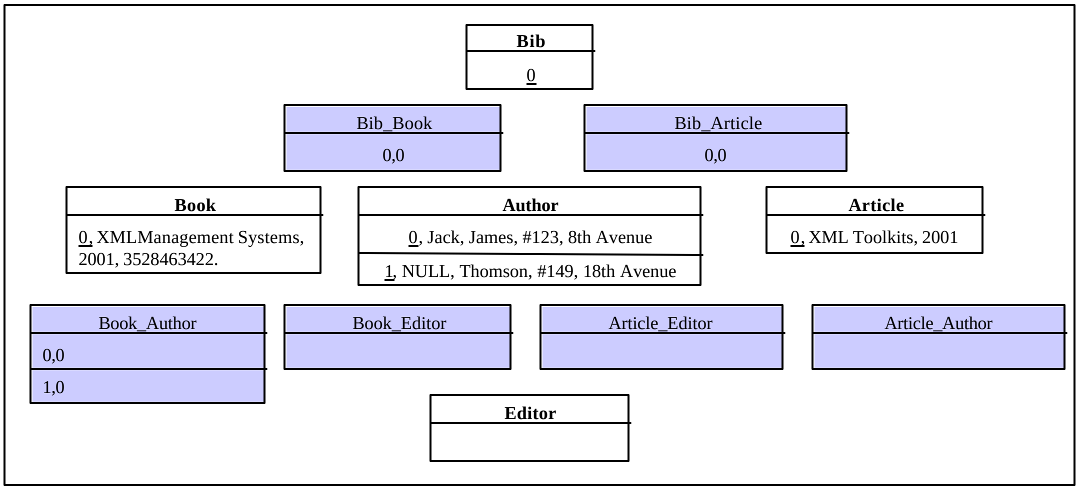
```
1. The input DTD file is parsed and a DTD DOM object is created.
2. Invoke method makeRelations (Root,”-”) passing the root of the DTD and ‘-’ to denote that it has no
   parent..
3. A node is a CONCEPT if  one of the following conditions holds:
    3.1. It has 1 or more attributes.
    3.2. It has 1 or more children nodes.
    3.3. It has a ‘*’ or a ‘+’ when it appears as a child of some other node.
4. Method makeRelations(Node current_node,String Parent)
   4.1 Check if the current node in the tree is a CONCEPT.
     4.1.1. Check if table is already created for the current node in the hash table.
         4.1.1.1. If false
            4.1.1.1.1  Generate the script file to create a table having with the current_node name and make
                 the attributes, and children of the current node which are not Concepts, fields in the table.
            4.1.1.1.2  Generate the script file to create the ‘Par-Child’ relational table using the Parent name.
            4.1.1.1.3.  Store the  table names for whom the script file are generated in a hash table
            4.1.1.1.4. Store the foreign key constraints in a separate vector.
   4.2. Recursively invoke the makeRelations method passing the children of the current node.
5. Append the foreign key constraints to the script file
6. Execute the script file to generate the ‘CONCEPT’ tables, the ‘Par-Child’ tables and the foreign key
constraints.
```

Figure 4.8: Pseudo code of the SCE

The decision engine module of the WHM uses the “Foreign_Keys_Info” and the “TimeStamp_Info” system tables.

```
<?xml version = "1.0"?>
<Bib>
       <Book>
              <Author>
                   <firstname> Jack </firstname>
                   <lastname> James</lastname>
                   <address> #123, 8th Avenue</address>
              </Author>
              <Author>
                   <lastname> Thomson</lastname>
                   <address> #149, 18th Avenue</address>
               </Author>
              <Title> XML Manangement Systems </Title>
              <Year> 2001 </Year>
             <ISBN> 3528463422  </ISBN>
     </Book>
     <Article>
             <Title> XML Toolkits </Title>
             <Year> 2001 </Year>
     </Article>
</Bib>
```

Figure 4.9: A sample XML document conforming to the input DTD in Figure 4.4.



Bib
0

Bib_Book
0,0

Bib_Article
0,0

Book
0, XMLManagement Systems, 2001, 3528463422.

Author
0, Jack, James, #123, 8th Avenue
1, NULL, Thomson, #149, 18th Avenue

Article
0, XML Toolkits, 2001

Book_Author
0,0
1,0

Book_Editor

Article_Editor

Article_Author

Editor

Figure 4.10: Contents of the tables after loading the sample XML document in Figure 4.9.

The pseudo-code of the algorithm is shown in Figure 4.8. The SCE also generates the script file for creating and dropping the database in the current directory. In case of any network error while accessing the database, the script file can be executed from the Oracle SQL*Plus window directly to create the relational tables.

