

University of Windsor

## Scholarship at UWindor

---

Electronic Theses and Dissertations

Theses, Dissertations, and Major Papers

---

2002

### Using XML views to improve data-independence of distributed applications that share data

Xun Luo

*University of Windsor*

Follow this and additional works at: <https://scholar.uwindsor.ca/etd>

---

#### Recommended Citation

Luo, Xun, "Using XML views to improve data-independence of distributed applications that share data" (2002). *Electronic Theses and Dissertations*. 4489.

<https://scholar.uwindsor.ca/etd/4489>

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email ([scholarship@uwindsor.ca](mailto:scholarship@uwindsor.ca)) or by telephone at 519-253-3000ext. 3208.

## **INFORMATION TO USERS**

**This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.**

**The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.**

**In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.**

**Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.**

**ProQuest Information and Learning  
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA  
800-521-0600**

**UMI<sup>®</sup>**



# **Using XML Views to Improve Data- Independence of Distributed Applications That Share Data**

**By  
Xun Luo**

**A Thesis**

**Submitted to the Faculty of Graduate Studies and Research  
through the School of Computer Science in Partial  
Fulfillment of the Requirements for the Degree  
of Master of Science at the  
University of Windsor**

**Windsor, Ontario, Canada**

**2002**

**© 2002 Xun Luo**



**National Library  
of Canada**

**Acquisitions and  
Bibliographic Services**

**385 Wellington Street  
Ottawa ON K1A 0N4  
Canada**

**Bibliothèque nationale  
du Canada**

**Acquisitions et  
services bibliographiques**

**385, rue Wellington  
Ottawa ON K1A 0N4  
Canada**

*Your file Votre référence*

*Our file Notre référence*

**The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.**

**The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.**

**L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.**

**L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.**

**0-612-75797-8**

**Canada**

**APPROVED BY:**



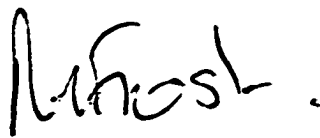
---

**Prof. Philip H. Alexander, External Reader**  
**Department of Electrical and Computer Engineering**



---

**Dr. Xiaojun Chen, Departmental Reader**  
**School of Computer Science**



---

**Dr. Richard A. Frost, Supervisor**  
**School of Computer Science**

## **Abstract**

The development and maintenance of distributed software applications that support and make efficient use of heterogeneous networked systems is very challenging. One aspect of the complexity is that these distributed applications often need to access shared data, and different applications sharing the data may have different needs and may access different parts of the data. Maintenance and modification are especially difficult when the underlying structure of the data is changed for new requirements.

The eXtensible Markup Language, or XML, has emerged as the universal standard for exchanging and externalizing data. It is also widely used for information modeling in an environment consisting of heterogeneous information sources.

CORBA is a distributed object technology allowing applications on heterogeneous platforms to communicate through commonly defined services providing a scalable infrastructure for today's distributed systems.

To improve data independence, we propose an approach based on XML standards and the notion of views to develop and modify distributed applications which access shared data. In our approach, we model the shared data using XML, and generate different XML views of the data for different applications according to the DTDs of the XML views and the application logic. When the underlying data structure changes, new views are generated systematically. We adopt CORBA as the distributed architecture in our approach. Our thesis is that: views to support data-independence of distributed computing applications can be generated systematically from application logic, CORBA IDL and XML DTD.

***To My Parents and My Family***



## **Acknowledgements**

**I would like to acknowledge my gratitude to my supervisor, Dr. Richard A. Frost, for all his guidance, valuable advice, and time commitments, without which this work would never be achieved. I would also like to express appreciation to committee members, Dr. Jessica Chen, and Prof. Philip Alexander for not only their suggestions, but also comprehensive feedback.**

**Many thanks go to secretaries of the Department of Computer Science Ms. Mary Mardegan and Ms. Margaret Garabon for their generous help in the past two years.**

## TABLE OF CONTENTS

<b>Abstract.....</b>	<b>iii</b>
<b>Dedication.....</b>	<b>iv</b>
<b>Acknowledgments.....</b>	<b>v</b>
<b>Table of Contents.....</b>	<b>vi</b>
<b>List of Figures.....</b>	<b>x</b>
<b>List of Document Listings.....</b>	<b>xi</b>
<b>1 Introduction. ....</b>	<b>1</b>
1.1 Motivating problems.....	1
1.2 Observations.....	2
1.3 Possible Solutions.....	5
1.4 The Thesis.....	6
1.5 Organization of the Thesis Report.....	7
<b>2 Background: Distributed Systems and Data-independence.....</b>	<b>8</b>
2.1 Overview of Distributed Systems.....	8
2.2 Architectures for Building Distributed Systems.....	8
2.3 Data-independence.....	10
<b>3 Investigation of XML.....</b>	<b>12</b>
3.1 Overview of XML.....	12
3.2 XML Syntax.....	13
3.3 Document Type Definition.....	16

3.3.1	DTD Syntax.....	16
3.3.2	DTD As the Content Model.....	19
3.4	DOM and SAX.....	20
3.4.1	The Document Object Model.....	20
3.4.2	The Simple API for XML.....	22
3.5	The XML Path Language.....	23
3.5.1	XPath Overview.....	23
3.5.2	XPath as a Query Language.....	24
3.6	Xquery: An XML Query Language.....	25
3.7	Extensible Stylesheet Language Transformation.....	30
<b>4</b>	<b>Investigation of CORBA.....</b>	<b>33</b>
4.1	Overview of CORBA.....	33
4.2	ORB Core.....	34
4.3	CORBA Invocations.....	35
4.4	OMG Interface Definition Language.....	36
4.5	Benefits of CORBA.....	38
<b>5</b>	<b>Constructing the XML Document for Storing the Shared Data.....</b>	<b>41</b>
5.1	Design XML Structure from Scratch.....	41
5.1.1	XML Content Models.....	41

5.1.2	Modeling data values.....	43
5.1.3	Modeling relationships among elements.....	45
5.2	Migrating a Database to XML.....	49
5.3	Transforming from Flat Files to XML.....	51
<b>6</b>	<b>Different Approaches for Generating XML Views.....</b>	<b>54</b>
6.1	Architecture of Distributed Systems for Data-independence.....	54
6.2	XQuery/XSL Approach.....	56
6.3	XPath Mapping Approach.....	61
<b>7</b>	<b>Guidelines for Systematically Generating XML Views for the Initial System....</b>	<b>64</b>
7.1	Guidelines for XQuery/XSL Approach.....	65
7.2	Guidelines for XPath Mapping Approach.....	72
<b>8</b>	<b>Guidelines for Systematically Modifying the System After Changes.....</b>	<b>78</b>
8.1	Guidelines for XQuery/XSL Approach.....	78
8.2	Guidelines for XPath Mapping Approach.....	78
<b>9</b>	<b>Analysis of Work Done and Results.....</b>	<b>80</b>
9.1	Comparison of the Two Approaches Proposed.....	80
9.2	Analysis of Using XML and Views to Improve Data-independence.....	82
9.3	Element Vs. Attribute in Data Modeling Using XML.....	83
9.3.1	Compatibility with databases.....	84
9.3.2	Data typing.....	85
9.3.3	Document size.....	85
9.4	Integration of CORBA and XML.....	86

<b>10 Conclusions.....</b>	<b>87</b>
10.1 The Defense of the Thesis.....	87
10.2 Problems and Future work.....	88
<b>References.....</b>	<b>89</b>
<b>Appendix: Program List.....</b>	<b>94</b>
<b>Vita Auctoris.....</b>	<b>110</b>

## LIST OF FIGURES

<b>Figure 1.1</b>	<b>Architecture of a distributed system. ....</b>	<b>2</b>
<b>Figure 1.2</b>	<b>Architecture of the possible solution ....</b>	<b>6</b>
<b>Figure 3.1</b>	<b>DOM node tree of bibliography document. ....</b>	<b>21</b>
<b>Figure 3.2</b>	<b>The SAX parser generate events ....</b>	<b>22</b>
<b>Figure 3.3</b>	<b>Flow of data in a FLWR expression ....</b>	<b>27</b>
<b>Figure 3.4</b>	<b>XSLT processor working.....</b>	<b>31</b>
<b>Figure 4.1</b>	<b>The CORBA architecture.....</b>	<b>35</b>
<b>Figure 4.2</b>	<b>Performing a request with CORBA.....</b>	<b>36</b>
<b>Figure 5.1</b>	<b>Relationship between Invoice and LineItem.....</b>	<b>45</b>
<b>Figure 5.2</b>	<b>Relationships among Invoice, LineItem and Product.....</b>	<b>46</b>
<b>Figure 6.1</b>	<b>Architecture of distributed systems for data-independence.....</b>	<b>54</b>
<b>Figure 6.2</b>	<b>Support XML view using single XQuery/XSL.....</b>	<b>56</b>
<b>Figure 6.3</b>	<b>An example of XML data and view.....</b>	<b>57</b>
<b>Figure 6.4</b>	<b>Structure of the XML document after change.....</b>	<b>58</b>
<b>Figure 6.5</b>	<b>An example of a view with major difference in structure ....</b>	<b>58</b>
<b>Figure 7.1</b>	<b>Structure of the shared data for Book Catalog System.....</b>	<b>63</b>
<b>Figure 7.2</b>	<b>Structure of the view for Book Catalog System.....</b>	<b>66</b>

## LIST OF DOCUMENT LISTINGS

<b>Listing 3.1 A bibliography in XML.....</b>	<b>14</b>
<b>Listing 3.2 DTD for the bibliography document .....</b>	<b>16</b>
<b>Listing 3.3 XQuery for Example1.....</b>	<b>28</b>
<b>Listing 3.4 Result of the XQuery for Example1.....</b>	<b>28</b>
<b>Listing 3.5 XQuery for Example2.....</b>	<b>29</b>
<b>Listing 3.6 Result of the XQuery for Example2.....</b>	<b>29</b>
<b>Listing 5.1 Definition of Customer using elements.....</b>	<b>44</b>
<b>Listing 5.2 Definition of Customer using attributes .....</b>	<b>44</b>
<b>Listing 5.3 Modeling relationships by containment .....</b>	<b>45</b>
<b>Listing 5.4 Example of containment .....</b>	<b>46</b>
<b>Listing 5.5 Modeling relationships using ID/IDREF pairs.....</b>	<b>47</b>
<b>Listing 5.6 Example of ID/IDREF pairs .....</b>	<b>48</b>
<b>Listing 6.1 Example XQuery to generate XML view .....</b>	<b>57</b>
<b>Listing 6.2 Example XSL to generate XML view. ....</b>	<b>58</b>
<b>Listing 6.3 Modifying the XQuery. ....</b>	<b>59</b>
<b>Listing 6.4 Modifying the XSL .....</b>	<b>60</b>
<b>Listing 6.5 Xquery to generate XML view with different structure.....</b>	<b>60</b>
<b>Listing 7.1 DTD for the shared data for Book Catalog System.....</b>	<b>64</b>
<b>Listing 7.2 A sample shared XML data for Book Catalog System.....</b>	<b>65</b>
<b>Listing 7.3 DTD for the view for Book Catalog System.....</b>	<b>66</b>
<b>Listing 7.4 priceView.xsl--XSL file for generating the view for Book Catalog System...67</b>	

<b>Listing 7.5 viewImpl.java – Java implementation of the view generating class.....</b>	<b>68</b>
<b>Listing 7.6 priceViewClient.java – Java implementation of the client application.....</b>	<b>69</b>
<b>Listing 7.7 viewServer.java – Java implementation of the CORBA server.....</b>	<b>70</b>
<b>Listing 7.8 priceView.xml – XPath mapping stored as XML file.....</b>	<b>72</b>
<b>Listing 7.9 priceViewImpl.java – Java implementation of the view supporting class.....</b>	<b>76</b>



# **1 Introduction**

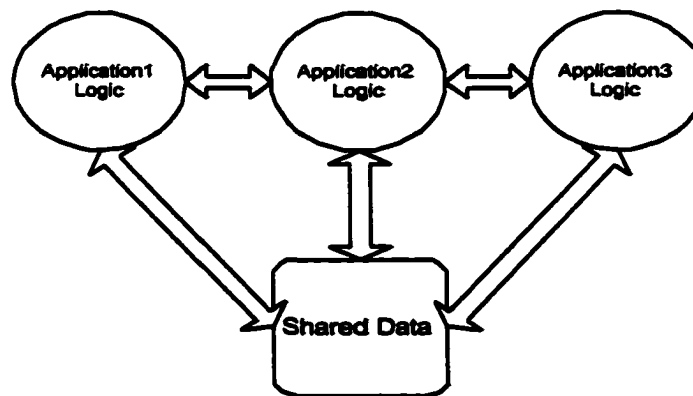
This thesis is primarily concerned with improving data-independence of distributed applications which access shared data through XML views. This chapter gives an introduction to this thesis work. It elicits the problems inspiring this work, and presents objectives and the thesis statement.

## **1.1 Motivating problems**

An important characteristic of large computer networks such as the Internet and corporate intranets is that they are heterogeneous. However, dealing with heterogeneity in distributed systems is not easy. In particular, the development and maintenance of software applications that support and make efficient use of heterogeneous networked systems is very challenging.

One aspect of the complexity is that these distributed applications often need to access shared data, and different applications sharing the data may have different needs and may access different parts of the data. Traditionally, shared data is often stored in a relational database to achieve data-independence and other amenities that database management system provides. The capability of database storage and processing is central in most information systems. Earlier, organizations used monolithic database management systems. However, nowadays there are often many isolated data repositories distributed over personal computers. Those data repositories

are often heterogeneous because of the differences in the semantics of data and DBMS differences such as different data models and query languages. Moreover, much of the existing electronic data lies outside of a DBMS. It lies in structured documents like HTML and SGML, non-standard data formats, legacy systems, etc. The structure of the non-relational data is sometimes irregular, unknown in advance, and changed often. This is a significant departure from the traditional database framework geared towards highly structured data described uniformly by a rigid schema. This makes the complexity of distributed systems more difficult to deal with. Figure 1.1 shows the architecture of a distributed system.



**Figure 1.1 Architecture of a distributed system**

For example, if the structure of the shared data is changed for new requirements, all applications accessing the data have to be modified accordingly. For a large system with hundreds or thousands of applications spread all over the world, this is nearly impossible.

## **1.2 Observations**

Research on semistructured data has aimed at extending database management techniques to data with irregular, unknown, and often changing structure. The advent of the eXtensible Markup Language (XML) and related technologies addresses this problem by providing a neutral syntax for interoperability among disparate systems and applying data management technology to documents. XML is a standard approved by the World Wide Web Consortium (W3C) that many believe will become the de facto data exchange format for the web.

The basic ideas underlying XML are very simple: tags on data elements identify the meaning of the data, rather than, e.g., specifying how the data should be formatted (as in HTML), and relationships between data elements are provided via simple nesting and references. Yet the potential impact is significant: Web servers and applications encoding their data in XML can quickly make their information available in a simple and usable format, and such information providers can interoperate easily. Information content is separated from information rendering, making it easy to process the data. XML supports the electronic exchange of machine-readable data. XML data shares many features of semistructured data: its structure can be irregular, is not always known ahead of time, and may change frequently and without notice. On the other hand it is easy to convert data from any source into XML, which will make it attractive for organizations to publish their electronic data in XML, and thus make them available to other applications in the system.

However, XML applications face the similar problem as those built using traditional databases: different users sharing XML data may have different needs and may want to see the same data differently, and this is not only at the presentation level. For XML applications to reach their full potential, we need the same mechanism, views as in databases to achieve data-independence at a higher level. The notion of views is essential in databases. It allows various users to see data from different viewpoints. When data is represented using XML, the problem of views is more crucial than in standard database applications because the data is often from heterogeneous sources and the structure of the data is changed frequently. Views provide the means to add a structured interface on top of the underlying data, which provides data-independence for applications that access the data via the views.

In the relational world, a view is simply specified by a query. A lot of the technology developed for relational databases remains meaningful in the context of views for XML data. XQuery is the XML query language the W3C is developing for extracting data from XML documents. The XML Stylesheet Language (XSL) also has facilities that could serve as a basis for an XML query language.

Therefore, our research is motivated by the idea of improving data-independence of distributed applications which access shared data by representing the shared data in XML and generating higher level abstractions of the data –XML views.

### **1.3 Possible Solutions**

**CORBA is both a language-independent and a location-transparent framework, which means objects at different locations are interchangeable as long as the interfaces to the objects remain the same. CORBA itself is an excellent technology for building distributed applications involving multiple languages and commercial vendors. Therefore, CORBA is adopted in our thesis as the infrastructure to build distributed systems. CORBA will be discussed in more detail in later chapters.**

**The observations above lead to a possible solution based on XML and CORBA. Figure 1.2 illustrates the architecture of the solution. Use CORBA as the middleware to build the distributed applications to facilitate the communication between the applications; use XML as the neutral syntax to represent the shared data; construct XML views on top of the XML data for different applications according to the application logic, CORBA IDL and XML DTD.**

**When the DTD of the XML data is changed for new requirements of the system, only the queries that generate the views for different applications need to be modified in this setting.**

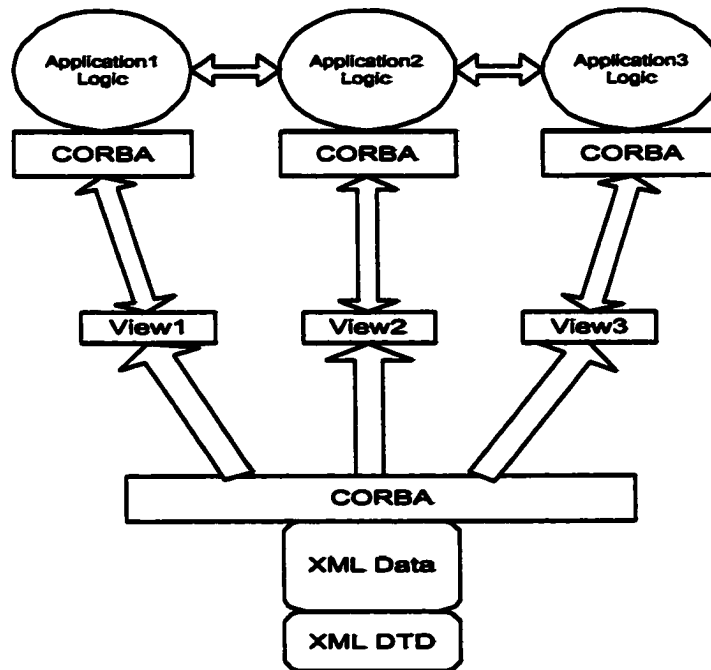


Figure 1.2 Architecture of the possible solution

## 1.4 The Thesis

The ultimate goal of this thesis work is to find a systematic approach to the development and maintenance of distributed applications based on XML data and XML views, including how to build the initial distributed system, especially the XML views, and how to modify the system when changes to DTD of the XML data occur.

