Northumbria Research Link

Citation: Elbekai, Ali Sayeh (2006) Generic model for application driven XML data processing. Doctoral thesis, Northumbria University.

This version was downloaded from Northumbria Research Link: http://nrl.northumbria.ac.uk/55/

Northumbria University has developed Northumbria Research Link (NRL) to enable users to access the University's research output. Copyright © and moral rights for items on NRL are retained by the individual author(s) and/or other copyright owners. Single copies of full items can be reproduced, displayed or performed, and given to third parties in any format or medium for personal research or study, educational, or not-for-profit purposes without prior permission or charge, provided the authors, title and full bibliographic details are given, as well as a hyperlink and/or URL to the original metadata page. The content must not be changed in any way. Full items must not be sold commercially in any format or medium without formal permission of the copyright holder. The full policy is available online: http://nrl.northumbria.ac.uk/policies.html

www.northumbria.ac.uk/nrl

Generic Model for Application driven XML data Processing

Ali Sayeh Elbekai

A thesis submitted for the degree of
Doctor of Philosophy in Computing Science

University of Northumbria - Newcastle

School of Computing, Engineering and Information Science

September 2005

TABLE OF CONTENTS

CONTENTS **PAGE**

Abstract .... **viii**

Declaration.. **ix**

Acknowledgments .... **x**

List of Tables.. **xi**

List of Figures.. **xii**

List of Publication and Presentations.. **xvi**

Chapter 1: Introduction.. **1**

1.1 Goals of the research.. 2

1.2 Terminology explanation.. 4

1.3 Aims and objectives of the research.. 6

1.4 The choice of methodology.. 8

1.5 Contributions.. 9

1.6 Outline of the thesis.. 10

Chapter 2: XML as a Universal Data specification Language and Technology for Processing data.. **11**

2.1 Introduction.. 11

2.1.1 Brief history.. 11

2.1.2 XML vs. XML processor.. 12

2.1.3 XML's design goals.. 13

2.1.4 XML's main usage.. 14

2.2 XML vs. DB.. 14

2.3 XML and JAXP Technologies.. 16

2.3.1 XML parsing techniques (SAX, DOM, and JDOM).. 16

2.3.1.1 SAX APIs.. 17

2.3.1.1.1 How SAX works.. 19

2.3.1.2 DOM APIs.. 22

2.3.1.3 JDOM APIs.. 23

2.3.2 Summary of comparison of different XML parsing Techniques (SAX, DOM and JDOM)......24
2.3.3 XML Transformations......24
2.3.3.1 Why (eXtensible) document transformation method is needed......25
2.3.3.2 eXtensible Stylesheet Language Transformation (XSLT)...26
2.3.3.3 The Transformation process......27
2.3.4 Review of approaches related to the transformation of XML documents......28
2.3.5 Characteristics of our approach......29
2.3.6 Document and XML document......30
2.3.6.1 Document Type Descriptors (DTDs)......30
2.3.6.2 XML schema......31
2.3.6.3 Comparison of DTD and XMLSchema......32
2.3.7 XQuery data model......33
2.3.7.1 XML XQuery language......33
2.3.7.2 Path expression......34
2.3.8 Storing XML document in Database......35
2.3.9 Transport......36
2.3.9.1 HTTP......36
2.3.9.2 SOAP (Simple Object Access Protocol)......37
2.3.9.2.1 Message structure: SOAP Envelope contained in a HTTP message......37
2.4 Conclusion......40

Chapter 3: Review of XML algebra......41
3.1 Definition of Algebra......41
3.1.1 Relational Algebras......41
3.1.2 Operators......42
3.1.3 Query Algebra......42
3.2 Semi-structured algebras......43
3.3 XML Algebra Models......45
3.3.1 IBM Algebra......45

3.3.2 Niagara Algebra..46
3.3.3 YATL Algebra..49
3.3.4 Lore Algebra..50
3.3.5 AT&T Algebra..52
3.3.6 Tree Algebra for XML..54
3.3.7 W3C XML Algebra 54
3.3.7.1 W3C Data model...55
3.4 Compression analysis of different algebraic models.........................55
3.5 Conclusion..58