4.6. XML Data Loader Engine(DLE)

The loading operation is a translation from one data model (XML) to the other (RDBMS). The data in the XML document is stored as relational tuples. The XML data loader module, also known as the loader, implements the load-append strategy. Figure 4.9 gives an example of an XML document conforming to the DTD shown in Figure 4.4. This module takes in an XML document DOM object as input and generates the script file to load the data into the relational tables. The loader uses the "Tag_Info" and the "Primary_Keys_Info" system tables. The data from both these tables is loaded into two hashtables, Tag_Info and PK_Vals, so that the database is accessed only once initially. The loader starts parsing the DOM object from the root until all the nodes in this n-ary tree are traversed. A node that is found to be a ‘*CONCEPT*” triggers loading of a tuple for the relational table with the same name. The schema information is obtained from the database. The tuple is initialized with null values. The attributes and children of this node are examined and the values are loaded into the tuple after matching the relational field name with the element name. The primary key value for this tuple is assigned from the PK_vals table. The children nodes are recursively processed passing the parent-name so that the foreign key references in the Parent-Child table are setup. The foreign key values for this table is obtained from the PK_vals table as well. The primary key value for the current table is updated in the Primary_Keys_Info hashtable only after processing all it's children. This script file and the PK_Vals hashtable is passed on to the database connection engine. The DBCE executes the script file to load the data contained in the

XML document DOM object into the database. The hashtable is used to update the contents of the Primary_Keys_Info system table so that if the system were to crash, then the values of the primary keys for the loading of the next document would begin with the correct values.

```
1. Input is a XML document DOM object.
2. Create the Tag_Info and PK_Vals hashtable from the system tables. Make a new hashtable
   Table_Names.
3. Invoke the method makeTuple (Root,'-')
4. Method makeTuple(Node current_node,String Parent)
  4.1 Check if the current node in the tree is a CONCEPT using the Tag_Info hash table.
    4.1.1.  If true
          4.1.1.1  Match the leaf children names with field names in the relational table and generate
                   the script to create a tuple using the PK_Vals hashtable.
          4.1.1.2  Generate the script file to create a tuple for the 'Parent_Child' table.
  4.2. Recursively invoke the makeTuple method for the children of the current node.
  4.3  Update the primary key value for this table in the PK_Vals hashtable.
5. Execute the script file to insert the data in the XML document into the relational tables and update the
   contents of the  Primary_Keys_Info system table.
```

Figure 4.11: Pseudo code of the loader

The tuples generated by the loader and stored in the schema for the document in Figure 4.5 are shown in Figure 4.10. The pseudo code for the loader module is in Figure 4.11.

4.7. Relational-to-XML-Engine (RXE)

The input to this module is a SQL query. The query is executed in the data warehouse and the results are wrapped into an XML document DOM object grouping the results if the SQL query contains the "Group By" clause. The module obtains the metadata information from the database. The module can create the XML document conforming to the ontology specifications if the paths of each of the resulting tags are provided. Initially the RXE looks for the "Group By" clause and stores the grouping attributes in a vector and a hashtable Grp_hash. The grouping attributes are located in the resultset and the values in the tuples are compared with the value stored in the Grp_hash hashtable. "GRP By" tags are created for each of them. The new values of the grouping attributes are updated in Grp_hash. Tags are created from the resultset meta data and are appended to the XML document DOM object. If the SQL query has no "Group By" clause, the resulting document created has the metadata name as tag name and the tuple value as the data value. The important feature of this module is the ability to create a document grouping the results. Every tuple mapped to the XML document contains a "ROW ID" attribute, which takes the tuple number as value.

```
SELECT Author.firstname, Author.lastname, Author.address, Book.Title, Book.Year,
Book.ISBN, Article.Title, Article.Year

FROM Bib,Article,Book, Bib_Article, Bib_Book,Author,Book_Author

WHERE Bib.Bib_PK_ID=Bib_Article.Bib_FK_ID and Bib.Bib_PK_ID=Bib_Book.Bib_FK_ID
and  Author.Author_PK_ID and Book_Author.Author_FK_ID;
```

Figure 4.12: SQL query to retrieve books and articles from the data warehouse.

```
<Result>
  <ROW ID="1">
    <FIRSTNAME>Jack</FIRSTNAME>
    <LASTNAME>James</LASTNAME>
    <ADDRESS>#123, 8th Avenue</ADDRESS>
    <TITLE>XML Manangement Systems</TITLE>
    <YEAR>2001</YEAR>
    <ISBN>3528463422</ISBN>
    <TITLE>XML Toolkits</TITLE>
    <YEAR>2001</YEAR>
  </ROW>
  <ROW ID="2">
    <FIRSTNAME>NULL</FIRSTNAME>
    <LASTNAME>Thomson</LASTNAME>
    <ADDRESS>#149, 18th Avenue</ADDRESS>
    <TITLE>XML Manangement Systems</TITLE>
    <YEAR>2001</YEAR>
    <ISBN>3528463422</ISBN>
    <TITLE>XML Toolkits</TITLE>
    <YEAR>2001</YEAR>
  </ROW>
</Result>
```

Figure 4.13: XML document generated by the Relational-to-XML-engine (RXE).