The thesis statement is:

**“XML views to support data-independence of distributed computing applications can be generated systematically from application logic, CORBA IDL and the XML DTD.”**

**The following is the work done to support the thesis:**

- **Investigate XML, DTD and XQuery.**
- **Investigate CORBA and IDL.**
- **Propose data modeling approaches using XML.**
- **Develop guidelines for systematically generating XML views for the initial distributed system.**
- **Develop guidelines for systematically modifying the system after the structure of shared data is changed.**
- **Develop a prototype system to demonstrate our approaches.**
- **Analyze the work done and results.**
- **Draw conclusions about the extent to which the thesis has been proven.**

## **1.5 Organization of the Thesis Report**

**Chapter 2 gives background information about distributed systems and data-independence. Chapter 3 introduces XML and some related technologies including DTD, DOM, SAX, XPath, XQuery and XSLT. Chapter 4 introduces CORBA and its IDL. Chapter 5 presents approaches for data modeling using XML. Chapter 6 gives details of the two approaches for improving data-independence of distributed applications accessing shared data. Chapter 7 and chapter 8 present the guidelines for systematically generating XML views for the initial distributed system and modifying the system after the structure of shared data is changed. Chapter 9 provides critical analysis of the work done. Chapter 10 gives our conclusions.**

## **2 Background: Distributed Systems and Data-independence**

This chapter primarily deals with some background information about distributed system and some generic concerns about the development and maintenance of distributed systems.

### **2.1 Overview of Distributed Systems**

A distributed system is a collection of autonomous computers linked by a network, with software designed to produce an integrated computing facility. Software of distributed systems can range from the provision of general-purpose computing facilities for groups of users to automated banking and multimedia communication systems. Distributed software coordinates the computers in the system to accomplish their tasks and share resources including hardware, software and data of the system. The key characteristics of distributed system software are: support for resource sharing, openness, concurrency, scalability, fault tolerance and transparency. The design goal of distributed systems is to achieve high performance, reliability, scalability, consistency and security.

### **2.2 Architectures for Building Distributed Systems**



Today, more and more distributed systems are built on large computer networks such as the Internet and corporate Intranets. An important characteristic of large computer networks is that they are heterogeneous. Ideally, heterogeneity and open systems enable us to use the best combination of hardware and software components for each portion of an enterprise. When the right standards for interoperability and portability between these components are in place, the integration of the components yields a system that is coherent and operational. However, dealing with heterogeneity in distributed-computing enterprises is not easy. In particular, the integration of software components that support and make efficient use of heterogeneous networked systems is very challenging.

In the mid-eighties, companies began to build distributed computing infrastructures using "Remote Procedure Call" such as Sun's Open Network computing (ONC) and OSF's DCE. In many cases, companies built their own in-house infrastructure necessary to support this architecture.

In the 1990s, a new distributed computing model became widely adopted, built around the concept of "Distributed Component Computing". While there is no universal approach to integrating the components, a number of protocols and technologies have emerged: SunSoft's Enterprise JavaBeans, OMG's CORBA, and Microsoft's DCOM. These distributed architectures directly address the technical issues of distribution and heterogeneity that are central to a modern large-scale software application. DCOM provides suitable solutions for the Microsoft

environment and EJB is ideal for Java applications. CORBA has clear advantages in terms of crossing platform and language boundaries, which is truly suitable for large-scale applications.

## **2.3 Data-independence**

Most software systems are subject to changing requirements that often necessitate alterations of the structure of shared data. However, these modifications imply time and money-consuming changes to application programs that run against the shared data. In order to avoid this annoying and costly situation, application programs should be independent of changes in the way data is structured and stored. This quality is known as data independence.

The single most significant distinguishing characteristic of a database is data independence. Database systems provide convenient access to shared data for a community of users having assorted requirements and database experience. This is accomplished in part by supplying each user with a view of only the relevant portions of the database. Views are tables that are defined in terms of queries over other tables. The view mechanism can be used to create a window on a collection of data that is appropriate for some group of users. Views are very useful from a security standpoint because they allow us to limit access to sensitive data by instead providing access to a restricted version of that data which is defined as a view. It is very important to note that views are very useful quite independently of security considerations because they

**allow us to create several presentations of the same data, each of which is tailored to the needs of a different group of users, without actually replicating the data.**

**The motivation behind the view mechanism is that we can tailor how users see the data. Users should not have to worry about the view versus base table distinction. The goal is indeed achieved in the case of queries on views. A direct consequence is that queries are generic; their answers depend only on the logical level of data.**

**In our approach based on XML data, we adopt the idea of XML views similar to the views in relational databases on top of the underlying data to achieve data-independence at a higher level. XPath, XQuery and XSLT are XML query languages that can be used to generate XML views.**

### **3 Investigation of XML**

The eXtensible Markup Language (XML) is an emerging open standard in the production and consumption of content managed by the W3C. XML as a text-based markup language has become the universal standard for exchanging and externalizing data in a platform-, language-, and protocol- independent fashion.

#### **3.1 Overview of XML**

XML is a data-oriented technology, based on a lightweight subset of the Standard Generalized Markup Language (SGML), suitable for the definition, storage, and retrieval of structured data. XML is inherently language independent. XML was originally created to improve document processing by separating presentation from content and by revealing documents' semantic structure. Now it is widely used as a common format for expressing data structure and content to reduce the obstacles to sharing data among diverse applications and databases.

XML is a language used to describe and manipulate structured documents. It is a flexible mechanism that accommodates the structure of specific applications. It provides a mechanism to encode both the information manipulated by the application and its underlying structure. XML lets information providers invent their own tags for particular applications and work with other organizations to define shared sets of tags that promote interoperability and that clearly separate content and presentation. XML

also supports validation in two ways. Application developers can associate an XML document with a document type description (DTD) that describes the structure to which the document should conform. In addition, because DTDs were intended for document management and cannot adequately model complex data, the W3C subsequently developed the XML schema specification, which adds data types, relationships, and constraints. Applications can use off-the-shelf XML parsers to validate imported data for conformation to a DTD or schema.

Although a young standard, XML already exerts significant influence on computing communities. A vibrant XML marketplace is providing inexpensive tools for preparing, validating, and parsing XML data. Application developers praise XML's extensibility; communities that share common data, such as the chemical industry, like XML's support for well-defined, common data representations. Several related standards also greatly increase XML's data sharing and management utility. These include Extensible Stylesheet Language (XSL), Document Object Mode (DOM), Simple API for XML (SAX), and XML query languages.

### **3.2 XML Syntax**

XML documents are composed of a structured collection of markup and content. Each tag either defines information used to describe how the document is to be interpreted or describes data contained within the document. There are six kinds of markup that can occur in an XML document: elements, entity references, comments,

processing instructions, marked sections, and document type declarations. Listing 3.1 is a small bibliography represented in XML.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE bib SYSTEM "bib.dtd">
3 <bib>
4     <book year="1994">
5         <title>TCP/IP Illustrated</title>
6         <author><last>Stevens</last><first>W.</first></author>
7         <publisher>Addison-Wesley</publisher>
8         <price> 65.95</price>
9     </book>
10    <!-- Unix Book -->
11    <book year="1992">
12        <title>Programming in the Unix environment</title>
13        <author><last>Stevens</last><first>W.</first></author>
14        <publisher>Addison-Wesley</publisher>
15        <price>79.95</price>
16    </book>
17
18    <book year="1999">
19        <title>The Economics of Technology for Digital TV</title>
20        <editor>
21            <last>Gerbarg</last><first>Darcy</first>
22            <affiliation>CITI</affiliation>
23        </editor>
24        <publisher>Kluwer Academic Publishers</publisher>
25        <price>129.95</price>
26    </book>
27 </bib>
```

**Listing 3.1** A bibliography in XML (from <http://www.bn.com>)

## Comments

Comments begin with “<!--” and end with “-->”. Line 10 of Listing 3.1 is an example of comments. Comments can contain any data except the literal string “--”.

Comments are not parsed by XML tools.

## XML Declarations

**`<?xml version= "versionNumber", encoding = "encodingFormat" standalone= "yes/no" ?>`**

An XML declaration is used to identify the associated block of data as an XML document; the declaration can have three attributes: "encoding" indicates the text encoding format of the document, "standalone" indicates the dependence on other documents defined external to the current document. Line 1 of Listing 3.1 contains the declaration indicating the version and encoding of the XML document.

### **Elements**

**`<elmentName>data value/ other element </elementName>`**

An element acts as a container which contain data values or other XML elements. Elements have a case-sensitive name that is defined within < and > characters. Each element has a start tag and an end tag. An empty element `<emptyElement/>` designates a place in a document where some action should occur. An XML document can have only one root element that contains all other elements. In Listing 3.1, bib, book, title, author, last, first, publisher, editor and price are elements, and bib is the root element of the document.

### **Attributes**

Attributes are white-space separated name-value pairs that occur inside tags after "element name" (`<elementName attribute = "attributeValue">` ). In XML, all attribute values must be quoted. In Listing 3.1, element book has an attribute year.

### 3.3 Document Type Definition

A Document Type Definition (DTD) is a structured collection of declarations which define the semantic constraints that apply to a particular type of XML document. A DTD is essentially a set of meta-information (includes tags required, relationships among tags, valid attribute values, named entities) that defines the required structure and characteristics of an XML document. Listing 3.2 is an example DTD for the bibliography document (Listing 3.1) introduced in last section.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT bib ( book+ ) >
<!ELEMENT book ( title, (author|editor)+, publisher, price ) >
<!ATTLIST book year NMTOKEN #REQUIRED >
<!ELEMENT publisher ( #PCDATA ) >
<!ELEMENT affiliation ( #PCDATA ) >
<!ELEMENT author ( last, first ) >
<!ELEMENT first ( #PCDATA ) >
<!ELEMENT price ( #PCDATA ) >
<!ELEMENT editor ( last, first, affiliation ) >
<!ELEMENT title ( #PCDATA ) >
<!ELEMENT last ( #PCDATA ) >
```

**Listing 3.2 DTD for the bibliography document**

#### 3.3.1 DTD Syntax

The syntax for DTDs is different from the syntax for XML documents.

#### DTD Header



A DTD must be declared at the beginning of an XML document after the XML declaration using the form:

```
<!DOCTYPE RootElementName [...]>
```

where root element name is used to identify the XML document type for which the DTD defines structure and characteristics. Line 2 of Listing 3.1 contains a DTD header. A DTD stored at some external URL is referenced with a system identifier using the following form:

```
<!DOCTYPE RootElementName SYSTEM "mySystemUrl" [...]>
```

For example, the DOCTYPE external subset might look like the following:

```
<!DOCTYPE rootElement SYSTEM "http://www.xmlserver.com/dtd/myDTD.dtd">
```

PUBLIC keyword can be used to replace SYSTEM to indicate that the DTD is widely used. The following DOCTYPE uses the PUBLIC keyword to reference the well-known DTD for HTML version 4.01.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3c.org/TR/html4/strict.dtd">
```

### **Element Declarations**

Element declarations identify the names of elements and the nature of their content. Element declarations are defined within <ELEMENT and > characters and can be one of the following forms:

- **Empty Content:** the element is defined to have no content.