Chapter 4: Our XML Algebraic and XQuery Framework...............59

4.1 Description of our XML algebra..59
4.2 Our XML data model...60
4.2.1 Concepts tree model and data model definitions..........................62
4.2.2 Definition of XML document as a tree structure..........................62
4.3 Test examples...63
4.4 Our algebraic operators and algebraic relational operators.................65
4.4.1 Algebra relational operators...65
4.4.1.1 Universal elements relation..66
4.4.1.2 Similarity relation...68
4.4.1.3 Equivalence relation...69
4.4.1.4 Subsumption relation...70
4.4.2 Algebraic operators...71
4.4.2.1 Join operator..76
4.4.2.2 Union operator...79
4.4.2.3 Complement operator..82
4.4.2.4 Projection operator..84
4.4.2.5 Selection operator...86
4.4.2.6 Expose operator..87
4.4.2.7 Vertex operator..89
4.4.3 A complete algebra..91
4.5 Our XML Query Language..91

4.5.1 Syntax..92
4.5.1.1 Simple Examples in Our XML Query Language.............94
4.5.1.2 Complex Examples in Our XML Query Language..........95
4.5.2 Semantics of Our XML Query Language 99
4.5.2.1 Concepts XQuery model...99
4.6 Conclusion..102

Chapter 5: XML Mapping and Component Generation..................103
5.1 Mapping..104
5.1.1 Mapping Tree to Tree..104
5.1.1.1 Mapping XML to XML..104
5.1.1.2 Mapping XML to XHTML......................................107
5.1.2 Mapping Tree to Relation..109
5.1.2.1 Mapping XML to Nested Relational Tables.................109
5.2 Developing a Technological Solution for the System.....................111
5.2.1 SQL Schema generations...111
5.2.1.1 How the algorithm works......................................113
5.2.1.2 Testing the algorithm...115
5.2.1.2.1 Output of the algorithm..................................119
5.2.2 XSL stylesheet generation...121
5.2.2.1 How the algorithm works.......................................123
5.2.2.2 Testing the algorithm..124
5.2.2.2.1 Output 1 of the algorithm...............................125
5.2.2.2.2 Output 2 of the algorithm (generation an XSL for transforming XML to HTML)..........126
5.2.2.3 Testing the generator (XSL)....................................128
5.2.2.3.1 Transforming an XML document to SQL statement and storing in the database...............128
5.2.2.3.2 Transforming XML to HTML document..............134
5.2.3 Test Plans..140
5.2.3.1 Test Reports...141
5.2.4 The XQuery Interpreter generation...................................143

5.2.4.1 How the algorithm works..145
5.2.4.2 Testing the algorithm..146
5.2.4.2.1 Output of the algorithm......................................146
5.2.4.2.2 Testing the generator (XSL stylesheet)
for transforming XQueries to SQL queries........148
5.2.4.2.3 Testing the generator and browsing
the result on server)...151
5.2.5 Test Reports..152
5.2.6 The contribution of XML algebra and the XQuery
in the implementation...153
5.3 Conclusion...154

Chapter 6: Prototype development of the system..............................155
6.1 Museum information System...155
6.1.1 Brief history...155
6.1.2 Object-Oriented Conceptual References Model (OOCRM).......157
6.1.3 Primary objectives of CRM 157
6.2 Object identification and class deriving..158
6.3 Use case diagram...161
6.4 Packages...163
6.5 Design stage..164
6.5.1 Design of database for the proposed system..............................164
6.5.2 Design of XML Schema specification..165
6.5.3 Design of the component diagram...166
6.6.4 Design of the deployment diagram..167
6.6 Prototype implementation of integrated system............................169
6.6.1 Client browser display institution information.........................169
6.6.2 Client search and display museum objects................................170
6.6.3 Client browser search and display object information................172
6.6.4 Client Search and display exhibition information......................173
6.6.5 Staff store objects information in database...............................175
6.7 Conclusion..176