The SQL query would have to be generated based on the schema information that can be known from the system tables. The table name is appended to the field name while retrieving so that it resolves any conflicts that could occur when the same column exists in both the tables. A join between two concepts in the global schema can be achieved by involving the parent-child relational tables. For example for the SQL query in Figure 4.12, a join has to be performed using the Parent table, `Bib`, `Book`, `Article` and the parent-relational tables `Bib_Article` and `Bib_Book`. The query could include a "Group By" clause, but as per the rules of RDBMS require the inclusion of the non-aggregateable attributes present in the select clause, in the "Group By" clause as well. The document created could be nested if the path information about each tag created is presented to the RXE. The result of a sample SQL, given in Figure 4.12, to extract the loaded document in Figure 4.9 is shown in Figure 4.13.

```
1. Input is a SQL query.
2. Check if the query  has a 'GROUP BY' clause.
   2.1 If true
       2.1.1. Check for all the alias names used for the grouping attributes and store the attribute names in a
              hashtable, Grp_hash and a vector Grp_vec.
3. Execute the SQL query using the Database Connection Engine (DBCE)
   3.1. Obtain the resultset containing the results of the query.
   3.2. Obtain the metadata details of the resultset.
4. Method makeXML()
   4.1. For all the tuples in the resultset do:
        4.1.1. For all the grouping attributes in Grp_Vec.
               4.1.1.1. Locate the grouping attribute in the resultset.
               4.1.1.2. Obtain the value of the attribute from the hashtable.
               4.1.1.3. Compare the value of the grouping attribute in the hashtable and tuple; if they differ:
                        4.1.1.3.1. Create a 'GRP By' tag and assign the value to it from the tuple.
                        4.1.1.3.2. Store the new value in the hashtable
                        4.1.1.3.3. Reset the values of the grouping attributes lower in priority than the current attribute in
                                   Grp_hash.
               4.1.1.4. All the other values that are not a part of the grouping attributes are stored as separate tags
                        created from the metatdata details and appended to the XML document DOM object.
5. Return the XML document DOM object created.
```

Figure 4.14: Pseudo code of the RXE.

The pseudo code for the algorithm is given in Figure 4.14.

4.8. Database Connection Engine (DBCE)

This module provides the database connectivity. It uses Oracle JDBC driver version 2.0 to communicate with the database. It provides a set of API that can be invoked by the other modules. There is only one instance of the DBCE operating and the other modules have a reference to it. The DBCE uses a configuration file to obtain the database details: hostname, database name, port number, user name and password. The DBCE uses the standard JDBC API classes, "Statement" and "PreparedStatement" to initiate all connections with the Oracle 8*i* database engine. All insert operations are implemented as batch operations in order to increase the efficiency of the database engine. All the relational attributes have "`varchar2`" as the type due to inadequate information in the DTD. Also, the default field size is assumed beforehand.

CHAPTER 5
PERFORMANCE EVALUATION

In this chapter, we explain a set of tests to evaluate the performance of our toolkit. The only XML-processing programs that are benchmarked are several XML parsers [35]. Up to this point, there is very little material discussing the benchmarking of XML data management systems [36-37]. Among them, Xmach-1 provides the benchmarks based on web applications, which are not directly applicable to the toolkit. Hence, we intend to demonstrate in an informal way the validity, functionality and performance of the toolkit. The focus of test 1 is to analyze the capability of the SCE module to produce syntactically valid script to generate a relational schema. The goals of test 2 are to study the correctness and efficiency of the entire toolkit. To illustrate the efficiency, we draw a comparison between the outputs generated by the XML-QL processor, implemented by AT&T and the RXE component. Thus, the various aspects of management systems, schema creation, data loading and data extraction are tested. Section 5.1 describes the hardware configuration and software packages used for testing. In Section 5.2, we explain briefly the tests that were performed. Section 5.3 discusses in detail about the results, bringing out the limitations. The inputs and outputs to the various components are illustrated in the Figures 5.1-5.7.

5.1. Experimental Setup

All the experiments were carried out on a Pentium II 233 Mhz processor with 256 MB of main memory running Windows NT 4.0. The toolkit was implemented using Java

(SDK 1.3) from the Sun Microsystems. Some of the other software tools and packages used are the XML Parser from Oracle version 2.0.2.9, Oracle 8*i*, Oracle JDBC driver version 2 and the XML-QL query processor, implementation by AT&T. The DTDs and XML documents were created using XML Authority v1.2 and XML instance v 1.1 respectively. All the modules of the toolkit ran in the same address space as the database, which was installed on the same machine to avoid network delays.

```
<!ELEMENT TVSCHEDULE (CHANNEL+)>
<!ELEMENT CHANNEL (BANNER, DAY+)>
<!ELEMENT BANNER (#PCDATA)>
<!ELEMENT DAY ((DATE1, HOLIDAY) | (DATE1, PROGRAMSLOT+))+>
<!ELEMENT HOLIDAY (#PCDATA)>
<!ELEMENT DATE1 (#PCDATA)>
<!ELEMENT PROGRAMSLOT (TIME, PROG_TITLE, DESCRIPTION?)>
<!ELEMENT TIME (HRS,MINS)>
<!ELEMENT HRS (#PCDATA)>
<!ELEMENT MINS (#PCDATA)>
<!ELEMENT PROG_TITLE (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>
<!ATTLIST TVSCHEDULE NAME CDATA #IMPLIED >
<!ATTLIST CHANNEL CHAN CDATA #IMPLIED >
<!ATTLIST PROGRAMSLOT VTR CDATA #IMPLIED >
```