```
<!ELEMENT elementName EMPTY>
```

- **Any Content:** the element is defined to have any content

**<!ELEMENT *elementName* ANY>**

- **Parsed character data: the element is defined to contain text**

**<!ELEMENT *elementName* #PCDATA>**

- **Content Model: A content model is defined to constrain the contents of the named element with one of the following symbols:**
  - **? designates that the model is optional**
  - **\* designates that the model may contain zero or more of the model elements.**
  - **+ designates that the model may contain one or more of the model elements.**

**So the basic format for the content model is:**

**<!ELEMENT *elementName* (*content\_model*)<sup>?</sup>|<sup>\*</sup>|<sup>+</sup> >**

**The content model definition within an element declaration itself can have the following formats:**

- ***sub\_element*<sup>?</sup>|<sup>\*</sup>|<sup>+</sup>**
- **sub elements separated by comma (,) indicate the elements must be contained in the sequence specified.**
- **sub elements separated by vertical bar (|) indicate any one of the elements can be contained within an enclosing element.**

### **Attribute Declarations**

**Attribute declarations define the characteristics of attributes that are associated with a named element. The attribute name along with an attribute type and default value may all be specified for an attribute. Attributes are declared using the following form:**

**<!ATTLIST *elementName attributedName AttributeType DefaultValue* >**

Where attribute type may be defined according to one of the following forms: ID, IDREF, IDREFS, ENTITY, ENTITIES, CDATA, NMTOKEN, or Enumerated-type. Optionally, the DTD can specify a default value for the attribute. Default attribute values are defined as one of the following forms: #REQUIRED, #IMPLIED, #FIXED or a literal value.

### **3.3.2 DTD As the Content Model**

DTD is the content model of an XML document. DTD encapsulates the hierarchical relationship of content in an XML document, and regulates what content is allowable in a conforming XML document. DTDs allow XML processing systems to validate data content and document structure.

DTDs can have multiple uses in creating views and querying XML data. QBE-style query interfaces [BARU99] may use DTDs to display the schema of a view and allow users to navigate it. DTDs may help in the design of the storage structures. Semistructured databases can use DTDs to semantically optimize their query plans [FERN98]. Finally DTDs may guide the production of style sheets, such as XSL scripts [ADLE01], that translate XML documents into browser-compatible HTML documents or other XML documents with different DTDs. It is clear that DTDs will be particularly useful. DTDs also guide the generation of XML views in our approach.

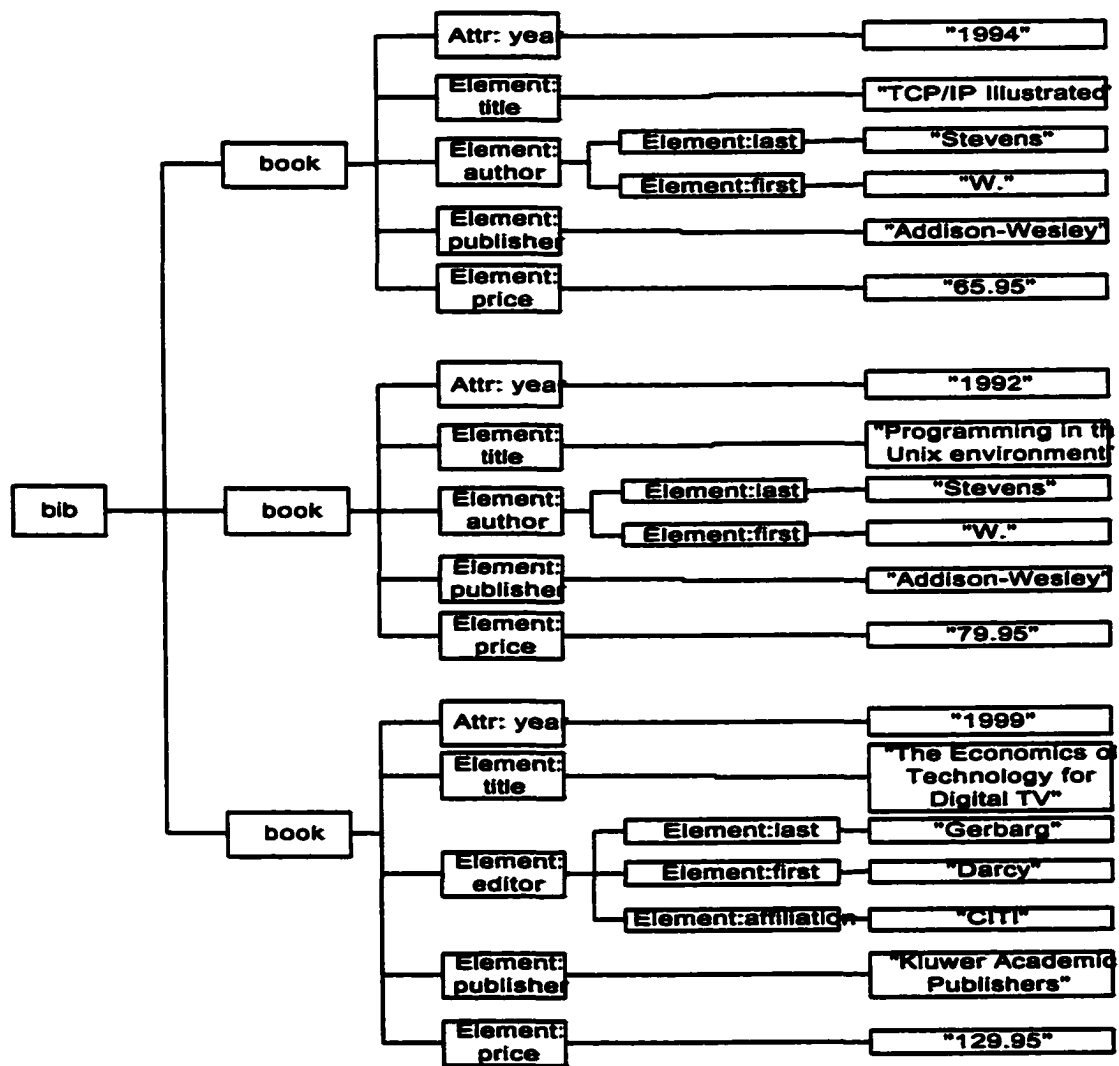
## **3.4 DOM and SAX**

DOM and SAX are two dominant APIs for processing XML documents.

### **3.4.1 The Document Object Model (DOM)**

The Document Object Model (DOM) is a platform- and language-neutral object-based interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The objects in the DOM allow the developer to read, search, modify, add to, and delete from a document. The DOM lays out a standard functionality for document navigation and manipulation of the content and structure of XML and HTML documents.

When the DOM is used to manipulate an XML text file, the first thing it does is to parse the file, breaking the file into individual elements, attributes, comments, and so on. It then creates a representation of the XML file as a node tree in memory. Developers may then access the contents of the document through the node tree, and make modifications to it as necessary. The DOM provides a robust set of interfaces to facilitate the manipulation of the DOM node tree. Figure 3.1 shows the DOM tree for our bibliography example from Listing 3.1. The DOM treats every item in the document as a node – elements, attributes, comments, processing instructions, and the text that makes up an element or an attribute.



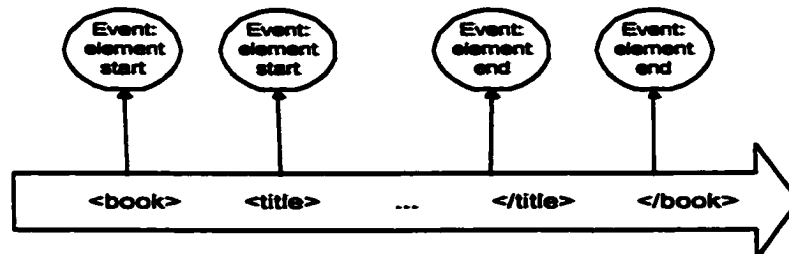
**Figure 3.1 DOM node tree of bibliography document**

DOM has several advantages over other available mechanisms for accessing XML documents. First, the DOM ensures proper grammar and well-formedness. Because the DOM transforms the text file into an abstract representation of a node tree, problems like unclosed tags and improperly nested tags can be completely avoided. Second, the DOM abstracts content away from grammar. The node tree created by the DOM is a logical representation of the content found in the XML file. It shows what

information is present and how it is related without necessarily being bound to the XML grammar. Third, the DOM closely mirrors typical hierarchical and relational database structures. This makes it very easy to move information between a database and an XML document using DOM. Last, the DOM simplifies internal document manipulation. Using DOM, access to any node in a document is a simple task without the need to perform a scan of the whole file. However, the DOM is very memory-intensive. It builds the node tree for the entire XML document in memory, which make it unsuitable for very large files.

### 3.4.2 The Simple API for XML (SAX)

The Simple API for XML (SAX) is an event-based interface for parsing XML documents. SAX is a Java interface, and, virtually, every Java XML parser supports SAX. With SAX, the parser sends events as it reads through an XML document. These events are related to elements in the XML documents being read. E.g., there are events for element start, element end and parsing errors etc. Figure 3.2 shows how the parser generates events as it progresses through the document.



**Figure 3.2 The SAX parser generate events**

The main advantage of using SAX is efficiency. Event-based interfaces are lower level than object-based interfaces. They give developers more control over parsing, which, on the other hand, means programming using SAX is more difficult. Another advantage is that a SAX parser consumes fewer resources than a DOM one, simply because it does not need to load the whole file into memory. The major limitation of SAX is that there is no random access to the document. Therefore, SAX is particularly popular with applications that process large files and for servers which process many documents simultaneously.

### **3.5 The XML Path Language (XPath)**

#### **3.5.2 XPath Overview**

XPath is a language for accessing parts of an XML document; it is used by XSLT and XQuery. The XML Path Language (XPath) provides a syntax for locating specific parts of an XML document effectively and efficiently. XPath operates on the abstract, logical structure of an XML document, rather than its surface syntax. XPath gets its name from its use of a path notation, as in URLs, for navigating through the hierarchical structure of an XML document.

In XPath, an XML document is viewed conceptually as a tree in which each part of the document is represented as a node. The nodes in this tree are similar to those in

the DOM mentioned in 3.4. XPath uses location-path expressions to specify how to navigate a node tree from one node to another. The following two examples show the usage of XPath against the XML document shown in Listing 3.1.

**Example1: /bib/book[@title='Data on the Web']/author**

**Returns the authors for the book whose title is “Data on the Web”.**

**Example 2: //book/publisher**

**Returns all books’ publishers**

XPath is intended primarily as a component that can be used by other specifications. Therefore, XPath relies on specifications that use XPath (such as XSLT and XQuery) to specify criteria for conformance of implementations of XPath and does not define any conformance criteria for independent implementations of XPath.

### **3.5.2 XPath as a Query Language**

XPath is a prototypical special-purpose XML query language. Originally designed as a helper language for XSL, XPath has enjoyed wider acceptance in part because of its compact syntax and ease-of-use. XPath provides ways to select nodes in an XML document based on simple criteria such as structure, position, or content.



Each XPath query is built from two basic components: navigation (using location steps) and selection (using filter expressions). The components can be combined in variety of ways to provide a rich expression language from only a few syntactic elements.

Navigation involves walking from one part of the XML document to another. For example, the XPath `/bib/book` consists of two location steps: one absolute step `/bib` which selects all top-level elements named `<bib>`, followed by a relative step `<book>` then selects the child elements named `<book>` of the current selection.

Selection can be performed either by selecting one node based on its position in the current selection, or by removing nodes from the current selection if they do not meet a Boolean condition. For example, `/bib/book[@title='Data on the Web']` selects only those book elements having an attribute title whose string value is 'Data on the Web'. `/bib/book[2]` selects the second book element. XPath provides the usual kinds of arithmetic and string operations, as well as functions for getting the current node's position or finding a node by XML ID.

### **3.6 Xquery: An XML Query Language**

XML is a versatile markup language, capable of labeling the information content of diverse data sources. As increasing amounts of information are stored, exchanged, and presented using XML, the ability to intelligently query XML data sources

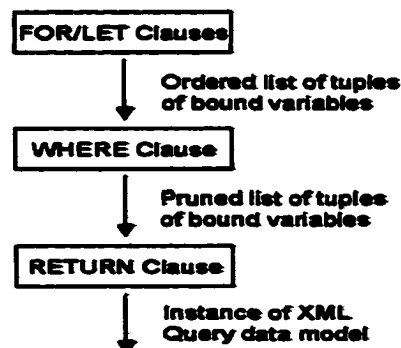
becomes increasingly important. XQuery is the working draft of W3C. It is designed to be a small, easy to implement language in which queries are concise and easily understood.

XQuery is derived from an XML query language called Quilt [CHAM99], which in turn borrowed features from several other languages. From XPath [XPath99] and XQL [ROBI98] it took a path expression syntax suitable for hierarchical documents. From XML-QL [DEUT98] it took the notion of binding variables and then using the bound variables to create new structures. From SQL it took the idea of a series of clauses based on keywords that provide a pattern for restructuring data (the SELECT-FROM-WHERE pattern in SQL). From OQL it took the notion of a functional language composed of several different kinds of expressions that can be nested with full generality. Quilt was also influenced by other XML query languages such as Lorel [SERG97] and YATL [CLUE99].

XQuery is a functional language in which a query is represented as an expression. XQuery supports several kinds of expressions, and the structure and appearance of a query may differ significantly depending on which kinds of expressions are used. The various forms of XQuery expressions can be nested and combined using arithmetic, logical and list operators, navigation primitives, function calls, operators like “sort”, conditional expressions, element constructors etc.

The XQuery data models XML documents as labeled trees with references. For navigating in a document, XQuery uses path expressions, whose syntax is borrowed from the abbreviated syntax of XPath. XPath is a notation for navigating along “paths” in an XML node tree. The evaluation of a path expression with respect to an XML document returns a list of information items, whose order is dictated by the order of elements within the document. XQuery provides a range predicate whose meaning is also based on order:  $E[\text{range } n \text{ to } p]$  evaluates the expression  $E$ , yielding a list, and selects from this list the sub-list of the  $n$ -th to  $p$ -th items. The precise semantics of path expressions is still under discussion; in this thesis, we consider a snapshot of the semantics for simple path expressions.

A powerful feature of XQuery is the presence of FLWR expressions (for-let-where-return). The *for-let* clause makes variables iterate over the result of an expression or binds variables to arbitrary expressions; the *where* clause allows specifying restrictions on the variables; and the *return* clause can construct new XML elements as output of the query. The overall flow of data in a FLWR expression is illustrated in Figure 3.3.



**Figure 3.3** Flow of data in a FLWR expression

The following two examples show the usage of XQuery. Example1: list books published by Addison-Wesley from the bibliography document, including their year and title. Listing 3.3 shows one possible solution in XQuery.

```
<bib>
{
  FOR $b IN document("bib.xml")/bib/book
  WHERE $b/publisher = "Addison-Wesley"
  RETURN
    <book year={ $b/@year }> ←
      { $b/title }
    </book>
}
</bib>
```

Note: "@" means attribute here.

**Listing 3.3 XQuery for Example1**

Expected result of the query is given in Listing 3.4.

```
<bib>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
  </book>
  <book year="1992">
    <title>Programming in the Unix environment</title>
  </book>
</bib>
```

**Listing 3.4 Result of the XQuery for Example1**

Example2: For each author in the bibliography, list the author's name and the titles of all books by that author, grouped inside a "result" element. Listing 3.5 shows one possible solution in XQuery. This example shows the nested XQuery expressions.

```

<results>
{
  FOR $a IN distinct(document("bib.xml")//author)
  RETURN
    <result>
      { $a }
      {
        FOR $b IN document("bib.xml")/bib/book[author = $a]
        RETURN $b/title
      }
    </result>
}
</results>

```

**Listing 3.5 XQuery for Example2**

This example shows the nested XQuery expressions. Expected result of the query is given in Listing 3.6.