Chapter 7: Conclusion and further research possibilities................177

7.1 Originality...179

7.2 Future research possibilities.... 183

7.3 The overall achievement of the thesis.................................184

REFERENCES...186

APPENDICES...204

Appendix A: XML Schema...204

Appendix B: Java classes, (implementation of Online
and Offline system)..219

Appendix C: Implementation of transforming XQuiers to SQL queries...233

Appendix D: (Analyses of use case diagram, Activity diagrams and
Sequences diagrams for all the use cases.....................................237

Abstract

XML technology has emerged during recent years as a popular choice for *representing* and *exchanging* semi-structured data on the Web. It integrates seamlessly with web-based applications. If data is stored and represented as XML documents, then it should be possible to query the contents of these documents in order to extract, synthesize and analyze their contents.

This thesis for experimental study of Web architecture for data processing is based on semantic mapping of XML Schema. The thesis involves complex methods and tools for specification, algorithmic transformation and online processing of semi-structured data over the Web in XML format with persistent storage into relational databases. The main focus of the research is preserving the structure of original data for data reconciliation during database updates and also to combine different technologies for XML data processing such as storing (SQL), transformation (XSL Processors), presenting (HTML), querying (XQUERY) and transporting (Web services) using a common framework, which is both theoretically and technologically well grounded. The experimental implementation of the discussed architecture requires a Web server (Apache), Java container (Tomcat) and object-relational DBMS (Oracle 9) equipped with Java engine and corresponding libraries for parsing and transformation of XML data (Xerces and Xalan).

Furthermore the central idea behind the research is to use a single theoretical model of the data to be processed by the system (XML algebra) controlled by one standard metalanguage specification (XML Schema) for solving a class of problems (generic architecture). The proposed work combines theoretical novelty and technological advancement in the field of Internet computing.

This thesis will introduce a generic approach since both our model (XML algebra) and our problem solver (the architecture of the integrated system) are XML Schema-driven. Starting with the XML Schema of the data, we first develop domain-specific XML algebra suitable for data processing of the specific data and then use it for implementing the main offline components of the system for data processing.

Declaration

I declare that this dissertation represents my own work except otherwise stated.

Ali Sayeh Elbekai

....

Acknowledgements

I would like to thank my principle supervisor Dr. Nick Rossiter for his advice and encouragement and also for giving me the opportunity to work on such an interesting and demanding study and his valuable support and guidance during the elaboration of this dissertation. I am also grateful to my second supervisor Dr. Vassil Vassilev for his help and support during the period of this dissertation.

My sincerest gratitude goes out to spirit of my father, my mother, my wife Awatef Raheel, my daughters and my brothers. Also, I would like to express my thanks to all the staff of the School of Computing, Engineering and Information Science and to all my friends for their support through my academic life.

March 2005

LIST OF TABLES

TABLE PAGE

Table 1: Main Event Handlers......18

Table 2: Events in parsing an XML document......18

Table 3: Comparison of different XML Parsing Techniques (SAX, DOM, and JDOM)......24