Figure 5.1: DTD describing the structure of a TV programs guide

TVSCHEDULE (TVSCHEDULE_PK_ID, NAME)

CHANNEL (CHANNEL_PK_ID, CHAN,BANNER)

DAY (DAY_PK_ID)

HOLIDAY(HOLIDAY_PK_ID, HOLIDAY)

DATE1(DATE1_PK_ID, DATE1)

PROGRAMSLOT(PROGRAMSLOT_PK_ID, VTR,PROG_TITLE, DESCRIPTION)

TIME(TIME_PK_ID, HRS, MINS)

TVSCHEDULE_CHANNEL(CHANNEL_FK_ID, TVSCHEDULE_FK_ID)

CHANNEL_DAY(DAY_FK_ID, CHANNEL_FK_ID)

DAY_DATE1(DATE1_FK_ID ,DAY_FK_ID)

DAY_HOLIDAY(HOLIDAY_FK_ID,DAY_FK_ID)

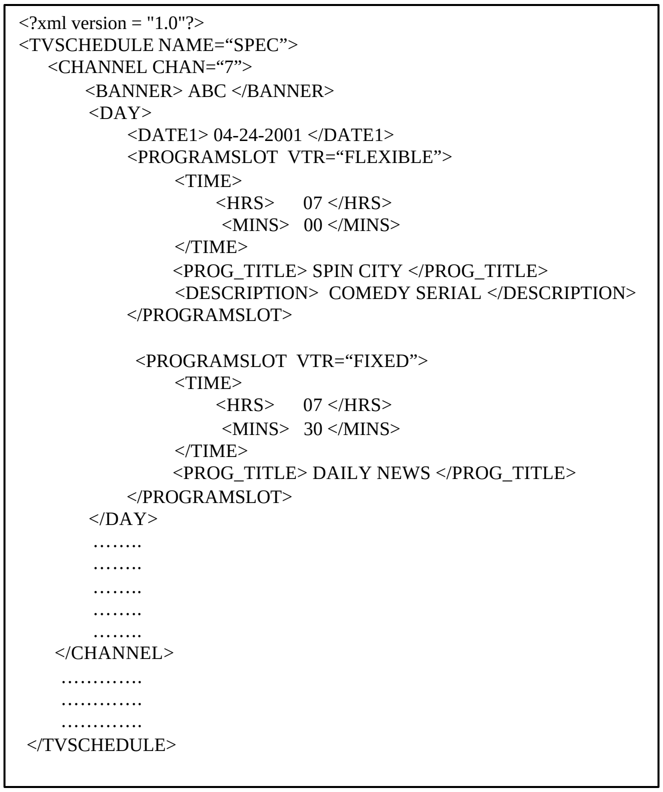
DAY_PROGRAMSLOT(PROGRAMSLOT_FK_ID,DAY_FK_ID)

PROGRAMSLOT_TIME(TIME_FK_ID, PROGRAMSLOT_FK_ID)

Figure 5.2: Tables created by the SCE for the TV programs guide DTD

5.2. Test Cases

Test 1 studies the ability of the SCE module to produce a valid relational schema. The input DTD is supplied to the SCE that creates the relational schema. The input DTD describes the structure of a TV programs guide. Figure 5.1 shows the input DTD. Figure 5.2 displays the relational schema that corresponds to the DTD.



```
<?xml version = "1.0"?>
<TVSCHEDULE NAME=“SPEC”>
    <CHANNEL CHAN=“7”>
         <BANNER> ABC </BANNER>
         <DAY>
              <DATE1> 04-24-2001 </DATE1>
              <PROGRAMSLOT  VTR=“FLEXIBLE”>
                    <TIME>
                         <HRS>     07 </HRS>
                          <MINS>   00 </MINS>
                    </TIME>
                    <PROG_TITLE> SPIN CITY </PROG_TITLE>
                    <DESCRIPTION>  COMEDY SERIAL </DESCRIPTION>
              </PROGRAMSLOT>

               <PROGRAMSLOT  VTR=“FIXED”>
                    <TIME>
                         <HRS>     07 </HRS>
                          <MINS>   30 </MINS>
                    </TIME>
                    <PROG_TITLE> DAILY NEWS </PROG_TITLE>
              </PROGRAMSLOT>
         </DAY>
          ……..
          ……..
          ……..
          ……..
          ……..
     </CHANNEL>
      …………
      …………
      …………
 </TVSCHEDULE>
```

Figure 5.3: An example XML document conforming to the TV programs guide DTD.