```

<results>
  <result>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
    <title>TCP/IP Illustrated</title>
    <title>Programming in the Unix environment</title>
  </result>
</results>

```

**Listing 3.6 Result of the XQuery for Example2**

In summary, XQuery is designed to support queries against a broad spectrum of information sources. The versatility of XQuery will help XML to realize its potential as a universal medium for data interchange. In this thesis, XQuery is used to query

XML documents and define views on top of XML data. However, Data definition facilities for persistent views are not included in XQuery specification, so views must be built programmatically in this thesis.

### **3.7 Extensible Stylesheet Language Transformation**

An alternative way to query XML documents is using Extensible Stylesheet Language (XSL). XSL is the current W3C Recommendation for expressing stylesheets. Although primarily targeted towards presentation, XSL has facilities that could serve as a basis for an XML query language. It is an XML-based language designed to transform an XML document into another XML document with a different DTD or into documents of other formats such as HTML or PDF. XSL Transformations (XSLT) is the part of XSL for transforming XML documents into other XML documents.

A transformation expressed in XSLT describes rules for transforming a source tree into a result tree. The transformation is achieved by associating patterns with templates. A pattern is matched against elements in the source tree. A template is instantiated to create part of the result tree. The result tree is separate from the source tree. The structure of the result tree can be completely different from the structure of the source tree. In constructing the result tree, elements from the source tree can be filtered and reordered, and arbitrary structure can be added. XSLT makes use of the expression language defined by XPath for selecting elements for processing, for

conditional processing and for generating text [XSLT]. Figure 3.4 illustrates how a XSLT processor works

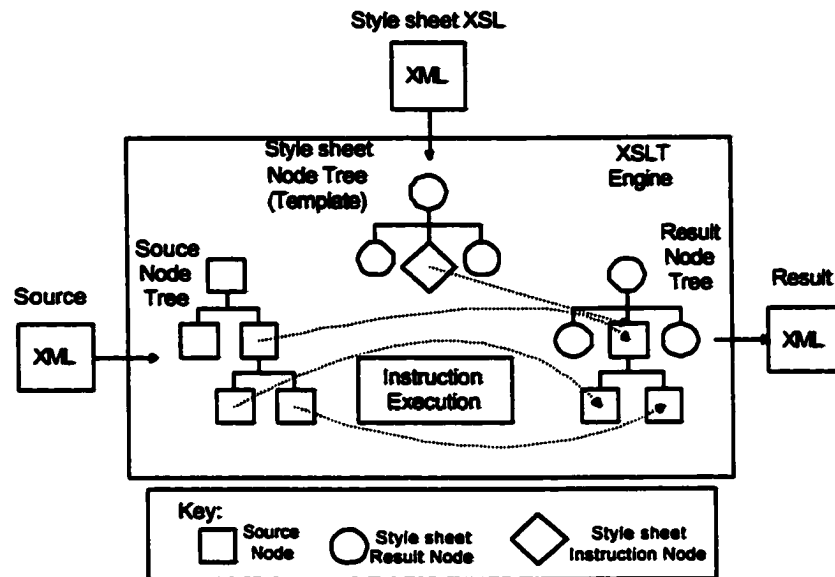


Figure 3.4 XSLT processor working (modified from [ANDE99] page375)

XSLT operates on the document model not the syntax. Both the source and destination formats are applications of XML, and the underlying structure of both is a tree. In addition, the XSLT script is an XML document, thus it too can be represented as a tree. So the XSLT processor holds three trees. The processor goes through the source tree, starting with the root, and looks for a matching template in the style sheet tree. When it finds one, it uses the rules in the template to write an abstract representation of the result into the result tree. Then it moves through the source document, node by node, lead by the XSLT instruction `<xsl:apply-template>`, looking for a match in the style sheet. At last, the result tree is translated into an XML document.

Overall, XSLT is a very flexible language, capable of performing rather complicated queries. However, XSLT requires well-formed XML documents, and it cannot operate on XML fragments. XSLT is very hard to optimize; in fact, every implementation requires the entire document to be loaded into memory before the XSLT can be executed.



## **4 Investigation of CORBA**

### **4.1 Overview of CORBA**

The Common Object Request Broker Architecture (CORBA) is the product of the Object Management Group OMG. The purpose of OMG is to create standards allowing for the interoperability and portability of distributed object-oriented applications. The CORBA specification includes an Object Request Broker (ORB), which is the protocol that enables the seamless interaction between distributed components; Object services, which facilitate standard client/server interaction with capabilities such as naming, event-based synchronization, and concurrency control; and the Interface Definition Language (IDL), which defines the object interfaces within the CORBA environment. By providing an object-oriented architecture with object interface inheritance, ORB interoperability and platform independence, CORBA meets one of the most essential requirements of modern computing, which is to maximize the portability, reusability and interoperability of software.

Distributed CORBA components, modeled as business objects, are an excellent fit for distributed architectures. They provide scalable and flexible solutions for client/server environments and for the Internet and intranets. Business objects, packaged as CORBA components, can be naturally decomposed and split across multiple tiers to meet an application's need.

## **4.2 ORB Core**

**CORBA ORB connects a client application with the objects it wants to use. The ORB delivers requests to objects and returns responses to the clients making the requests. The key feature of the ORB is the transparency of how it facilitates communication between distributed components/objects. The ORB takes care of the details of locating the object, routing the request, and returning the result. Ordinarily, object location, implementation, execution state, and communication mechanisms are hidden from the client. This feature allows application developers to worry more about their own application domain issues and less about low-level distributed system programming issues.**

**To make a request, the client specifies the target object by using an object reference. When a CORBA object is created, an object reference for it is also created. References are valid in the whole system and can thus be passed from one node to another. Clients can obtain object references in several different ways: object creation**

- a client can create a new object in order to get an object reference; directory service**
- a client can invoke a lookup service to obtain object references; convert to string and back**
- an application can ask the ORB to turn an object reference into a string, and this string can be stored into a file or a database, and later, this string can be retrieved and turned back in to an object reference by the ORB. Objects are not tied to**

a client or server role; they can act in turn as client and as server (peer-to-peer communication).

### 4.3 CORBA Invocations

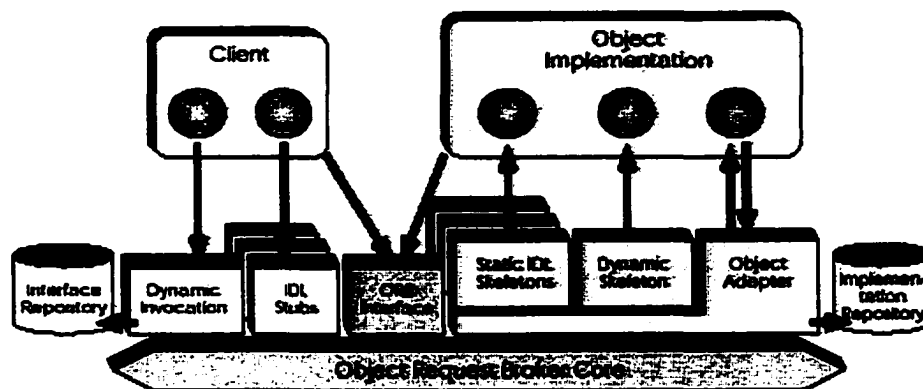
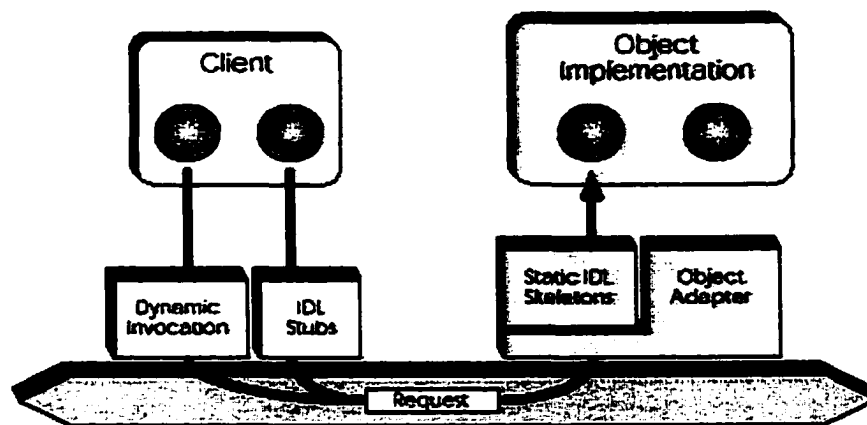


Figure 4.1 The CORBA Architecture ([BARA] page 3)

The CORBA architecture is depicted in figure 4.1. The *client IDL stubs* provide the static interfaces to object services. They define how clients invoke corresponding services on remote servers. The *Dynamic Invocation Interface (DII)* lets the clients choose at run-time the operation invoked through a set of standard APIs. In contrast to the static stubs, the *DII* is independent of the target object's interface. The *server IDL stubs*, also called *skeletons*, provide static interfaces to each service exported by servers. The *Dynamic Skeleton Interface (DSI)* provides a run-time binding mechanism for servers that do not have static IDL skeletons to handle any request dynamically. The *Object Adapter* accepts requests for service and provides a run-time environment for instantiating server objects, passing requests to them, and assigning

them *object references*. The object implementation may call the object adaptor and the ORB while processing a request. The *Implementation Repository* provides run-time information about the classes the server supports, the objects that are instantiated, and their identifiers.

Figure 4.2 illustrates how a request is performed through the ORB, using either the *DII* or the *client IDL stubs*. Requests are handled at the server site by the *Object Adapter*, and given to the server's *IDL skeleton*.



**Figure 4.2 Performing a Request with CORBA ([BARA] page 3)**

## 4.4 OMG Interface Definition Language

An object's interface specifies the operations and types which the object supports and thus defines the requests that can be made on the object. Interfaces of CORBA objects are defined in the OMG Interface Definition Language (IDL). An important feature of OMG IDL is its language independence. Owing to the fact that OMG IDL

is a specification language, not a programming language, it forces interfaces to be defined separately from object implementations. This allows objects to be constructed using different programming languages and still communicate with one another. Language-independent interfaces are important within heterogeneous systems, since not all programming languages are supported or available on all platforms. OMG IDL interfaces can inherit from one or more other interfaces. This makes it possible to reuse existing interfaces when defining new services. Interface inheritance is very important in CORBA. It allows the system to be open for extension while keeping it closed for modification, which is called the Open-Closed Principle. This allows object references for derived interfaces to be substituted anywhere object references for base interfaces are allowed.

OMG IDL provides a set of types that are similar to those found in a number of programming languages. It provides basic types such as long, double, and boolean, constructed types such as struct and union, and template types such as sequence and string. Types are used to specify the parameter types and return types for operation. To define exceptional conditions that may arise during the course of an operation, OMG IDL provides exception definitions. The OMG IDL type system is sufficient for most distributed applications, yet at the same time it is minimal. Keeping OMG IDL as simple as possible means that it can be used with many more programming languages than it could if it contained types that could not be realized in some popular programming languages. Given the inevitable heterogeneity of distributed object

systems, the simplicity of OMG IDL is critical to the success of CORBA as a protocol and integration technology.

In addition to generating programming-language types, OMG IDL language compilers and translators also generate client-side stubs and server-side skeletons. A stub is a mechanism that effectively creates and issues requests on behalf of a client, while a skeleton is a mechanism that delivers requests to the CORBA object implementation. Stubs are sometimes called proxies because the stub essentially is a stand-in within the local process for the actual target object. The stub works directly with the client ORB to marshal the request. Once the request arrives at the target object, the server ORB and the skeleton cooperate to unmarshal the request and dispatch it to the object. Once the object completes the request, any response is sent back the way it came. Figure 2 shows the positions of the stub and skeleton in relation to the client application, the ORB, and the object implementation.

## **4.5 Benefits of CORBA**

CORBA provides interoperability across programming languages. Interfaces to CORBA objects are written in the programming-language-independent notation IDL. These IDL interfaces are then mapped to language-specific interfaces, according to rules standardized by the OMG, known as language mappings. Standardized mappings exist for Java, C, C++, COBOL, Ada, and Smalltalk. De-facto standard mappings exist for other popular languages such as TCL, Perl, Delphi, and Python.

**As a result, CORBA objects -- servers and their clients can be written in any language. It is not abnormal for a CORBA-based distributed application to consist of pieces that are written in completely different languages.**

**CORBA provides Interoperability across platforms. The protocol that CORBA uses to allow distributed objects to communicate with each other, is specified by the OMG, and is known as IIOP. CORBA implementations from multiple vendors can interoperate with each other. Different CORBA products can be used on different platforms. Most popular ORB products have themselves been ported to multiple hardware platforms and operating systems. For example, VisiBroker products are available on a number of platforms -- Windows, Solaris, Linux, HP-UX, SGI, IBM's AIX, Digital Unix, IBM OS/390, and others.**

**CORBA provides independence from CORBA vendors. Standardized APIs (IDL, language mappings, etc.) and the standardized wire protocol(IIOP) together provide developers and their customers with independence from the CORBA products and vendors. The availability of interoperable CORBA ORBs from a number of different vendors means that developers do not depend on any single vendor so long as they do not use vendor-specific add-on features.**

**CORBA provides source portability. When the CORBA standards were initially released, source portability (the ability to recompile and run existing CORBA applications with a different CORBA product, without making changes) was**

**promised, but not delivered. Source portability for pure-clients was acceptable. With some care, one could write CORBA clients that would port across CORBA products. However, source portability for servers was not acceptable. A key server-side API, known as the BOA, was heavily under-specified and ambiguous in many places. This led CORBA vendors to make incompatible assumptions and extensions, sacrificing server-side source portability. A new server-side specification, known as the Portable Object Adapter (POA), corrects this problem. In addition to being complete and unambiguous, it contains many more features than the old BOA specification. The POA reinforces the fact that CORBA is the ideal substrate for most distributed applications.**

**With these benefits, CORBA is adopted in our approach to facilitate the communications between distributed applications.**



## **5 Constructing the XML Document for Storing the Shared Data**

### **5.1 Design XML Structure from Scratch**

#### **5.1.1 XML Content Models**

Using the terminology usual in databases, it is possible to view XML as the language for data modeling. A well-formed XML document (or a set of such documents) is then a database and the associated Document Type Definition (DTD) is its schema. The data modeling of the XML document for storing the shared data has a direct and significant impact on performance, document size, and code size. As we have mentioned in Chapter 3, when using a DTD to define the structure of an XML vocabulary, there are five possible content models for elements: element-only content, mixed content, text-only content, the EMPTY model, and the ANY model.

Element-only content is used when elements only contain other elements. For example, the following content model is element-only:

**<!ELEMENT Invoice (Customer, LineItem+)>**

Here, the element Invoice can contain a Customer element, followed by one or more LineItem elements. This structure provides the cleanest way to contain one structure inside another. This is a good structure for the nesting of elements.

In the text-only content model, elements may only contain text strings. An example of the text-only content model might be:

**<!ELEMENT Customer (#PCDATA)>**

Using text-only elements is one way to include single data values in XML document.

However, we could also use attributes, which can have advantages over this method.

In the mixed content model, elements may contain zero or more instances of a list of elements, in any order, along with any amount of text in any position. An example of the mixed content model might be:

**<!ELEMENT Invoice (#PCDATA | LineItem | Customer)\*>**

This model is not suitable for modeling data because the allowable subelements might appear at any point within the element, and any number of times. This makes it very difficult to map to data elements, and makes writing code to handle the document a nightmare. The use of the mixed content model for data should be avoided.

In the EMPTY content model, an element cannot contain anything at all; it must be expressed either as the empty element tag, or as a start tag followed immediately by an end tag. The following is an example of the empty content model:

**<!ELEMENT Customer EMPTY>**

This content model will come in useful when the only additional information associated with an element is at the data value level. For example, if we have a Customer element that only has a Name associated with it, we could use the empty

content model for the Customer element and represent the Name value as attribute (refer to example on page 44).

In the ANY content model, any element or text may appear inside the element when defined this way, in any order. So, for this example:

**<!ELEMENT Customer ANY>**

An example document might be:

**<Customer>Kevin Willians</Customer>**

or:

**<Customer>**

**<Customer>**

**<Customer>Kevin Willians</Customer>**

**</Customer>**

**<Customer>**

Like the mixed content model, this content model is too permissive for data. Without some idea of what structure might appear, and in what order, operating upon the data will be extremely difficult. For this reason, the use of the ANY content model for data should also be avoided.

### **5.1.2 Modeling data values**

The first way data values are represented in XML documents is by using elements. For text-only elements, a <Customer> element might be defined like this:

```
<!ELEMENT Customer (firstName, lastName, address, city,  
                    province, country, postalCode)>  
<!ELEMENT firstName (#PCDATA)>  
<!ELEMENT lastName (#PCDATA)>  
<!ELEMENT address (#PCDATA)>  
<!ELEMENT city (#PCDATA)>  
<!ELEMENT province (#PCDATA)>  
<!ELEMENT postalCode (#PCDATA)>
```

**Listing 5.1** Definition of Customer using elements

When representing data in an XML document, any element that is defined as having text-only content using the #PCDATA keyword will correspond to a column in a relational database.

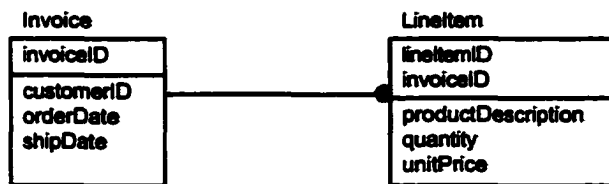
Another way of representing data values in XML documents is with attributes. In this approach, elements that represent tables have attributes associated with them that represent columns:

```
<!ELEMENT Customer EMPTY>  
<!--ATTLIST Customer  
    firstName CDATA #REQUIRED  
    lastName CDATA #REQUIRED  
    address CDATA #REQUIRED  
    city CDATA #REQUIRED  
    province CDATA #REQUIRED  
    postalCode CDATA #REQUIRED-->
```

**Listing 5.2** Definition of Customer using attributes

### 5.1.3 Modeling relationships among elements

To associate groups of data with other groups of data, we need to add relationships between these groups. In a relational database, this can be modeled using foreign key constraints. For example, we have the following two tables:



**Figure 5.1** Relationship between Invoice and LineItem

The foreign key on the LineItem table is the primary key on the Invoice table. One invoice can have one or more lineItems. In XML, one-to-one and one-to many relationships are best represented by containment. One possible design would be:

```
<ELEMENT Invoice (LineItem+)>
<ATTLIST Invoice
  orderDate CDATA #REQUIRED
  shipDate CDATA #REQUIRED>
<ELEMENT LineItem EMPTY>
<ATTLIST LineItem
  productDescription CDATA #REQUIRED
  quantity CDATA #REQUIRED
  unitPrice CDATA #REQUIRED>
```

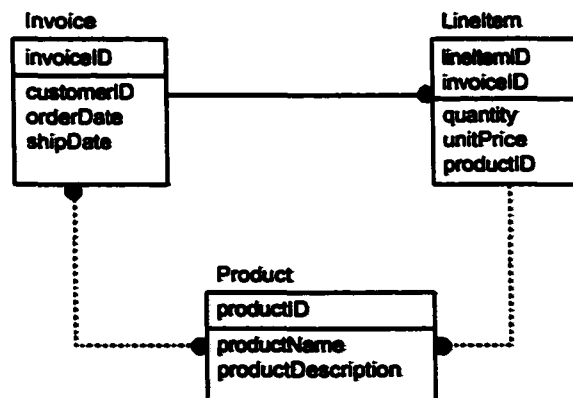
**Listing 5.3** Modeling relationships by containment

An example of a document with this structure looks like this:

```
<Invoice
  orderDate="7/23/2000"
  shipDate="2/28/2000">
  <LineItem
    productDescription="Widgets"
    quantity="17"
    unitPrice="0.10"/>
  <LineItem
    productDescription="Grommets"
    quantity="22"
    unitPrice="0.05"/>
</Invoice>
```

**Listing 5.4** Example of containment

Here, it is clear that the LineItem information is part of the Invoice. Containment is best suited for modeling one-to-one and one-to-many relationships. However, it is possible to have more complex relationships in applications than can be represented with containment alone. If we extend the previous example by adding a Product table, and the following are the tables:



**Figure 5.2** Relationships among Invoice, LineItem and Product

In this case, there is a many-to-many relationship between Invoice and Product. Many products may appear on one invoice, and one product may appear on many invoices. In a relational database, this is expressed through the relating table LineItem. An invoice may have many line items, and a product may appear on many line items. In XML, ID/IDREF pairs can be used to model many-to-many relationship. The above example can be modeled in XML as following:

```
<!ELEMENT OrderData (Invoice+, Product+)>
<!ELEMENT Invoice (LineItem+)>
<!ATTLIST Invoice
    orderDate CDATA #REQUIRED
    shipDate CDATA #REQUIRED>
<!ELEMENT LineItem EMPTY>
<!ATTLIST LineItem
    productIDREF IDREF #REQUIRED
    quantity CDATA #REQUIRED
    unitPrice CDATA #REQUIRED>
<!ELEMENT Product EMPTY>
<!ATTLIST Product
    productID ID #REQUIRED
    productName CDATA #REQUIRED
    productDescription CDATA #REQUIRED>
```

**Listing 5.5 Modeling relationships using ID/IDREF pairs**

In this way, repetition of information can be avoided. If we nested Product inside, LineItem, the product information would have to be repeated for every invoice where it appears. The following XML document is an example instance of the structure defined in Listing 5.5:

```

<OrderData>
  <Invoice orderDate="7/23/2000" shipDate="7/30/2000">
    <LineItem
      productIDREF="prod1"
      quantity="17"
      unitPrice="0.10" />
    <LineItem
      productIDREF="prod2"
      quantity="22"
      unitPrice="0.06" />
  </Invoice>
  <Invoice orderDate="7/25/2000" shipDate="8/2/2000">
    <LineItem
      productIDREF="prod2"
      quantity="30"
      unitPrice="0.05" />
    <LineItem
      productIDREF="prod3"
      quantity="19"
      unitPrice="0.15" />
  </Invoice>
  <Product
    productID="prod1"
    productShortName="Widgets"
    productDescription="Rubberized Brown Widgets" />
  <Product
    productID="prod2"
    productShortName="Grommets"
    productDescription="Vulcanized Orange Grommets" />
  <Product
    productID="prod3"
    productShortName="Sprockets"
    productDescription="Anodized Silver Sprockets" />
</OrderData>

```

**Listing 5.6 Example of ID/IDREF pairs**



## **5.2 Migrating a Database to XML**

**Relational databases are a mature technology, which has enabled users to model complex relationships between data that they need to store. As we have discussed, there are a number of reasons why data stored in databases need to be exposed as XML: sharing business data with other systems; interoperability with incompatible systems; exposing legacy data to applications that use XML; and business-to business transactions. The following steps show how to migrate a database to XML.**

**Step 1. Choose the data to include. Based on the business requirements the XML document will be fulfilling, decide which tables and columns from the relational database will need to be included in the XML document.**

**Step 2. Create a root element. Create a root element for the document. Add the root element to the DTD, and declare any attributes of that element that are required to hold additional semantic information. For example, we might want to add a source attribute, so that the users know where the information comes from. Root element names should describe their content.**

**Step 3. Model the content tables. Create an element in the DTD for each table that has been chosen to model. Declare these elements as EMPTY temporarily.**

**Step 4. Model the non-foreign key columns. Create an attribute for each column that has been chosen to be included in the XML document except foreign key columns. These attributes should appear in the !ATTLIST declaration of the element corresponding to the table in which they appear. Declare each of these attributes as CDATA, and declare it as #IMPLIED or #REQUIRED depending on whether the original column allowed nulls or not.**

**Step 5. Add ID attributes to the elements. Add an ID attribute to each of the elements in the XML structure with the exception of the root element. Use the element name followed by ID for the name of the new attribute to avoid name collisions. Declare the attribute type ID, and #REQUIRED.**

**Step 6. Add element content to root elements. Add a child element or elements to the allowable content of the root element for each table that models the type of information that needs to be represented in the XML document.**

**Step 7. Add relationships through containment. For each one-to-one or one-to-many relationship, add the child element as element content of the parent element with the appropriate cardinality.**

**Step 8. Add relationships using IDREF/IDREFS. For each many-to-many relationship, add an IDREF or IDREFS attribute to the element on the parent side of**

the relationship, which points to the ID of the element on the child side of the relationship.

**Step 9. Add missing elements.** For any element that is only pointed to in the structure created so far, add that element as allowable element content of the root element. Set the cardinality suffix of the element being added to \*.

**Step 10. Remove unwanted ID Attributes.** Remove ID attributes that are not referenced by IDREF or IDREFS attributes elsewhere in the XML structures.

### **5.3 Transforming from Flat Files to XML**

Flat files store data in a way that is generally specific to the application using them and are commonly encountered when working with legacy systems. Flat files can vary widely. There are many ways to create a flat file. There are a couple of common issues that need to be considered when moving data from flat files to XML documents.

The most obvious difference between XML documents and flat files is the level of normalization. Flat files are usually completely un-normalized. Since a well-designed XML structure will be fairly normalized, the code for the transformation will have to normalize data when moving from flat files to XML documents.

Another issue commonly encountered with flat files is the format of data. Many legacy systems, due to size constraints or other concerns, use specific formatting schemes that may not be easily understood by users of the data. A library of routines should be built up to decipher the values.

When transforming flat files to XML, there are a couple of approaches. First, perform file parsing to extract the data from the flat file – reading the file a line at a time and breaking it apart into its individual components. Then, use manual serialization, SAX, or the DOM to produce the XML output.

Using manual serialization, the XML document, including all of the tags and other text that goes with an XML document, is created on the fly by appending to a string. This approach has relatively small memory consumption, but tends to be error-prone. For example, a start tag could be produced accidentally without a corresponding end tag. The other problem with this approach is it forces the information to be written to the target in the order prescribed by the target. Objects cannot be appended to the tree at will. This requires some more sophisticated parsing approaches to obtain the desired output, especially if the target document has many-to-many relationships. Traditional I/O functions can be used to generate the output string or file.

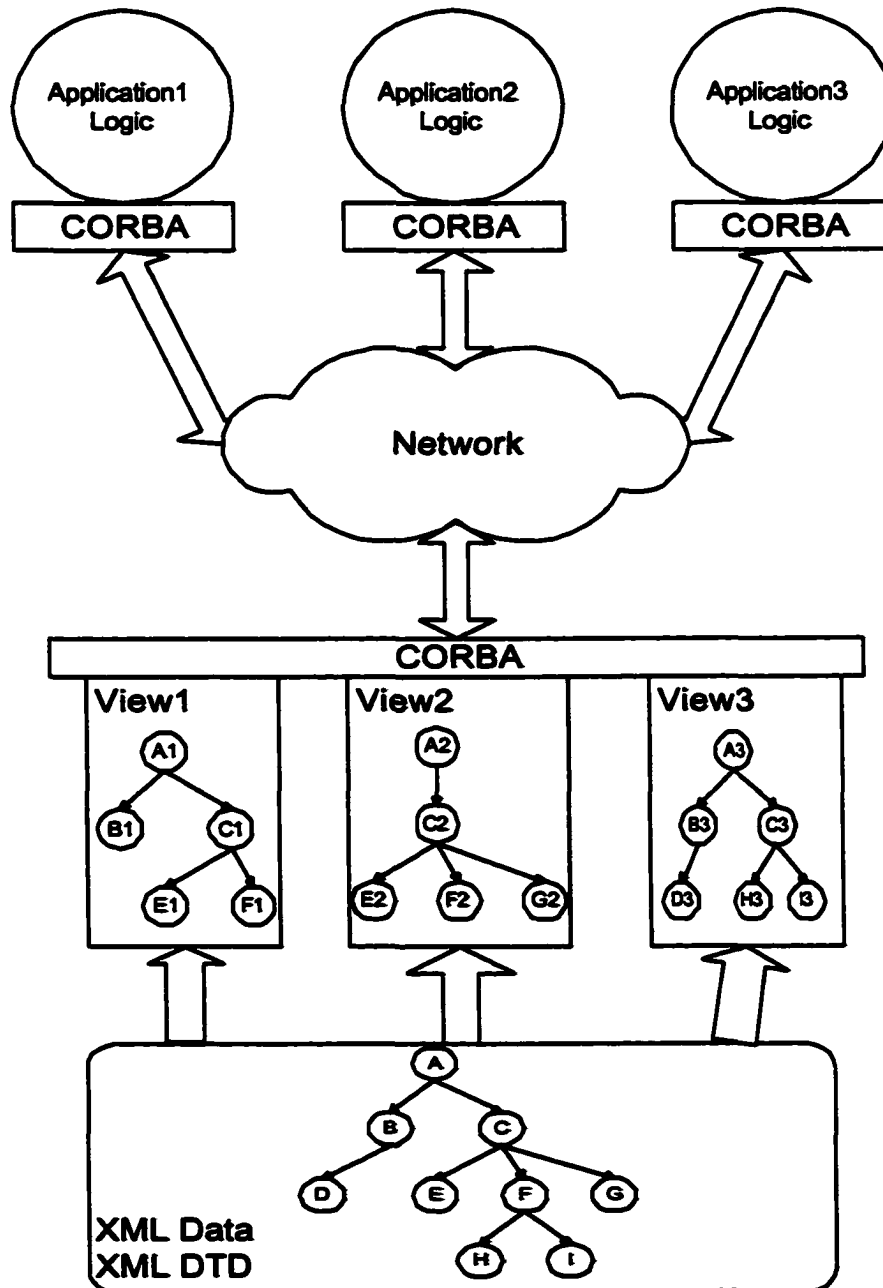
Using SAX, a SAX handler is initialized and a stream of events are sent to it, causing it to generate an XML document. But, the start and end element events still need to be manually generated to send to the SAX event handler. This approach is not much

better than manual serialization. SAX also requires that the document be serialized in the order mandated by its DTD. While this approach does not afford much better control of XML document serialization than creating the document manually, it is often a good choice when creating very large XML documents.

In the DOM approach, a document tree is created using an implementation of the XML DOM. This approach tends to consume more memory than a simple serialization approach, but is much less error prone. There is no risk of accidentally omitting a start or end tag. The random-access nature of the DOM also allows elements to be added to the result document tree as they are processed, rather than being required to be cached and written in the order required by the target document. Another advantage of using the DOM is the ease of coding. Therefore, this approach is most suitable for most applications without particular performance and memory requirements.

## 6 Different Approaches for Generating XML Views

### 6.1 Architecture of Distributed Systems for Data-independence



**Figure 6.1** Architecture of distributed systems for data-independence

Figure 6.1 shows the architecture of our solution to the problem of data-independence of distributed applications that access shared data. There are several distributed applications in the system, Applications 1, 2 and 3, and they may be written in different languages and reside on different platforms. These applications need to access the shared XML data, which is distributed at another location on the network.

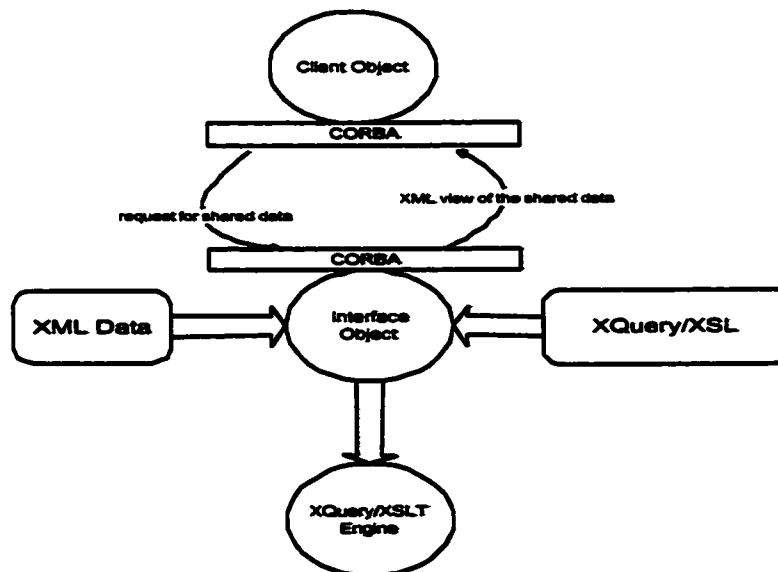
For the distributed applications to communicate with each other and each application to access the shared data easily, CORBA is used to wrap the applications and the data. As we have discussed in Chapter 4, CORBA allows distributed applications to interoperate, regardless of what language they are written in, or where these applications reside. The cost of developing applications is reduced, because the complexity of the communication between different applications is taken care of by CORBA.

To improve the data-independence of the shared XML data, different XML views are built on top of the XML data for different applications. Only the data or structures that are needed by an application are selected to compose the view corresponding to that specific application. When the structure of the shared data changes, the views, from the applications perspective, remain the same. Thus, the applications accessing the shared data through the views are shielded from the changes on the structure of the shared data.

We propose two approaches to generate and support the XML views: the XQuery/XSL approach and the XPath Mapping approach.

## 6.2 The XQuery/XSL Approach

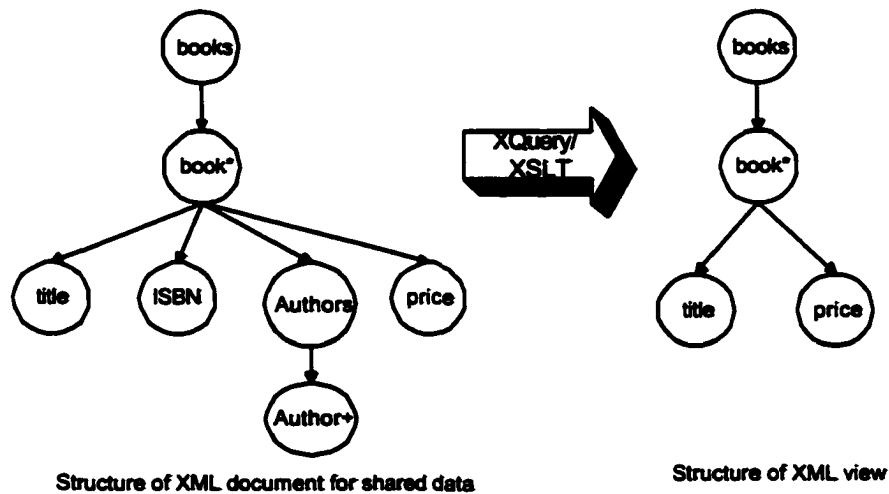
In this approach, a XML view for the shared data is defined using a single XQuery or XSL. The view could be supported by building an object as an interface of the shared data. When a client needs to access the shared data, the client sends a request to the interface object; this interface object calls the XQuery/XSLT engine to generate the XML view against the shared data using the XQuery or XSL predefined for the client, and sends the view back to the client. The view is simply another XML file possibly much smaller than the original shared data file. Then the client can further process the XML view document using the DOM or SAX for presentation or other purposes. This process is depicted in Figure 6.2.



**Figure 6.2** Support XML view using single XQuery/XSL



When the structure of the shared data changes, only the XQuery or the XSL file used to define the XML view needs to be modified to maintain the view. The view generation is carried out on the server side, so the XQueries and the XSL files to support the views are stored on the server side. Nothing on the client applications needs to be changed. The cost of the modification can be minimized. For example, if the structures of the shared data and the view for an application look like this:



**Figure 6.3** An example of XML data and view

The XQuery that defines this view would be:

```
<books>
{
  FOR $b IN document(book.xml)/books/book
  RETURN
    <book>
      {$b/title}
      {$b/price}
    </book>
}
</books>
```

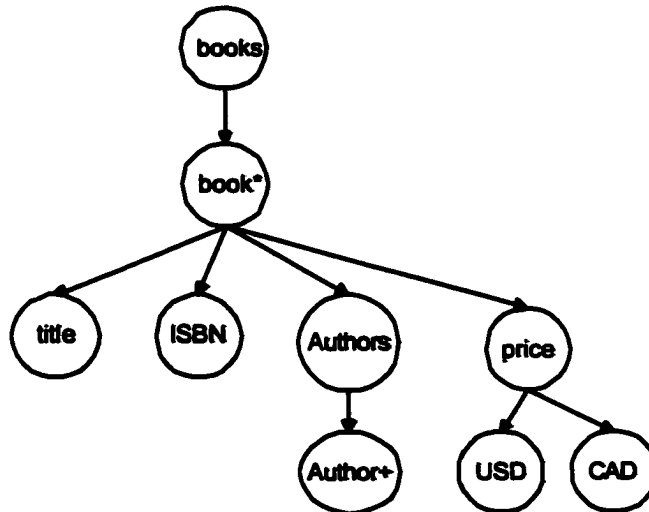
**Listing 6.1** Example XQuery to generate XML view

Using XSL to define the view is also simple:

```
<xsl:stylesheet xmlns:xsl =  
    http://www.w3.org/1999/XSL/Transform version="1.0" >  
  <xsl:output method ="xml"/>  
  <xsl:template match = "/">  
    <books><xsl:apply-templates/></books>  
  </xsl:template>  
  <xsl:template match ="book">  
    <book>  
      <title><xsl:value-of select = "title"/></title>  
      <price><xsl:value-of select = "price"/></price>  
    </book>  
  </xsl:template>  
</xsl:stylesheet>
```

**Listing 6.2** Example XSL to generate XML view

Suppose the price here is in CAD (Canadian dollar). Now if another application using the shared data needs the price information in both USD (US dollar) and CAD, the structure of the shared data may change to:



**Figure 6.4** Structure of the XML document after change

To maintain the views defined for different applications, we only need to modify the XQueries and XSL files that define the views. For the example view shown in Figure 6.3, the XQuery is modified as follows: (Text in bold is modified)

```
<books>
{
  FOR $b IN document(book.xml) /books/book
  RETURN
    <book>
      {$b/title}
      <price>
        {$b/price/CAD/text()}
      </price>
    </book>
}
</books>
```

**Listing 6.3** Modifying the XQuery

As we can see from this example, since the structure of the price element is changed in the XML document for the shared data, we only need to go straight to the part that generates the price element of the XML view and do the modification accordingly. Locating the points in the XQuery that need to be modified is a fairly simple task, and the modification is also straightforward.

Similarly the XSL file can also be modified easily: Only one XPath expression (the bold text) is modified in this example.

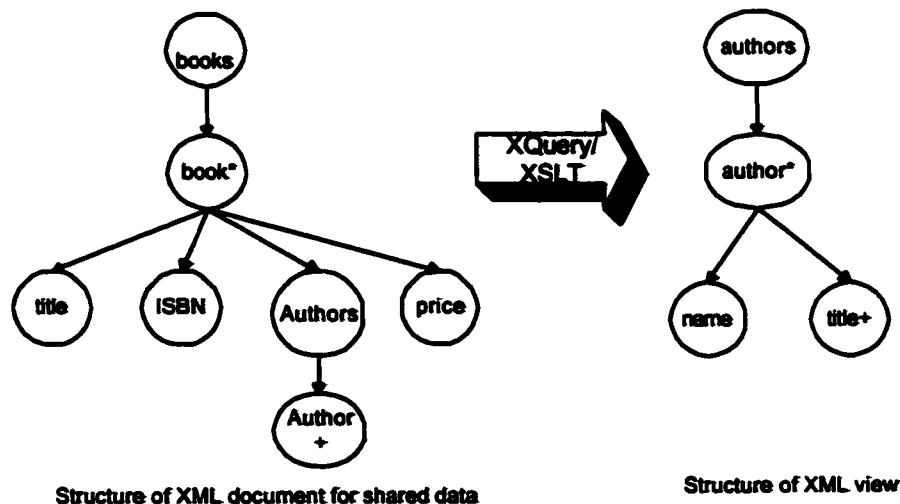
```

<xsl:stylesheet xmlns:xsl =
    http://www.w3.org/1999/XSL/Transform version="1.0" >
  <xsl:output method ="xml"/>
  <xsl:template match = "/">
    <books><xsl:apply-templates/></books>
  </xsl:template>
  <xsl:template match ="book">
    <book>
      <title><xsl:value-of select = "title"/></title>
      <price><xsl:value-of select = "price/CAD" /></price>
    </book>
  </xsl:template>
</xsl:stylesheet>

```

**Listing 6.4** Modifying the XSL

This approach can generate views with totally different structure compared to the structure of the source data. For example, the view depicted in Figure 6.5 lists each author in books, and the titles of all books by that author.



**Figure 6.5** An example of a view with major difference in structure

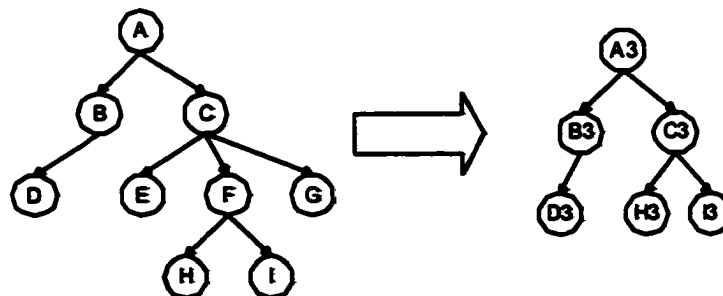
The XQuery that defines this view would be:

```
<authors>
{
  FOR $a IN distinct(document(book.xml)//author)
  RETURN
    <author>
      <name> {$a/text()} </name>
      {
        FOR $b IN document(book.xml)/books/book[author=$a]
        RETURN $b/title
      }
    </author>
}
</authors>
```

**Listing 6.5** XQuery to generate XML view with different structure

### 6.3 The XPath Mapping Approach

In this approach, the views of the shared data do not physically exist. Instead, they are logical views defined by mappings between the XPath expressions for each node in the structures of the views and corresponding XPath expressions against the source data. For example, the view3 shown in Figure 6.1 can be defined as the following mapping:



View Path	Source Path
/A3	/A
/A3/B3	/A/B
/A3/C3	/A/C
/A3/B3/D3	/A/B/D
/A3/C3/H3	/A/C/F/H
/A3/C3/I3	/A/C/F/I

When a client sends an XPath query against a view of the shared data, the query is translated into another query using a mapping function according to the source-to-view path mapping defined as the view. The new query is then executed against the source data, and the result is returned to the client. Therefore, the views can be supported by a mapping function, which performs the query transformation. To simplify the mapping function, only absolute paths are used in the mapping. The following are two examples of query transformation:

**/A3/C3[H3='value1']/I3[2] ➔ /A/C/F[H='value1']/I[2]**

**//B3[D3='value2'] ➔ /A/B[D='value2']**

When the structure of the shared data changes, only the corresponding source paths in the mapping table need to be changed. For example, if an element X is added between A and C, all source paths containing steps /A/C will be changed to paths containing

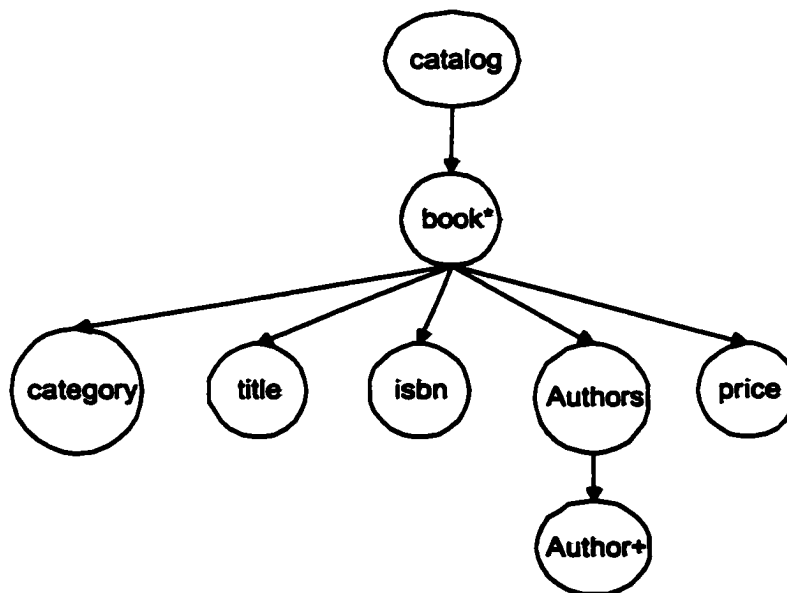
**/A/X/C. The modification is easy, and it can be done automatically by writing a simple function to replace strings on the source paths.**

**In the above example, only absolute and simple paths are used in the mapping table.**

**If there is a major change to the source data, we may need more complicated XPath expressions, including relative paths and selections, to define the views. To handle these complex XPath expressions, a more sophisticated view-to-source query transformation function must be built.**

## 7 Guidelines for Systematically Generating XML Views for the Initial Data

We will walk through the design of a prototype system to demonstrate our approaches. The prototype system is a simple Book Catalog System. The structure of the shared data can be depicted as following:



**Figure 7.1** Structure of the shared data for the Book Catalog System

Suppose one of the distributed applications accessing the shared data answers queries about price information of a book according to the title or the ISBN of the book.



## 7.1 Guidelines for the XQuery/XSL Approach

Step 1. Construct the XML document for storing the shared data. Using the methods discussed in chapter 5, design the XML document from scratch or migrate the data from databases or flat files according to the system requirements. The DTD for the shared XML data of the Book Catalog System can be constructed as following:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT catalog ( book )+ >
<!ELEMENT book ( isbn, authors, price ) >
<!ATTLIST book title CDATA #REQUIRED
              category (fiction|nonfiction) "fiction">
<!ELEMENT authors ( author )+ >
<!ELEMENT isbn ( #PCDATA ) >
<!ELEMENT author ( #PCDATA ) >
<!ELEMENT price ( #PCDATA ) >
```