Table 4: Simplification of the main relation algebra......42

Table 5: Niagara Algebra Operators......47

Table 6: Comparison of Different algebraic models......57

Table 7: Evolution of the Algorithms......142

Table 8: Evolution of the Reports......143

Table 9: Transforming XQueries to SQL queries using XSL generated......150

Table 10: Evolution of the Reports......153

Table 11: Objects in the museum system......160

LIST OF FIGURES

FIGURE	PAGE
Figure 2.1: An Example of an XML document	13
Figure 2.2: XML Schema mapping	15
Figure 2.3: Event SAX handlers	18
Figure 2.4: Diagram to show Parsing of an XML Document	21
Figure 2.5: JAXP APIs in Action	22
Figure 2.6: A Fragment of XSLT	25
Figure 2.7: The XSLT Transformation Processor	27
Figure 2.8: An Example of DTD	31
Figure 2.9: A Fragment of XML Schema for the Proposed System	32
Figure 2.10: Basic SOAP Message Structure with an Example	38
Figure 3.1: An XML Document and its Associated DTD	43
Figure 3.2: An XML Data Model	44
Figure 3.3: Set of Vertices	47
Figure 3.4: Implementation of Niagara Operators	48
Figure 3.5: Query Evaluated by YTAL Algebra	50
Figure 3.6: Logical-Physical Query Plans	52
Figure 4.1a: An XML document	61
Figure 4.1b: Tree data model for Figure 4.1a	62
Figure 4.2: Similarity relational	68
Figure 4.3: Equivalence relational	70
Figure 4.4: Subsumption relational	71
Figure 4.5: An algorithm for joining any two XML tree or two XML Documents	77
Figure 4.6: An Example showing the Join Operator	78
Figure 4.7: An Algorithm for Union of any two XML trees or two XML Documents	79
Figure 4.8: An Example of a Union Operator	80
Figure 4.9: An Algorithm to complement any two XML trees or two XML documents	82
Figure 4.10: An Example of a Complement Operator	84
Figure 4.11: An Algorithm for projecting nodes of XML Tree	85

Figure 4.12: Projection Operator, an Example.....85
Figure 4.13: Selection Operator, an Example.....87
Figure 4.14: An Algorithm to expose specific node of tree or XML document.....88
Figure 4.15: Expose Operator, an Example.....89
Figure 4.16: Vertex Operator, an Example.....90
Figure 4.17: FLWR expression semantics.....99
Figure 5.1: An Example for mapping XML to XML document.....105
Figure 5.2: An Algorithm for mapping XML to XML document (Tree to Tree).....106
Figure 5.3: An Algorithm for generating XML data by depth-first traversing.....107
Figure 5.4: An Example for mapping XML to XHTML document.....108
Figure 5.5: An algorithm for mapping XML to XHTML (Tree to Tree).....109
Figure 5.6: An Example for mapping XML to Nested Relational Tables.....110
Figure 5.7: An Algorithm to generate SQL Schema from XMLSchema.....112
Figure 5.8: Diagram for Mapping XML Schema to SQL schema.....114
Figure 5.9: SQL Schema for initial database.....117
Figure 5.10: Generic XSL stylesheet(GenericSQL.xsl) to generate SQL Schema...117
Figure 5.11: A Fragment of our XMLSchema.....118
Figure 5.12: SQL Schema-generated after using XSL Stylesheet (generator).....119
Figure 5.13: Execution for Generation of SQL schema for Relational Schema with GenericSQL.xsl.....120
Figure 5.14: An Algorithm for generating XSL from XMLSchema.....122
Figure 5.15: Diagram for generating XSL Stylesheet from XML schema.....123
Figure 5.16: Generic XSL stylesheet generated using the algorithm in Figure 5.14..125
Figure 5.17: Execution for generation of XSL Stylesheet using XMLSchema GenXSL.xsl.....126
Figure 5.18: Generic XSL Stylesheet for transforming XML to HTML Document..127
Figure 5.19: Execution for the Generation of an XSL Stylesheet of XMLSchema for transforming XML to HTML.....128
Figure 5.20: Example 1 for transforming an XML document to SQL Statement.....130
Figure 5.21: The Result of execution the Java Servlet Class (online) for transforming XML to SQL by using GenXSL.xsl.....131