Test 2 focuses on proving the correctness of the entire toolkit. Initially a relational schema is created using the SCE component. Then, a sample XML document is queried using the XML-QL processor and an output XML document is created. The same XML document is loaded into the database. The input XML-QL query is translated to SQL. The correctness of the toolkit is proved when an equivalent XML document is created by RXE. Even though efficiency is not the primary focus, this aspect of the toolkit can be illustrated by comparing the output of the XML-QL processor, implemented by AT&T and the output of the RXE. To illustrate, the XML-QL processor is used to query an XML document. The XML document being queried is displayed in Figure 5.3. The XML-QL processor expects the input query to be stored in a file and then generated an output file containing the results. The XML-QL processor is invoked from the command-line as follows:

xmlql -q <query_filename> -o <output_filename>.

```
function query()
{
 WHERE
 <Ontology>
  <TVSCHEDULE>
     <CHANNEL>
       <BANNER>   $B </>
       <DAY>
          <DATE1> $D </>
          <PROGRAMSLOT>
            <TIME> $T </>
            <PROG_TITLE> <PCDATA> $P_TITLE </> </>
          </>
       </>
     </>
  </>
 </> in "TV_data.xml",
    $P_TITLE="SPIN CITY"

 CONSTRUCT
 <Ontology>
  <MY_SCHEDULE>
    <BANNER> $B </>
    <DATE1>  $D </>
    <TIME_SLOT> $T </>
    <PROGRAM_NAME> $P_TITLE </>
  </>
 </>
}
```

Figure 5.4: An XML-QL query to retrieve information about a particular TV program.

The XML-QL query file is shown in Figure 5.4.

```
<?xml version="1.0" encoding="UTF-8"?>
 <MY_SCHEDULE>
  <PROGRAM_NAME>SPIN CITY</PROGRAM_NAME>
  <BANNER> ABC </BANNER>
  <DATE1> 04-24-2001 </DATE1>
  <TIME_SLOT>
   <HRS>     07 </HRS>
    <MINS>   00 </MINS>
  </TIME_SLOT>
 </MY_SCHEDULE>
```

Figure 5.5: XML-QL processor output in the form of an XML document.

The output of the XML-QL processor is an XML document as shown in Figure 5.5. The XML-QL query generated must have the same structure as the XML document being queried. The processor uses the "CONSTRUCT" clause in the XML-QL query to format the results in the XML document.

```
SELECT
             CHANNEL.BANNER, DATE1.DATE1, TIME.HRS, TIME.MINS,
            PROGRAMSLOT.PROG_TITLE
FROM
            CHANNEL, DATE1, TIME, PROGRAMSLOT, DAY,  CHANNEL_DAY,
            DAY_DATE1, DAY_PROGRAMSLOT, PROGRAMSLOT_TIME
WHERE
            PROGRAMSLOT.PROG_TITLE LIKE '%SPIN CITY%'  AND
            CHANNEL.CHANNEL_PK_ID=CHANNEL_DAY.CHANNEL_FK_ID  AND
            CHANNEL_DAY.DAY_FK_ID = DAY.DAY_PK_ID  AND
            DAY.DAY_PK_ID = DAY_DATE1.DAY_FK_ID  AND
            DAY_DATE1.DATE1_FK_ID = DATE1.DATE1_PK_ID  AND
            DAY.DAY_PK_ID = DAY_PROGRAMSLOT.DAY_FK_ID  AND
            DAY_PROGRAMSLOT.PROGRAMSLOT_FK_ID =
            PROGRAMSLOT.PROGRAMSLOT_PK_ID  AND
            PROGRAMSLOT.PROGRAMSLOT_PK_ID =
            PROGRAMSLOT_TIME.PROGRAMSLOT_FK_ID  AND
            PROGRAMSLOT_TIME.TIME_FK_ID = TIME.TIME_PK_ID
```

Figure 5.6: Equivalent SQL query to retrieve information about a particular TV program.

Then the XML document in Figure 5.3 is stored into the relational database using the loader component. The XML-QL query in Figure 5.5 is translated to SQL using a conversion tool that is part of another ongoing research project in the Database Research and Development Center. Figure 5.6 displays the equivalent SQL query. The SQL query is used to query the underlying relational data warehouse by the Relational-to-Xml-Engine (RXE) component of the toolkit. The RXE, then converts the relational results into an equivalent XML document.

```
<Result>
  <ROW ID="1">
    <BANNER>ABC</BANNER>
    <DATE1>04-24-2001</DATE1>
    <HRS>07</HRS>
    <MINS>00</MINS>
    <PROG_TITLE>SPIN CITY</PROG_TITLE>
  </ROW>
</Result>
```

Figure 5.7: Output of the RXE in the form of an XML document.

The RXE, then converts the relational results into an equivalent XML document as shown in Figure 5.7.