**Listing 7.1 DTD for the shared data for Book Catalog System**

A sample document for the shared data is shown in Listing 7.2.

```

<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <book category="nonfiction" title="JAVA">
    <isbn>0-13-028417-3</isbn>
    <authors>
      <author>Harvey Deitel</author>
      <author>Paul Deitel</author>
      <author>Ted Lin</author>
    </authors>
    <price>85.00</price>
  </book>
  <book category="fiction" title="Love Story">
    <isbn>0-13-023431-5</isbn>
    <authors>
      <author>John Lee</author>
    </authors>
    <price>19.99</price>
  </book>
</catalog>

```

**Listing 7.2** A sample shared XML data for Book Catalog System

Step 2. Define the application interface using CORBA IDL. This interface includes a function that generates XML view documents. The IDL for the prototype system is defined in view.idl as follows:

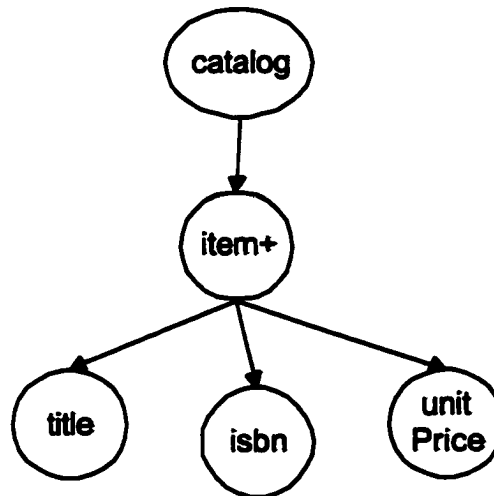
```

module XMLView{
    interface View {
        string generateView(in string viewName);
    };
};

```

For different applications, the function generateView will generate different XML views according to the input viewName.

Step 3. Construct the structure of the XML view for each application according to the application logic. For an example application, which needs to answer queries about price information of a book according to the title of the book, the structure of the XML view can be constructed as the following tree:



**Figure 7.2** Structure of the view for Book Catalog System

The DTD for this XML view is shown in Listing 7.3.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT catalog ( item )+ >
<!ELEMENT item ( title, unitPrice ) >
<!ELEMENT unitPrice ( #PCDATA ) >
<!ELEMENT title ( #PCDATA ) >
<!ELEMENT isbn ( #PCDATA ) >
```

**Listing 7.3** DTD for the view for Book Catalog System

**Step 4. Build XQueries or XSL files to define the XML views against the shared data.** Because Xquery is still the working draft of W3C, there is no mature engine for XQuery. We use the XSL approach in our prototype system. The XSL file for generating the view is shown in Listing 7.4.

```
<xsl:stylesheet xmlns:xsl =  
    http://www.w3.org/1999/XSL/Transform version="1.0"  >  
  <xsl:output method ="xml"/>  
  <xsl:template match ="/">  
    <catalog><xsl:apply-templates/></catalog>  
  </xsl:template>  
  <xsl:template match ="book">  
    <item>  
      <title><xsl:value-of select = "@title"/></title>  
      <isbn><xsl:value-of select = "isbn"/></isbn>  
      <unitPrice><xsl:value-of select = "price"/></unitPrice>  
    </item>  
  </xsl:template>  
</xsl:stylesheet>
```

**Listing 7.4 priceView.xsl--XSL file for generating the view for Book Catalog System**

**Step 5. Generate client stubs and server skeletons from CORBA IDL.** VisiBroker and Java are used to implement our prototype system. The interface specification we created in IDL is used by VisiBroker's idl2java compiler to generate Java classes for the client program, and skeleton code for the object implementation. The Java classes are used by the client program for all method invocations. The view.idl file can be compiled with the following command: idl2java view.idl.

**Step 6. Implement the view object on the server side. This object should inherit the generated skeleton classes and implement the function to generate XML views. The following java code is the important excerpts of the view server implementation.**

```
public class viewImpl extends viewPOA{
    ...
    public String generateView(String title) {
        try {
            File stylesheet = new File(title+".xsl");
            File datafile = new File("catalog.xml");
            BufferedInputStream bis = new BufferedInputStream(new
FileInputStream(datafile));
            InputSource input = new InputSource(bis);
            SAXParserFactory spf = SAXParserFactory.newInstance() ;
            SAXParser parser = spf.newSAXParser();
            XMLReader reader = parser.getXMLReader();
            SAXTransformerFactory stf =
(SAXTransformerFactory)TransformerFactory.newInstance() ;
            XMLFilter filter = stf.newXMLFilter(new StreamSource(stylesheet));
            filter.setParent(reader);
            StreamResult result = new StreamResult();
            Transformer transformer = stf.newTransformer();
            SAXSource transformSource = new SAXSource(filter, input);
            transformer.transform(transformSource, result) ;
            ...

        } catch (TransformerConfigurationException tce){
            ...
        }
    }
}
```

**Listing 7.5 viewImpl.java – Java implementation of the view generating class**

**Step 7. Implement the client applications. For our example, the priceViewClient class implements the client application, which obtains the XML view and processes the view document using the DOM. Listing 7.6 is composed of the excerpts of the client**

program. The program performs the following steps: initializes the ORB; binds to a View object; obtains the XML view by invoking generateView on the View object; query the returned view to get the price by title or by isbn.

```
public class priceViewClient{
    price.view.priceView _priceView;
    com.borland.cx.OrbConnect orbConnect1;
    String _name = "View";
    String view;
    . . .

    public static void main(String[] args) {
        if (!bInitialized) {
            try {
                org.omg.CORBA.ORB orb = null;
                if (orbConnect1 != null) {
                    orb = orbConnect1.initOrb();
                }
                if (orb == null) {
                    orb = org.omg.CORBA.ORB.init((String[])null,
System.getProperties());
                }
                _priceView = ViewHelper.bind(orb, "/" + _name + "_poa",
_name.getBytes());
                bInitialized = true;
            }
            catch (Exception ex) {
                ex.printStackTrace();
            }
        }
        view = _priceView.generateView("price");

        //further process the string view to query the price by the title or the
isbn provided by the client using the DOM
        . . .
    }
}
```

**Listing 7.6 priceViewClient.java – Java implementation of the client application**

**Step 8. Implement the CORBA server to initialize the ORB, create the POA, activate the server objects and wait for the client request. Listing 7.7 is composed of the excerpts of server program.**

```
public class viewServer{

    public static void main(String[] args) {
        try {
            String name;
            System.getProperties().put("vbroker.agent.port", "14000");
            System.getProperties().put("org.omg.CORBA.ORBClass",
"com.inprise.vbroker.orb.ORB");
            System.getProperties().put("org.omg.CORBA.ORBSingletonClass",
"com.inprise.vbroker.orb.ORB");

            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,
System.getProperties());
            POA poaRoot =
POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
            name = "View";
            org.omg.CORBA.Policy[]ViewPolicies = {
                poaRoot.create_lifespan_policy(LifespanPolicyValue.PERSISTENT)
            };
            POA poaView = poaRoot.create_POA(name + "_poa",
                                           poaRoot.the_POAManager(),
                                           ViewPolicies);
            poapriceView.activate_object_with_id(name.getBytes(), new
ViewImpl());

            poaRoot.the_POAManager().activate();
            orb.run();
        }
        catch(Exception ex) {
            System.err.println(ex);
        }
    }
}
```

**Listing 7.7 viewServer.java – Java implementation of the CORBA server**

## **7.2 Guidelines for XPath Mapping Approach**

**Step 1. Construct the XML document for storing the shared data. (Same as the XQuery/XSL approach)**

**Step 2. Define application interface using CORBA IDL. The interface includes the functions performing the client queries, each of which corresponds to an XPath query to the XML data. The IDL for the prototype system is defined in mappingView.idl:**

```
module view {  
  
    interface priceView {  
  
        string priceByTitle(in string title);  
  
        string priceByISBN(in string isbn);  
  
    };  
  
};
```

**Step 3. Construct the structure of the XML view for each application according to the CORBA IDL and the application logic. From the IDL defined in step2, we know that the application needs the title, isbn and price information of the books in the catalog. Then we can construct the structure of the XML view for this application. The tree representation of the view is the same as shown in Figure 7.2.**



Step 4. Define XML view as the mapping between the XPath expressions against the source data and those against the view. The mapping for the Book Catalog System can be defined as following:

View Path	Source Path
/catalog	/catalog
/catalog/item	/catalog/book
/catalog/item/title	/catalog/book/@title
/catalog/item/isbn	/catalog/book/isbn
/catalog/item/unitPrice	/catalog/book/price

This mapping can be stored as an XML file shown in Listing 7.8.

```
<?xml version="1.0" encoding="UTF-8"?>
<mapping>
  <mapPair>
    <view>/catalog</view>      <source>/catalog</source>
  </mapPair>
  <mapPair>
    <view>/catalog/item</view>  <source>/catalog/book</source>
  </mapPair>
  <mapPair>
    <view>/catalog/item/title</view>  <source>/catalog/book/@title</source>
  </mapPair>
  <mapPair>
    <view>/catalog/item/isbn</view>  <source>/catalog/book/isbn</source>
  </mapPair>
  <mapPair>
    <view>/catalog/item/unitPrice</view>  <source>/catalog/book/price</source>
  </mapPair>
</mapping>
```

**Listing 7.8 priceView.xml – XPath mapping stored as XML file**

**Step 5. Generate client stubs and server skeletons from CORBA IDL. (Same as the XQuery/XSL approach) The mappingView.idl file can be compiled with the following command: idl2java mappingView.idl.**

**Step 6. Implement the view object on the server side. This object should inherit the generated skeleton classes and implement the function to transfer XPath expressions according to the XPath mapping. The following java code is the priceView server implementation for Book Catalog System.**

```
package viewServer.view.server;

import java.sql.*;
import java.util.*;
import java.math.*;

import org.omg.PortableServer.*;

import java.io.*;
import java.util.*;

import org.w3c.dom.*;
import javax.xml.parsers.*;
import org.apache.crimson.tree.XmlDocument;

import org.xml.sax.*;
import org.xml.sax.helpers.DefaultHandler;

import com.fatdog.textEngine.XmlEngine;
import com.fatdog.textEngine.query.ResultListener;
import com.fatdog.textEngine.exceptions.*;
import com.borland.jbuilder.xml.database.template.*;
import com.borland.jbuilder.xml.database.xmldbms.*;

public class priceViewImpl extends price.view.priceViewPOA
    implements ResultListener{

    Document document;
```

```

Hashtable mapping;
final String viewName = "view\\priceView.xml";

String XML_FILE = "catalog.xml";
String query;
XmlEngine engine;
int maxHits;
String queryResult = "No result";

String _name = "priceView";

private void tableInit(){
    mapping = new Hashtable();

    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    factory.setValidating(true);
    factory.setIgnoringElementContentWhitespace(true);
    try {
        DocumentBuilder builder = factory.newDocumentBuilder();
        document = builder.parse(new File(viewName));

        } catch (SAXException se){
            se.printStackTrace();
        }catch (ParserConfigurationException pce){
            pce.printStackTrace();
        }catch (IOException ioe){
            ioe.printStackTrace();
        }
        Element root = document.getDocumentElement();
        NodeList mapPairs = root.getElementsByTagName("view");

        for (int i=0; i<mapPairs.getLength(); i++){
            String view = mapPairs.item(i).getFirstChild().getNodeValue();
            String source =
mapPairs.item(i).getNextSibling().getFirstChild().getNodeValue();
            mapping.put(view, source);
        }
    }

    private void engineInit() {
        engine = new XmlEngine();
        engine.setListenerType( XmlEngine.STANDARD_LISTENER );
        engine.setMaxHits( 200 );
        engine.addResultListener( this );
    }

```

```

engine.setDoFullText( true ); // the default
SAXParserFactory spf = SAXParserFactory.newInstance();
try
{
    // the SAX parser of choice (must be set)
    SAXParser parser = spf.newSAXParser();
    XMLReader reader = parser.getXMLReader();
    engine.setXMLReader( reader );
    // index the document(s) of choice
    engine.setDocument( XML_FILE );
} catch ( ParserConfigurationException pce ){
    pce.printStackTrace();
}
catch ( SAXParseException spe ){
    System.out.println( "\n** Parsing error"
        + ", line " + spe.getLineNumber()
        + ", uri " + spe.getSystemId());
    System.out.println( "  "+spe.getMessage() );
    Exception x = spe;
    if (spe.getException() != null)
        x = spe.getException();
    x.printStackTrace();
} catch ( SAXException sxe ) {
    Exception x = sxe;
    if (sxe.getException() != null)
        x = sxe.getException();
    x.printStackTrace();
}

catch( MissingOrInvalidSaxParserException e ) {
    System.out.println( "Missing or invalid SAX parser" ); return;
}
catch( FileNotFoundException e ) {
    System.out.println( "Couldn't find the XML file: " + e.getMessage() );
return;
}

catch( CantParseDocumentException e ) {
    System.out.println( "Couldn't parse the XML file: " + e.getMessage() );
return;
}

}

public void results( String results ){
    queryResult = results;
}

public priceViewImpl() {
    tableInit();
    engineInit();
}
}

```

```

        //Method queryToSource is the mapping function which translates queries to the
view
        //into queries to the source data
        private void queryToSource(){
            ...
        }

        public String priceByTitle(String title) {
            query = "/bib/book[title='" + title + "']/price";
            queryToSource();
            try {
                engine.setQuery( query );
            }
            catch( InvalidQueryException e )
            {
                queryResult = e.getMessage();
            }
            return queryResult;
        }