5.3. Analysis of the Results

In test 1, the relational schema created using the toolkit captures the cardinality constraints conveyed in the DTD as referential constraints. The current version does not represent the domain constraints. To exemplify, an attribute defined as #REQUIRED is mapped to a relational attribute but without being constrained as “NOT NULL”. This transformation is valid due to the fact that the loaded XML documents conform to the DTD describing the IWiz schema. Hence, the domain constraints are checked in the XML document, and can be ignored in the relational schema. One of the other differences is in the treatment of composite attributes. In relational algebra, the members of the composite attributes are nested into the relation. For example, a composite attribute such as “Time” having “Hrs” and “Mins” as members will be stored as two relational attributes, “Hrs” and “Mins”. But the SCE creates a separate table for “Time” leading to an inefficient translation. Elements in a DTD are used to express both nested attributes and entities; due to this it is not possible to distinguish between a nested attribute and an entity from a DTD declaration. Also, in the relational schema, tables are created with a single attribute; table “DAY” in Figure 5.2, for example. This translation helps in recreating the DTD from the schema and constraints description. Thus it provides a “round-tripping” between the DTD and relational schema. The SCE treats all the referential constraints as many to many even though they are expressed as one to one. According to the norms of normalization, such a mapping is inefficient. But the principles of normalization hold for rigid firmly established relations. But in the case of XML, this sort of mapping can be acceptable and can be handy when there is a change to the global schema. Thus, a change to a cardinality constraint in the DTD can be easily incorporated into the relational schema. More importantly, a DTD only describes the cardinality relationship between elements in a coarse manner. The SCE is also constrained by the restrictions in Oracle. Oracle does not allow the creation of a table with a standard data-type as the name. For example, if “Number” were a *‘CONCEPT”* element, then the name of this element has to be altered in the DTD because “Number” is a standard data type in Oracle. The maximum size of relational attribute name in the schema is 30 characters. The algorithmic complexity for the schema creation process can be broken down as follows: It takes constant time to find out if an element is a CONCEPT. Creating the attributes for the relational table will require to traversing all the children of this node, which can take a worst case time of O(n). Creation of the parent_child tables and referential constraints take constant time. This process is continued recursively on the children. Hence the overall complexity is O(n^2), where n is the number of element and attribute definitions in the DTD.

The minimal representational features in the DTD and the rules governing the names and sizes of attribute names in Oracle 8*i* limit the SCE. The SCE is able to map the metadata information represented in the DTD to the relational model validating the schema creation.

In test 2, the correctness of the toolkit is experimented and proved. The toolkit is able to retrieve a loaded document and construct an equivalent document. Structure can be imparted to this document by having additional information regarding the output tags. As shown in Figure 5.5, it can be seen that the output XML document produced by the XML-QL processor has an already embedded structure. But the performance of the processor degrades with the size of the data set. In particular, when the size of the input XML document exceeds 4 Megabytes of data, the processor crashes. On the other hand the XML document generated by the RXE lacks structure, when additional information is not provided. It is merely a representation of the relational data set. However structure can be imparted to the output document if, path expressions for all the output tags in the relational resultset are provided to the RXE. Further enhancement to the performance can be achieved if the XML document is generated inside the relational engine. The RXE is robust and can handle large size of input data. Contrasting the operations of the XML-QL processor and the RXE, a clear difference in execution speeds can be noticed when the data size is increased. Although the RXE has an initial database connection time overhead, it executes at a faster rate when the data size is increased. The loading time for the input data set requires further improvements. The algorithmic complexity of this process is computed as shown. When the input SQL query is simple ie, it does not contain a “GROUP By” clause, the RXE looks up the metadata data structure provided by the JDBC API, which would take constant time, to create the “tag” and includes the data contained in the resultset as the tag value. Thus this operation requires a O(n) time. If the input SQL query includes a “GROUP By” clause, the RXE has to group the resulting data by the grouping attributes mentioned in the clause. Thus this operation could take O(n*g) where n is the number of resulting tuples and g is the number of grouping attributes. All in all, RXE clearly outperforms the XML-QL processor when the size of the data set gets large (> 4MB).

CHAPTER 6
CONCLUSIONS

6.1. Summary

XML management systems are an important area of research and development in the industry today. The area is relatively new and is undergoing constant changes. The W3C recommendation for an XML query language and XML schema would impact the development of such systems. The objective of this thesis is to provide a solution to one such system that manages XML data in a relational database system.

In this thesis, we used the toolkit as a part of the Warehouse Manager component of the IWiz prototype. The toolkit was used to automatically create the schema for the data warehouse and to store the data contained in XML documents. The internal data model and query language used in the IWiz system is XML and XML-QL; this is abstracted from the user. The toolkit uses relational system as the data model, and understandably SQL as the query language. The toolkit requires all incoming XML documents to adhere to a global schema, a DTD. The DTD is also used to create the relational schema. The Document Object Model (DOM) is used to parse both the DTD and the XML documents to generate the relational scripts, to generate the schema and to create the tuples for the relational tables.

6.2. Contributions

The contributions of this work are as follows. We designed and implemented an algorithm to automatically translate a DTD to a relational schema, a feature absent in a few commercial products. The algorithm was able to identify the foreign key constraints automatically from the structure of the DTD. The XML loader was designed and the load-append strategy implemented. Thus the data in an XML document was automatically stored in the relational database. A database connection engine, providing a set of API to other modules was implemented. Finally, the XML document generator was designed and implemented, having features to group the results in the outputted XML document. Thus, a user currently using a traditional database could "XMLize" his system by using this toolkit and by manually creating a global DTD and generating XML documents from the relational tables.

The underlying research for the development of the toolkit would have a considerable influence on the current technology. It has provided an XML wrapper system around an existing relational system. XML is the dominant model to represent information on the Web today. The need for an XML data management system has led to the development of several commercial products. The toolkit has been developed in view of the existing products in the industry today. Finally, with respect to the IWiz prototype, the toolkit has provided the data warehouse manager functionality. Enhancements to the toolkit, especially to the Relational-to-XML-engine and the XML loader could reduce the overall *wait-time* for the user, increasing the throughput of the IWiz system.