        public String priceByISBN(String last, String first) {
            ...
        }
    }

```

**Listing 7.9 priceViewImpl.java – Java implementation of the view supporting class**

**Step 7. Implement the client applications and the CORBA server. (Same as the XQuery/XSL approach)**

## **8 Guidelines for Systematically Modifying the System After Changes**

### **8.1 Guidelines for XQuery/XSL Approach**

In this approach, a view for the shared data is implemented as a single XQuery or an XSL file. When the structure of the shared data changes, we only need to modify the XQueries or the XSL files representing the views for the distributed applications. If the changes on the structure of the shared data do not affect the XPath expressions used in the XQuery or XSL file for generating a view, that specific view does not need to be modified at all. Therefore, to modify the system after changes, first, compared the new structure of the shared data with the original one, and identify the views that are affected by the changes on the structure of the shared data. Then modify the Xqueries or XSL files accordingly to generate the same views from the changed data as we have discussed in chapter 6.

### **8.2 Guidelines for XPath Mapping Approach**

In this approach, a view for the shared data is represented using a mapping between the XPath expressions against the view and those against the source data. Similar to the XQuery/XSL approach, first, we need to identify the views that are affected by

**the changes on the shared data. Then for the views that need to be modified, we can check the mapping tables representing the views and modify the XPath expressions against the source data that are affected by the changes on the shared data accordingly. We have given examples of the modification in chapter 6.**

## **9 Analysis of Work Done and Results**

### **9.1 Comparison of the Two Approaches Proposed**

One important advantage of the single XQuery/XSL approach is its simplicity. The idea is straightforward, so it is easy to understand and implement. The problem with this approach is that the view must be generated in a batch mode – the whole XML document must be processed to construct the view each time when a request comes. It is very inefficient, especially when the size of the shared data is huge. Moreover, the view document is physically generated and being sent to the client through the network, so the cost for file transfer would be very expensive if the size of the view document is not trivial and the data is accessed frequently.

For some applications (e.g. University Course Information System) that don't require high up-to-date concurrency of the shared data and the shared data is not updated very frequently, a simple improvement can be made to this approach for efficiency. A copy of the view document can be stored on the client machine. Instead of waiting for the server to generate the view and send it over the network each time a client requests the shared data, the client uses the local copy of the view document directly. Then the local copy of the view document is updated on a schedule (hourly, daily or weekly) depending on the requirements of the applications, or it can be updated each time the shared data is updated if the updated frequency of the shared data is low.



However, this approach is not suitable for other applications (e.g. transaction applications) that require high accuracy of the shared data and the shared data is updated frequently.

The XPath mapping approach is more efficient compared to the single XQuery/XSL approach because no batch transformation is involved. Only the data needed to answer the client query is retrieved and sent to the client instead of the whole view. This approach is more suitable for the applications that require high performance.

From the perspective of software engineering, the XPath mapping approach supports stronger data typing than the single XQuery/XSL approach. The CORBA IDL for the XQuery/XSL approach exposes very little type information about the data. A client only need to provide the name of the view to get the physical view generated, and the data is sent to the client in plain text. Further processing of the view data is performed on the client side, while in the XPath approach, strong typing is enforced by more specific methods defined in IDL. More data processing is done on the server side, and only data points that are of to the interest of the client are extracted and sent back to the client.

After changes to the DTD of the shared XML data, the XQueries or XSL files representing the affected views need to be identified and modified accordingly in the XQuery/XSL approach, while in the XPath approach, the affected XPath expressions against the original data need to be changed. Since these XPath expressions are all

gathered in the mapping files defining the views, it is quite easy to make the process of identifying the affected expressions and performing the modifications automatic. Therefore, making modification after changes is easier in the XPath approach than in the XQuery/XSL approach.

## **9.2 Analysis of Using XML and Views to Improve Data-independence**

XML provides a neutral syntax for describing graph-structured data as nested, tagged elements. Because developers can transform diverse data structures into such graphs, XML along with XPath, XSLT and XQuery provides a means to construct XML documents to store shared data for distributed applications and build XML views for different applications. With the views on top of the shared data, changes to the DTD of the shared data can be made without requiring changes to applications that access the data through the views. Hence, the modification the system after changes of the structure of the shared data is minimized and data-independence is improved.

One obvious advantage to XML is that it provides a way to represent structured data without any additional information. Because this structure is inherent in the XML document rather than needing to be driven by an additional document that describes how the structure appears as that in a database or a flat file, it becomes very easy to send structured information between systems. Since XML documents are simply text files, they may also be produced and consumed by legacy systems allowing these

systems to expose their legacy data in a way that can easily be accessed by different consumers.

Another advantage to the use of XML is the ability to leverage tools that use XML to drive more sophisticated behavior. A vibrant XML marketplace is providing inexpensive tools for preparing, validating and parsing XML data. XML's strong base of freeware and commercial tools affords flexibility at greatly reduced development costs.

These advantages provide generic operations on XML data, which avoids narrowly tailoring software for a given set of data. This in turn reduces the coupling between the applications' logic and the structure of the shared data and improves data-independence of the system.

However, relational databases will perform better than XML documents. This means that for many internal uses, if there are no network or usage barriers, relational databases will be a better storage for the data than XML.

### **9.3 Element Vs. Attribute in Data Modeling Using XML**

This is the issue that has caused the most heated debate in the XML area. We have discussed data modeling using XML for the shared data in Chapter 5. Both elements

and attributes can be used to represent data values when modeling the shared data. The following comparisons summarize our findings on this issue in the thesis work.

### **9.3.1 Compatibility with databases**

In relational database, content (data values), and structure are totally different. Structure is represented with tables and relationships, and data values are represented with columns. In XML, if we use text-only elements to represent data values, we lose this clear distinction. Sometimes elements represent structure, and other times they represent content. Any code that is parsing these structures must be aware which element represents data and which represents structure, and must handle it appropriately. However, if we use attributes for data values, structure and content are separate and distinct. Structure is represented by the elements and the way they are nested, while content is represented in attributes.

In addition, attributes are unordered. They are identical from the perspective of an XML parser, and the parser does not attach any particular importance to the order in which attributes are encountered in the original document. This is similar to the way a relational database works, where the meaning of a data value is simply indicated by its name, and not a combination of its name and location. While element order has meaning for documents, it loses importance when representing data. Thus, the ordering of elements just adds unnecessary complexity.

### **9.3.2 Data typing**

When using DTD to define the content of XML structures, there is little scope for strong data typing. The one exception would be the ability to constrain attributes to a particular list of allowable values. However, if the data value is expressed as an element, there is no similar way to limit these allowable values.

### **9.3.3 Document size**

When an element is used to describe a data value, three things appear in the serialized XML document: the start tag, the data value, and the end tag. For example:

```
<title>Java</title>
```

However, when an attribute is used to describe a data value, the attribute name, the equals sign, the quotes, and the attribute value are required:

```
title="Java"
```

It is obvious that repeating the name of the data value in the end element tag increases the size of the document, compared with the size of the document when an attribute is used to represent the data value. As a result, more disk space will be consumed by the documents and more network bandwidth will be consumed when transmitting files using elements than when using attributes.

Therefore, attributes are better suited to the representation of data values than text-only elements are. Attributes are best suited when only one value is to be expected

whereas elements are necessary when multiple values are needed. An example is that an <authors> element can contain multiple <author> elements.

## **9.4 Integration of CORBA and XML**

In this thesis work, we proposed approaches to build complex distributed software system based on the integration of CORBA and XML. CORBA provides a scalable infrastructure to distributed systems, allowing systems on heterogeneous platforms to communicate through commonly defined services. CORBA makes the components in the systems portable. XML and its related technologies provide a neutral syntax for storing the shared data and standard operations for accessing the data. XML supports portable data for distributed systems. CORBA and XML are complimentary standards; the integration of them is powerful for building flexible, scalable and low maintenance distributed systems.

## **10 Conclusions**

### **10.1 The Defense of the Thesis**

**The thesis statement of this work is:**

**“XML views to support data-independence of distributed computing applications can be generated systematically from application logic, CORBA IDL and the XML DTD.”**

**We have proved the thesis statement by providing a framework for building distributed computing applications that access shared data, presenting a methodology for modeling the shared data using XML, and developing systematic ways for generating XML views to support data-independence of the system and modifying the system after changes to the structure of the shared data according to the application logic, CORBA IDL and the XML DTD. A prototype system was implemented to demonstrate the effectiveness of our approaches.**

**The framework and approaches proposed in this thesis work provide distributed applications with transparent access to shared data from heterogeneous sources, and make it easy to design, implement and maintain the system.**

## **10.2 Problems and Future work**

XML is a rather new standard, and some of the technologies related to it are not stable yet. As we have mentioned in chapter 3, XPath is intended primarily as a component that can be used by other specifications, so there is no independent product for processing XPath expressions. XQuery is now the working draft of W3C, and it is still changing. A mature XQuery engine product has not come out yet. Therefore, our prototype system only implemented some very basic functions using very simple XPath expressions. However, the importance of the independent existence of XPath has been realized; thus, XPath processors are expected to emerge very soon.

The approaches presented in this thesis only deal with queries to the shared data because XPath, XSLT and XQuery are all query languages for XML documents, and none of them handle updates to the data. Therefore, the research on the updates of the shared data could be the future direction of our work.



## REFERENCES

- [ABIT97] S. Abiteboul, R. Goldman, J. McHugh, V. Vassalos, and Y. Zhuge. Views for Semistructured Data. *Proceedings of the Workshop on Management of Semistructured Data*, Tucson, Arizona, May 1997.
- [ABIT98] S. Abiteboul, J. McHugh, M. Rys, V. Vassalos, and J. Wiener. Incremental Maintenance for Materialized Views over Semistructured Data. *Proceedings of the Twenty-Fourth International Conference on Very Large Databases*, New York, August 1998.
- [ADLE01] S. Adler and A. Berglund. Extensible Stylesheet Language (XSL) 1.0, *W3C Recommendation*. <http://www.w3.org/TR/xsl/>
- [ANDE99] R. Anderson, M. Birbeck, M. Key and S. Livingstone. Professional XML. *Wrox Press Ltd.*, 1999.
- [BARA97] A. Baratloo, M. Karaul, H. Karl, and Z. Kedem. KnittingFactory: An Infrastructure for Distributed Web Applications. *Technical Report TR 1997-748*, Department of Computer Science, New York University, New York, NY, November 1997.
- [BARU99] C. Baru et al. XML-based Information Mediation with MIX. In *Demonstrations Program of ACM SIGMOD Conf.*, 1999.

[BERG98] K. Bergner, A. Rausch, M. Sihling, and A. Vilbig. An integrated view on componentware - concepts, description techniques, and development process. *In Roger Lee, editor, Software Engineering : Proceedings of the IASTED Conference '98*. ACTA Press, Anaheim, 1998.

[CHAM99] D. Chamberlin, J. Robie, and D. Florescu. Quilt: an XML Query Language for Heterogeneous Data Sources. See  
“[http://www.almaden.ibm.com/cs/people/chamberlin/quilt\\_lncs.pdf](http://www.almaden.ibm.com/cs/people/chamberlin/quilt_lncs.pdf)”.

[CHAW98] S. Chawathe, S. Abiteboul, and J. Widom. Representing and Querying Changes in Semistructured Data. *Proceedings of the Fourteenth International Conference on Data Engineering*, Orlando, Florida, February 1998.

[CHUN97] E. Chung, Y. Huang, S. Yajnik, D. Liang, J. C. Shih, C. Wang and Y. Wang. DCOM and CORBA Side by Side, Step by Step and Layer by Layer. November 1997, <http://www.bell-labs.com/~emerald/dcom-corba/Paper.html>.

[CLUE99] S. Cluet, S. Jacqmin, and J. Simeon. The New YATL: Design and Specifications. *Technical Report*, INRIA, 1999.

- [DEUT98] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A Query Language for XML. See <http://www.research.att.com/~mff/files/final.html>
- [EWAL96] Alan Ewald and Mark Roy. Choosing Between CORBA and DCOM. *Object Magazine* 6, 8 (October 1996): 24-30.
- [GAMM94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Design*. Addison-Wesley, 1994.
- [GOLD00] R. Goldman, J. McHugh, and J. Widom. From Semistructured Data to XML: Migrating the Lore Data Model and Query Language. *Markup Languages: Theory & Practice*, 2(2), 2000.
- [KRIE98] David Krieger and Richard M. Adler. The Emergence of Distributed Component Platforms. *Computer*, Vol. 31, No. 3, March 1998, pp. 43-53 .
- [LAHI99] T. Lahiri, S. Abiteboul, and J. Widom. Ozone: Integrating Structured and Semistructured Data. *Proceedings of the Seventh International Conference on Database Programming Languages*, Kinloch Rannoch, Scotland, September 1999.
- [MATJ99] Matjaz B. Juric, Ivan Rozman and Marjan Hericko. A Method for Integrating Legacy Systems within Distributed Object Architecture. *Proceedings of the*

*international conference on enterprise information systems*, Setúbal, Portugal, March 27-30, 1999

[MCHU99] J. McHugh and J. Widom. Query Optimization for XML. *Proceedings of the Twenty-Fifth International Conference on Very Large Data Bases*, pages 315-326, Edinburgh, Scotland, September 1999.

[MERL96] P. Merle, C. Gransart, and J. Geib. CorbaWeb: A WWW and Corba Worlds Integration. *In The 2th COOTS, Workshop on Distributed Object Computing on the Internet*, Toronto, Canada, June 1996.

[ORFA96] R. Orfali, D. Harkey, and J. Edwards, *The Essential Client/Server Survival Guide*. Wiley and Sons, 1996.

[POUR98] G. Pour. Developing Web-Based Enterprise Applications with Java, JavaBeans, and CORBA. *Proceedings of the 3' World Conference of the Www, Internet, and Intranet (WebNet)*, 1998.

[ROBI98] J. Robie, J. Lapp, D. Schach. XML Query Language (XQL). See <http://www.w3.org/TandS/QL/QL98/pp/xql.html>.

[SERG97] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The Lorel Query Language for Semistructured Data. *International Journal*

on *Digital Libraries*, 1(1):68-88, April 1997. See <http://www-db.stanford.edu/~widom/pubs.html>

[SUZU99] J. Suzuki and Y. Yamamoto. Toward the Interoperable Software Design Models: Quartet of UML, XML, DOM and CORBA. *Proc. fourth IEEE International Software Engineering Standards Symposium*, IEEE Press, to appear, 1999.

[STEV97] Steve Vinoski, CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments, *IEEE Communications Magazine*, vol. 14, no. 2, February 1997.

[XML00] World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Second Edition). *W3C Recommendation*, 6 October 2000. See <http://www.w3.org/TR/2000/WD-xml-2e-20000814>.

[XPath99] World Wide Web Consortium. XML Path Language (XPath) Version 1.0. *W3C Recommendation*, Nov. 16, 1999. See <http://www.w3.org/TR/xpath.html>

[XSLT99] World Wide Web Consortium. *XSL Transformations (XSLT)*. W3C Recommendation. Available at <http://www.w3.org/TR/xslt>.

[YANG96] Z. Yang and K. Duddy. CORBA: A Platform for Distributed Object Computing. *Operating System Review*, ACM, 30(2):pages 4--31, April 1996.

## APPENDIX: Program List

```
/**
 * Copyright (c) 2001
 * IDL Source File   view.idl
 * Abstract:  CORBA server application.
 * @version 1.0
 */

package viewServer;

import viewServer.view.server.*;
import javax.swing.UIManager;
import java.awt.*;
import org.omg.PortableServer.*;

public class viewServerApp {

    boolean packFrame = false;

    public viewServerApp() {
        ServerFrame frame = new ServerFrame();

        if (packFrame)
            frame.pack();
        else
            frame.validate();

        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height)
            frameSize.height = screenSize.height;
        if (frameSize.width > screenSize.width)
            frameSize.width = screenSize.width;
        frame.setLocation((screenSize.width - frameSize.width) / 2,
            (screenSize.height - frameSize.height) / 2);
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        try {

            UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");

            //UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

            //UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

            //UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif.MotifLookAndFeel");

            //UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
        }
    }
}
```

```

        catch (Exception ex) {
        }
        new viewServerApp();

        try {
            java.util.ResourceBundle res =
java.util.ResourceBundle.getBundle("viewServer.view.server.ServerResour
ces");
            String name;

            //(debug
support)System.getProperties().put("vbroker.agent.debug", "true");
            //(debug support)System.getProperties().put("vbroker.orb.warn",
"2");
            if (System.getProperties().get("vbroker.agent.port") == null) {
                System.getProperties().put("vbroker.agent.port", "14000");
            }
            if (System.getProperties().get("org.omg.CORBA.ORBClass") == null)
{
                System.getProperties().put("org.omg.CORBA.ORBClass",
"com.inprise.vbroker.orb.ORB");
            }
            if (System.getProperties().get("org.omg.CORBA.ORBSingletonClass")
== null) {
                System.getProperties().put("org.omg.CORBA.ORBSingletonClass",
"com.inprise.vbroker.orb.ORB");
            }

            org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,
System.getProperties());

            POA poaRoot =
POAHelper.narrow(orb.resolve_initial_references("RootPOA"));

            name = "priceView";
            org.omg.CORBA.Policy[] priceViewPolicies = {
                poaRoot.create_lifespan_policy(LifespanPolicyValue.PERSISTENT)
            };
            POA poapriceView = poaRoot.create_POA(name + "_poa",
                poaRoot.the_POAManager(),
                priceViewPolicies);
            poapriceView.activate_object_with_id(name.getBytes(), new
priceViewImpl());

            ServerMonitor.log(ServerResources.format(res.getString("created"),
"viewServerApp.java priceView"));

            poaRoot.the_POAManager().activate();

            ServerMonitor.log(ServerResources.format(res.getString("isReady"),
"viewServerApp.java view"));
            orb.run();
        }
        catch (Exception ex) {
            System.err.println(ex);
        }
    }
}

```

```

}

/**
 * Abstract: Server application frame which is the container for the
 * Server Monitor.
 */

package viewServer.view.server;

import java.awt.*;
import java.awt.event.*;

public class ServerFrame extends javax.swing.JFrame {

    BorderLayout borderLayout1 = new BorderLayout();
    ServerMonitor serverMonitor = new ServerMonitor();

    public ServerFrame() {
        enableEvents(AWTEvent.WINDOW_EVENT_MASK);
        try {
            jbInit();
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void jbInit() throws Exception {
        this.getContentPane().setLayout(borderLayout1);
        this.setSize(new Dimension(600, 300));
        this.setTitle("view server");
        this.getContentPane().add(serverMonitor);
    }

    protected void processWindowEvent(WindowEvent e) {
        super.processWindowEvent(e);
        if (e.getID() == WindowEvent.WINDOW_CLOSING) {
            System.exit(0);
        }
    }
}

/**
 * Abstract: Maintains the server log and is the container for all the
 * Server Monitor pages.
 */

package viewServer.view.server;

import java.awt.*;
import java.text.*;

public class ServerMonitor extends javax.swing.JPanel {
    static com.borland.dbswing.JdbTextArea myLog = null;
    static ServerMonitor monitor;

```



```

    java.util.ResourceBundle res =
java.util.ResourceBundle.getBundle("viewServer.view.server.ServerResour
ces");
    BorderLayout borderLayout1 = new BorderLayout();
    BorderLayout borderLayout2 = new BorderLayout();
    GridBagLayout gridBagLayout1 = new GridBagLayout();
    javax.swing.JTabbedPane tabPanell = new javax.swing.JTabbedPane();
    javax.swing.JPanel panell = new javax.swing.JPanel();

    BorderLayout moduleBorderLayoutView = new BorderLayout();
    javax.swing.JPanel panelView = new javax.swing.JPanel();
    com.borland.dbswing.JdbTextArea textView = new
com.borland.dbswing.JdbTextArea();
    javax.swing.JScrollPane scrollView = new javax.swing.JScrollPane();
    java.util.Vector pagesToRefresh = new java.util.Vector();

    public ServerMonitor() {
        monitor = this;
        try {
            jbInit();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    private void jbInit() throws Exception{
        this.setLayout(borderLayout1);
        this.add(panell, BorderLayout.CENTER);
        panell.setLayout(borderLayout2);
        panell.add(tabPanell, BorderLayout.CENTER );

        textView.setEnabled(true);
        panelView.setLayout(moduleBorderLayoutView);
        scrollView.getViewPort().add(textView);
        panelView.add(scrollView, BorderLayout.CENTER);
        myLog = textView;
        tabPanell.addTab(ServerResources.format(res.getString("logTitle"),
"view"), panelView);
        tabPanell.setSelectedComponent(panelView);
    }

    private void addPage(ServerMonitorPage page, String name) {
        tabPanell.addTab(name, page);
        pagesToRefresh.addElement(page);
        //panell.updateUI();
        //tabPanell.setEnabledAt(0, true);
    }

    private static ServerFrame createFrame() {
        ServerFrame frame = new ServerFrame();
        frame.pack();
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height)
            frameSize.height = screenSize.height;
        if (frameSize.width > screenSize.width)

```

```

        frameSize.width = screenSize.width;
        frame.setLocation((screenSize.width - frameSize.width) / 2,
(screenSize.height - frameSize.height) / 2);
        frame.setVisible(true);

        return(frame);
    }

    protected static void ensureFrame() {
        if (ServerMonitor.monitor == null) {
            ServerMonitor.monitor = new ServerMonitor();
            ServerFrame frame = createFrame();
            frame.getContentPane().add(ServerMonitor.monitor);
        }
    }

    public static synchronized ServerMonitorPage addPage(Object obj,
String name) {
        ensureFrame();
        ServerMonitorPage page = new ServerMonitorPage(obj);
        ServerMonitor.monitor.addPage(page, name);
        return page;
    }

    public static synchronized void log(String str) {
        if (myLog != null) {
            DateFormat df = DateFormat.getDateInstance(DateFormat.SHORT,
DateFormat.LONG);
            myLog.append(df.format(new java.util.Date()) + " " + str +
System.getProperty("line.separator"));
        }
    }
}

/**
 * Abstract: Implements a Server Monitor page to display interface
counters.
 */

package viewServer.view.server;

import java.awt.*;

public class ServerMonitorPage extends javax.swing.JPanel {
    java.util.ResourceBundle res =
java.util.ResourceBundle.getBundle("viewServer.view.server.ServerResour
ces");
    GridBagLayout gridBagLayout1 = new GridBagLayout();
    GridLayout gridLayout1 = new GridLayout(1, 2, 3, 0);
    javax.swing.JPanel panelOuter1 = new javax.swing.JPanel();

    javax.swing.JPanel panelObjects1 = new javax.swing.JPanel();
    com.borland.dbswing.JdbLabel labelObjects1 = new
com.borland.dbswing.JdbLabel();
    com.borland.dbswing.JdbTextField textObjects1 = new
com.borland.dbswing.JdbTextField();

```

```

int objectsCounter = 0;

Object monitoredObject;

public ServerMonitorPage(Object obj) {
    monitoredObject = obj;
    try {
        jbInit();
    }
    catch (Exception ex) {
        ex.printStackTrace();
    }
}

private void jbInit() throws Exception{
    panelOuter1.setLayout(gridBagLayout1);
    panelObjects1.setLayout(gridLayout1);
    textObjects1.setEnabled(false);
    labelObjects1.setText(res.getString("numberObjects"));

    labelObjects1.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);
    panelObjects1.setVisible(false);
    panelObjects1.add(labelObjects1);
    panelObjects1.add(textObjects1);
    panelOuter1.add(panelObjects1,
        new java.awt.GridBagConstraints(1, 2, 2, 1, 1.0, 1.0,
            java.awt.GridBagConstraints.NORTH,
            java.awt.GridBagConstraints.HORIZONTAL,
            new Insets(3, 0, 3, 3), 0, 0));

    add(panelOuter1);
}

public void showObjectCounter(boolean bVisible) {
    refresh();
    panelObjects1.setVisible(bVisible);
}

public synchronized void updateObjectCounter(int n) {
    objectsCounter += n;
    textObjects1.setText(String.valueOf(objectsCounter));
}

public void refresh() {
    textObjects1.setText(String.valueOf(objectsCounter));
}
}

package price;

import javax.swing.UIManager;
import java.awt.*;

public class checkPrice {

```

```

boolean packFrame = false;

/**Construct the application*/
public checkPrice() {
    Frame2 frame = new Frame2();
    frame.setTitle("Price Client");
    //Validate frames that have preset sizes
    //Pack frames that have useful preferred size info, e.g. from their
    layout
    if (packFrame) {
        frame.pack();
    }
    else {
        frame.validate();
    }
    //Center the window
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height) {
        frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
        frameSize.width = screenSize.width;
    }
    frame.setLocation((screenSize.width - frameSize.width) / 2,
(screenSize.height - frameSize.height) / 2);
    frame.setVisible(true);
}
/**Main method*/
public static void main(String[] args) {
    try {

        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
    catch(Exception e) {
        e.printStackTrace();
    }
    new checkPrice();
}
}

```

```

package price;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.borland.jbcl.layout.*;

import java.io.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import javax.xml.parsers.*;
import org.apache.crimson.tree.XmlDocument;

```

```

public class Frame2 extends JFrame {

    JPanel contentPane;
    BorderLayout borderLayout1 = new BorderLayout();
    JTabbedPane jTabbedPane1 = new JTabbedPane();
    JPanel byTitle = new JPanel();
    JPanel byAuthor = new JPanel();
    JLabel title = new JLabel();
    XYLayout xYLayout1 = new XYLayout();
    JTextField jTextTitle = new JTextField();
    JButton checkPriceByTitle = new JButton();
    JScrollPane jScrollPane1 = new JScrollPane();
    JLabel result = new JLabel();
    JTextArea jTextArea1 = new JTextArea();
    JLabel first = new JLabel();
    XYLayout xYLayout2 = new XYLayout();
    JTextField jTextFirst = new JTextField();
    JLabel last = new JLabel();
    JTextField jTextLast = new JTextField();
    JButton checkPriceByAuthor = new JButton();
    JScrollPane jScrollPane2 = new JScrollPane();
    JLabel result1 = new JLabel();
    JTextArea jTextArea2 = new JTextArea();

    priceViewClientImpl1 pvc;

    /**Construct the frame*/
    public Frame2() {
        enableEvents(AWTEvent.WINDOW_EVENT_MASK);
        try {
            jbInit();
            pvc = new priceViewClientImpl1();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
    /**Component initialization*/
    private void jbInit() throws Exception {

        //setIconImage(Toolkit.getDefaultToolkit().createImage(Frame2.class.getResource("[Your Icon]")));
        contentPane = (JPanel) this.getContentPane();
        contentPane.setLayout(borderLayout1);
        this.setSize(new Dimension(400, 300));
        this.setTitle("Frame Title");
        title.setText("Title");
        byTitle.setLayout(xYLayout1);
        checkPriceByTitle.setText("Check Price");
        checkPriceByTitle.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(ActionEvent e) {
                checkPriceByTitle_actionPerformed(e);
            }
        });
        result.setText("Result");
        first.setText("First Name");
    }
}

```

```

        byAuthor.setLayout(xYLayout2);
        last.setText("Last Name");
        checkPriceByAuthor.setText("Check Price");
        checkPriceByAuthor.addActionListener(new
java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        checkPriceByAuthor_actionPerformed(e);
    }
});
result1.setText("Result");
jTabbedPane1.add(byTitle, "By Title");
byTitle.add(title, new XYConstraints(33, 35, 43, -1));
byTitle.add(jTextTitle, new XYConstraints(92, 33, 286, 22));
byTitle.add(checkPriceByTitle, new XYConstraints(132, 74, 132,
25));
byTitle.add(jScrollPane1, new XYConstraints(38, 131, 331, 115));
byTitle.add(result, new XYConstraints(30, 98, 56, 28));
jTabbedPane1.add(byAuthor, "By Author");
jScrollPane1.getViewport().add(jTextArea1, null);
byAuthor.add(first, new XYConstraints(19, 63, -1, -1));
byAuthor.add(jTextFirst, new XYConstraints(107, 57, 218, 26));
byAuthor.add(last, new XYConstraints(15, 26, 62, 18));
byAuthor.add(jTextLast, new XYConstraints(107, 16, 217, 30));
byAuthor.add(checkPriceByAuthor, new XYConstraints(140, 96, 132,
25));
byAuthor.add(jScrollPane2, new XYConstraints(36, 142, 345, 115));
byAuthor.add(result1, new XYConstraints(14, 110, 56, 28));
jScrollPane2.getViewport().add(jTextArea2, null);
contentPane.add(jTabbedPane1, BorderLayout.CENTER);
}

/**Overridden so we can exit when window is closed*/
protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
        System.exit(0);
    }
}

void checkPriceByTitle_actionPerformed(ActionEvent e) {

    StringReader sr = new
StringReader(pvc.priceByTitle(jTextTitle.getText()));
    InputSource is = new InputSource(sr);

    DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
    try {
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document document = builder.parse(is);
        String price =
document.getElementsByTagName("price").item(0).getFirstChild().getNodeVa
alue();
        jTextArea1.append(price + "\n");

```

```

        }catch (SAXException se){
            se.printStackTrace();
        }catch (ParserConfigurationException pce){
            pce.printStackTrace();
        }catch (IOException ioe){
            ioe.printStackTrace();
        }
    }

    void checkPriceByAuthor_actionPerformed(ActionEvent e) {
        jTextArea2.append(pvc.priceByAuthor(jTextLast.getText(),
jTextFirst.getText()) + "\n");
    }
}

```

```

package price;

import java.awt.*;
import org.omg.CORBA.*;

public class priceViewClientImpl1 {
    boolean bInitialized = false;
    price.view.priceView _priceView;
    com.borland.cx.OrbConnect orbConnect1;
    String _name = "priceView";

    public priceViewClientImpl1() {
        try {
            jbInit();
        }
        catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    private void jbInit() throws Exception {
    }

    public boolean init() {
        if (!bInitialized) {
            try {
                org.omg.CORBA.ORB orb = null;
                if (orbConnect1 != null) {
                    orb = orbConnect1.initOrb();
                }
                if (orb == null) {
                    orb = org.omg.CORBA.ORB.init((String[])null,
System.getProperties());
                }
                _priceView = price.view.priceViewHelper.bind(orb, "/" + _name +
*_poa", _name.getBytes());
                bInitialized = true;
            }
        }
    }
}

```

```

        catch (Exception ex) {
            ex.printStackTrace();
        }
    }
    return bInitialized;
}

public price.view.priceView getCorbaInterface() {
    return _priceView;
}

public void setCorbaInterface(price.view.priceView intf) {
    _priceView = intf;
}

public com.borland.cx.OrbConnect getORBConnect() {
    return orbConnect1;
}

public void setORBConnect(com.borland.cx.OrbConnect orbConnect) {
    this.orbConnect1 = orbConnect;
}

public String priceByTitle(String title) {
    init();
    return _priceView.priceByTitle(title);
}

public String priceByAuthor(String last, String first) {
    init();
    return _priceView.priceByAuthor(last, first);
}
}

```

```

package configbib;

import javax.swing.UIManager;
import java.awt.*;

public class config {
    boolean packFrame = false;

    /**Construct the application*/
    public config() {
        Frame1 frame = new Frame1();
        //Validate frames that have preset sizes
        //Pack frames that have useful preferred size info, e.g. from their
        layout
        if (packFrame) {
            frame.pack();
        }
        else {
            frame.validate();
        }
    }
}

```



```

    }
    //Center the window
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height) {
        frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
        frameSize.width = screenSize.width;
    }
    frame.setLocation((screenSize.width - frameSize.width) / 2,
(screenSize.height - frameSize.height) / 2);
    frame.setVisible(true);
}
/**Main method*/
public static void main(String[] args) {
    try {

        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
    catch(Exception e) {
        e.printStackTrace();
    }
    new config();
}
}

```

```

package configbib;

```

```

import java.io.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import javax.xml.parsers.*;
import org.apache.crimson.tree.XmlDocument;

```

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import com.borland.jbcl.layout.*;

```

```

public class Frame1 extends JFrame {
    JPanel contentPane;
    BorderLayout borderLayout1 = new BorderLayout();
    JTabbedPane main = new JTabbedPane();
    JPanel mapping = new JPanel();
    JLabel viewPath = new JLabel();
    JTextField jTextVPath = new JTextField();
    XYLayout xYLayout1 = new XYLayout();
    JLabel sourcePath = new JLabel();
    JTextField jTextSPath = new JTextField();
    JButton AddMapping = new JButton();

    Document document;
    String viewName;

```

```

JButton ModifyMapping = new JButton();
JLabel result = new JLabel();
JButton deleteMapping = new JButton();

/**Construct the frame*/
public Frame1() {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);

    XMLInit();

    try {
        jbInit();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}

private void XMLInit(){
    viewName = JOptionPane.showInputDialog( " Please the name of the
view you want to edit:");
    viewName = "view\\" + viewName + ".xml";
    DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
    factory.setValidating(true);
    factory.setIgnoringElementContentWhitespace(true);
    try {
        DocumentBuilder builder = factory.newDocumentBuilder();

        File f = new File(viewName);
        if (!f.exists())

        {

            String init = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n" +
                "<!DOCTYPE mapping SYSTEM 'config.dtd'\n" +
                "<mapping/>\n";

            PrintWriter output = new PrintWriter(new
FileOutputStream(viewName));
            output.println(init);
            output.flush();

        }

        document = builder.parse(f);

    } catch (SAXException se){
        se.printStackTrace();
    }catch (ParserConfigurationException pce){
        pce.printStackTrace();
    }catch (IOException ioe){
        ioe.printStackTrace();
    }
}

```

```

        document.normalize();
    }

    /**Component initialization*/
    private void jbInit() throws Exception {

//setIconImage(Toolkit.getDefaultToolkit().createImage(Frame1.class.get
Resource("[Your Icon]")));
        contentPane = (JPanel) this.getContentPane();
        contentPane.setLayout(borderLayout1);
        this.setSize(new Dimension(400, 300));
        this.setTitle("Frame Title");
        viewPath.setText("View Path :");
        mapping.setLayout(xYLayout1);
        sourcePath.setText("Source Path :");
        AddMapping.setText("Add Mapping");
        AddMapping.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(ActionEvent e) {
                AddMapping_actionPerformed(e);
            }
        });
        ModifyMapping.setActionCommand("Modify Mapping");
        ModifyMapping.setText("Modify Mapping");
        ModifyMapping.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(ActionEvent e) {
                ModifyMapping_actionPerformed(e);
            }
        });
        deleteMapping.setActionCommand("deleteMapping");
        deleteMapping.setText("Delete Mapping");
        deleteMapping.addActionListener(new java.awt.event.ActionListener()
        {
            public void actionPerformed(ActionEvent e) {
                deleteMapping_actionPerformed(e);
            }
        });
        main.add(mapping, "mapping");
        contentPane.add(main, BorderLayout.CENTER);
        mapping.add(jTextVPath, new XYConstraints(95, 34, 274, 28));
        mapping.add(viewPath, new XYConstraints(12, 36, 73, 22));
        mapping.add(sourcePath, new XYConstraints(9, 92, 73, 22));
        mapping.add(jTextSPath, new XYConstraints(94, 92, 274, 28));
        mapping.add(result, new XYConstraints(18, 198, 355, 21));
        mapping.add(AddMapping, new XYConstraints(11, 150, -1, -1));
        mapping.add(ModifyMapping, new XYConstraints(128, 150, 118, -1));
        mapping.add(deleteMapping, new XYConstraints(260, 149, 126, -1));
    }

    /**Overridden so we can exit when window is closed*/
    protected void processWindowEvent(WindowEvent e) {
        super.processWindowEvent(e);
        if (e.getID() == WindowEvent.WINDOW_CLOSING) {
            try{
                ((XmlDocument)document).write(new FileOutputStream(viewName));
            }catch (IOException ioe){
                ioe.printStackTrace();
            }
        }
    }

```

```

        }
        System.exit(0);
    }
}

void AddMapping_actionPerformed(ActionEvent e) {
    Element mapPair = document.createElement("mapPair");
    Element vPath = document.createElement("view");
    Element sPath = document.createElement("source");

    vPath.appendChild(document.createTextNode(jTextVPath.getText()));
    sPath.appendChild(document.createTextNode(jTextSPath.getText()));

    mapPair.appendChild(vPath);
    mapPair.appendChild(sPath);

    Node mapping = document.getElementsByTagName("mapping").item(0);

    mapping.appendChild(mapPair);

    result.setText("New mapping has been added!");
}

void ModifyMapping_actionPerformed(ActionEvent e) {
    Element config = document.getDocumentElement();
    NodeList views = config.getElementsByTagName("view");
    // System.out.println (views.getLength());

    for (int i=0; i<views.getLength(); i++){
        Node v = views.item(i);
        if ( v.getFirstChild()!=null){
            String value = v.getFirstChild().getNodeValue();
            if (value.equals(jTextVPath.getText())){
                // System.out.println(v);
                // System.out.println(v.getNextSibling().getNextSibling());

                v.getNextSibling().getFirstChild().setNodeValue(jTextSPath.getText());
                result.setText("Mapping modified!");
                return;
            }
        }
    }
    result.setText("View path not found!");
}

void deleteMapping_actionPerformed(ActionEvent e) {
    Element config = document.getDocumentElement();
    NodeList views = config.getElementsByTagName("view");

    for (int i=0; i<views.getLength(); i++){
        Node v = views.item(i);
        if ( v.getFirstChild()!=null){
            String value = v.getFirstChild().getNodeValue();
            if (value.equals(jTextVPath.getText())){

```

```

        Node vp = v.getParentNode();
        vp.getParentNode().removeChild(vp);
        result.setText("Mapping deleted!");
        return;
    }
}
}
result.setText("View path not found!");
}
}

```

## **VITA AUCTORIS**

**NAME:** Xun Luo

**PLACE OF BIRTH:** Hunan, China

**YEAR OF BIRTH:** 1974

**EDUCATION:** Guangdong University of Technology Guanzhou,  
China  
1990-1994 B. Sc. in Computer Science

University of Windsor, Windsor, Ontario  
1999-2002 M. Sc. in Computer Science