Our solution to the problem of XML data management was derived over a period of 12 months. The first two months were spent in getting acclimatized to XML and related technologies. After this phase, about three months were spent in studying related literature and discovering the challenges to this problem. We framed the architecture of the toolkit in the next two months. The implementation was completed in five months. Currently, testing of the toolkit and its integration in IWiz is underway.

We have developed the toolkit version 1.0 and hope it will be a starting point in the development of a more robust and extensive system with additional features. In this ever-changing Information technology era, changes are happening to XML rapidly. Several aspects of the XML language are yet to be standardized. This factor has prohibited us from thoroughly covering and using all aspects of the XML technology at this point in time.

6.3. Future Work

The mechanism of data definition for XML is shifting from using DTDs to using XML schema. The XML Schema has been recently advanced to proposed recommendation status. The richer data definition facilities in XML schema when compared to DTDs will help the SCE in creating "apt" field types in the relational schema.

When the global schema (DTD) changes, the current system would require the relational schema to be re-created. Thus a new feature to handle the incremental maintenance of the relational schema could work towards altering the database metadata like creating new tables and adding new columns to currently existing tables. A one to one relation between two elements can become a m:n relation due to the introduction of a "*", "+" or due to a new recursive definition in the DTD. The handling of parent child relations as m:n can help when the DTD changes.

Recursive definitions in DTD declarations are not currently handled. But this could never cause an incorrect schema creation in the current implementation because when elements recursively define one another, automatically they are concepts, and are hence mapped to separate relational tables. A future feature could detect these recursive definitions and try to optimize the schema creation.

A component to translate an existing relational schema to a DTD and to generate a set of XML documents using the data stored in the relational tables adhering to the DTD. Then, the output of this component can be fed as the inputs to the toolkit. A new relational schema loaded with the data as before can be created. Thus a relational database system can be shifted to the XML system automatically. For all future transactions, this system will behave like an XML system but using the relational sources as before.

The Loader currently implements the load-append strategy. The load-merge and load-erase maintenance strategies could be built into the toolkit making it more powerful.

Many researchers are suggesting changes to the relational system that moves the generation of XML documents to the relational engine. Using such built-in features of the database can increase the efficiency of the toolkit. Usage of stored procedures and functions, reducing the utilization of JDBC can fasten the loading and document generation process.

LIST OF REFERENCES

[1] R. Aranha, J. Cho, A. Crespo, H. Garcia-Molina, J. Hammer. “Extracting Semistructured Information from the Web.” In Proceedings of the Workshop on Management of Semistructured Data, Tucson, Arizona, 1997.

[2] S. Abiteboul, "Querying Semistructured Data," Proceedings of the International Conference on Database Theory,” Delphi, Greece, 1997.

[3] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom, “The TSIMMIS Project: Integration of Heterogeneous Information Sources,” in Proceedings of the Tenth Anniversary Meeting of the Information Processing Society of Japan, Tokyo, Japan, 1994.

[4] S. Abiteboul, R. Goldman, K. Haas, Q. Luo, J. McHugh, S. Nestorov, D. Quass, A. Rajaraman, H. Rivero, J. Ullman, J. Widom and J. Wiener, “LORE: A Lightweight Object REpository for Semistructured Data.” Proceedings of the ACM SIGMOD International Conference on Management of Data, Montreal, Canada, 1996.

[5] World Wide Web Consortium, "Extensible Markup Language (XML) 1.0," W3C Recommendation, 1998, available at http://www.w3.org/TR/1998/REC-xml-19980210, March 2000.

[6] R. Goldman, J. McHugh, J. Widom. “From Semistructued Data to XML: Migrating the Lore Data Model and Query Language,” Proceedings of the 2nd International Workshop on the Web and Databases (WebDB ’99), Philadelphia, Pennsylvania, 1999.

[7] D. Suciu. “Semistructured Data and XML,” In Proceedings FODO Conference, Kobe, Japan, 1998.

[8] J. Hammer, “The Information Integration Wizard (IWiz) Project,” University of Florida Technical Report, Department of Computer and Information Science and Engineering, 1999.

[9] G. Kappel, E. Kapsammer, W. Retschitzegger. “Towards Integrating XML and Relational Database Systems,” International Conference on Conceptual Modeling/ the Entity Relationship Approach, Salt Lake City, USA, 2000.

[10] R. Barr, M. Carey, H. Pirahesh, B. Reinwald, J. Shanmugasundaram, E. Shekita. “Efficiently Publishing Relational Data as XML Documents,” VLDB Conference, Egypt, 2000.

[11] R. Anderson, D. Baliles, D. Birbeck, M. Kay, S. Livingstone, B. Loesgen, D. Martin, N. Ozu, B. Pear, J. Pinnock, P.Stark, K. Williams, “Professional XML,” Wrox Press, 2000.

[12] World Wide Web Consortium, "Overview of SGML Resources," October 2000, available at http://www.w3.org/MarkUp/SGML, May 2000.

[13] World Wide Web Consortium, "XML Schema Part 1: Structures," Working Draft, 22 September 2000, available at http://www.w3.org/TR/xmlschema-1/, May 2000.

[14] World Wide Web Consortium, "XML Schema Part 2: Datatypes," Working Draft, 22 September 2000, available at http://www.w3.org/TR/xmlschema-2/, June 2000.

[15] World Wide Web Consortium, "Document Object Model (DOM) Level 1 Specification," 1998, available at http://www.w3.org/TR/REC-DOM-Level-1/, April 2000.

[16] “The Document Object Model (DOM) Level 2 Core Specification,” The World Wide Web Consortium (W3C), http://www.w3.org/TR/DOM-Level-2-Core/2000-11-13 2000, December 2000.

[17] XML-QL User’s Guide: Basics, http://www.research.att.com/~mff/xmlql/doc/sitegraph.grnoid5.html, March 2001.

[18] S. Cluet, S. Jacqmin and J. Siméon, “The New YATL: Design and Specifications,” Technical Report, INRIA, 1999.

[19] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener, “The Lorel Query Language for Semistructured Data,” International Journal on Digital Libraries, vol. 1, pp. 68 - 88, available at ftp://db.stanford.edu/pub/papers/lorel96.ps, April 2001.

[20] J. Robie, “The design of XQL,” 1999, http://www.texcel.no/whitepapers/xql-design.html, April 2001.

[21] “XML-QL : A Query Language for XML, Version 0.9” : available at http://www.research.att.com/~mff/xmlql/, 2000, March 2001.

[22] J. Hammer, “Data Warehousing Seminar,” information available at http://www.cise.ufl.edu/~jhammer/classes/wh-seminar/overview.html, April 2001.

[23] H. Garcia-Molina, J. Hammer, J. Widom, W. Labio, and Y. Zhuge, "The Stanford Data Warehousing Project," Data Engineering Bulletin, vol. 18, 1995.

[24] D. Florescu, D. Kossman. "Storing and Querying XML Data Using an RDBMS." In IEEE Data Engineering Bulletin, volume 22(3), 1999.

[25] D. DeWitt, G. He, J. Naughton, J. Shanmugasundaram, K. Tufte, C. Zhang, " Relational Databases for Querying XML Documents: Limitations and Opportunities," Proceedings on the 25th International Conference On Very Large Databases (VLDB), Edinburg, 1999.

[26] W.W. Chu, D. Lee, " Constraints-preserving Transformation from XML Document Type Definition to Relational Schema (Extended Version)," UCLA-CS-TR 200001, 2000, available at http://www.cs.ucla.edu/dongwon/paper, 2001.

[27] H. Garcia-Molina, H. Gupta, J. Labio, J. L. Wiener, J. Widom, Y. Zhuge, "The WHIPS Prototype for Data Warehouse Creation and Maintenance." In Proceedings of the ACM SIGMOD Conference, Tuscon, Arizona, 1997.

[28] H. Garcia-Molina, H. Gupta, J. Labio, J. L. Wiener, J. Widom, Y. Zhuge, "A System Prototype for Warehouse View Maintenance," Proceedings of the ACM Workshop on Materialized Views: Techniques and Applications, Montreal, Canada, 1996.

[29] Oracle XML SQL Utility (XSU) for Java and XSQL Servlet, available at http://technet.oracle.com/tech/xml, August 2000.

[30] GMD-IPSI implementation of PDOM, available at http://xml.darmstadt.gmd.de/xql/index.html, October 2000.

[31] M. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu, "STRUDEL: A Website Management System," presented at SIGMOD'97, Proceedings ACM SIGMOD International Conference on Management of Data, Tucson, Arizona, 1997.

[32] B.Beauchemin, "Investigating the differences between SQL Server 2000's XML integration and Microsoft's XML technology preview," available at http://www.msdn.microsoft.com/library/periodic/period00/thexmlfiles.htm, April 2001.

[33] R. Bourret, "XML and Databases." Manuscript available at http://www.rpbourret.com/xml/XMLAndDatabases.htm, April 2001.

[34] H. Loeser, N. Ritter, B. Surjanto, “XML Content Management based on Object-Relational Database Technology,” Proceedings of the 1st International Conference On Web Information Systems Engineering (WISE), Hongkong, 2000

[35] C. Cooper, "Benchmarking XML Parsers: A performance comparison of six stream-oriented XML parsers," 1999, available at http://www.xml.com/pub/Benchmark/article.html, March 2001.

[36] D. Florescu, D. Kossman. “A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database.” Technical Report, INRIA, France, 1999.

[37] T.Bohme, E.Rahm, “XMach-1: A Benchmark for XML Data Management,” In Proceedings of German database conference BTW2001, Oldenburg, Springer, Berlin, 2001.

BIOGRAPHICAL SKETCH

Ramasubramanian Ramani was born in New Delhi, India, in 1977. He received his bachelor’s degree in computer science from Sri Venkateswara College of Engineering, University of Madras in 1995. In 1999, he obtained admission to the Master of Science program in the Computer and Information Science and Engineering Department, University of Florida. He will be graduating in August 2001. His research work in the University of Florida has been focussed on information integration, XML and database systems.