Figure 5.22: Example 2 for transforming an XML document to SQL Statement....132
Figure 5.22a: The Result of executing the Java Servlet Class (online) for transforming XML to SQL by using GenXSL.xsl....133
Figure 5.23: Example 3 for transforming XML document to SQL (offline)....134
Figure 5.24: Transforming XML to HTML by using the generated XSL stylesheet....136
Figure 5.24a: The Result of executing the Java Servlet Class for transforming XML to HTML by using GenXSLHTML.xsl....137
Figure 5.25: Transforming XML to HTML by using the generated XSL stylesheet....138
Figure 5.25a: The Result of executing the Java Servlet Class for transforming XML to HTML by using GenXSLHTML.xsl....139
Figure 5.26: The Result of execution the Java Class (offline) for transforming XML to HTML by using GenXSLHTML.xsl....140
Figure 5.27: An Algorithm for generating XSL to transform XQuery to SQL....144
Figure 5.28: Diagram for generating an XSL stylesheet....145
Figure 5.29: Generic XSL Stylesheet for transforming XQueries to SQL Queries....147
Figure 5.29a: The Result of executing the Java Class for generating the XSL stylesheet (GenXQuerytoSQLTran.xsl)....148
Figure 5.30: Execution of Transformation from XQuery to SQL (xq1.ixq) in Table 9....150
Figure 5.30a: Execution of Transformation from XQuery to SQL (xq2.ixq) in Table 9....150
Figure 5.30b: Execution of Transformation from XQuery to SQL (xq12.ixq) in Table 9....151
Figure 5.31: Transforming XQuery to SQL (xq3.ixq) in Table 9 and generating XML Document from Database....151
Figure 5.32: Contribution of Algebra and XQuery in the Implementation....153
Figure 6.1: Association diagram....160
Figure 6.2: Use case diagram....162
Figure 6.3: Packages for the Museum System....163
Figure 6.4: Entity Relationship diagram....164

Figure 6.5: Component diagram..167
Figure 6.6: Deployment diagram..168
Figure 6.7: Institution information displayed..169
Figure 6.8: Institution information displayed on Tomcat server.......................170
Figure 6.9: Search and display museum objects..171
Figure 6.10: Museum objects displayed by the client..................................171
Figure 6.11: Search and display museum objects.......................................172
Figure 6.12: Objects with registration date display....................................173
Figure 6.13: Search and display the opened exhibition................................174
Figure 6.14: Opening exhibition information..174
Figure 6.15: Storage of objects in database...175

List of Publications

➢ Publications

❑ Papers

- **Paper 1.** A Tree-Based Algebra Framework for XML Data Systems, 7th ICEIS, Florida, USA, 25-28 May, pp. 305-312 (2005).

- **Paper 2.** XML Schema-driven generation of architecture components. 3rd ICEIS International Workshop on Web Services: Modeling, Architecture and Infrastructure. Florida, USA, 159-64 (2005); republished as pp. 137-142 in selected papers from Workshop.

- **Paper 3**. Virtual Exhibitions Framework: Utilisation of XML Data Processing for Sharing Museum Content over the Web, Proc CIDOC Annual Conference 2005, Zagreb, Croatia, 24-27 May, CD-ROM, paper 7, 16pp (2005).

- **Paper 4**. Mapping XML document to variety formats, 2nd ICETE International Conference on E-business Telecommunication. Networks, UK, 9pp (2005).

❑ Technical Reports

- **Technical Report 1:** XML Data Systems: A Tree-Based Algebra Framework with Worked Examples, Technical Report, School of Informatics, Northumbria University 11pp (2005).

- **Technical Report 2:** XML Schema-driven generation of architecture components and integration of on-line and off-line Systems. Technical Report, School of Informatics, Northumbria University 12pp (2005).

Chapter 1:

Introduction

As is well known, data is very important to every industry. The way data is stored and transferred between applications, and as a means to reduce redundancy and inefficiency have gained in importance. A great deal of work in managing data has been done for e-commerce applications using EDI (Electronic Data Interchange). Currently work is being done on XML (eXtensible Markup Language) for data handling. In today's world, data presentation over the web is of great concern. XML is one such technology that has created a revolution in handling and managing data [1]. XML started out as a way of presenting data over the Web and has later been extended to a wide range of applications because of its platform-independent interoperable nature. The features such as data presentation over the web and interoperable nature have made XML popular in a wide range of applications. The coming of XML has widened the scope for companies and technologists the world over. It has enabled them to explore possibilities in the area of efficient data exchange, presentation, querying, generation and portability of the data. XML documents are self-describing and platform independent. Also XML makes data exchange between different databases and different platforms relatively easy [2].
Furthermore XML is fast emerging as the dominant standard for *representing* and *exchanging* information between web applications and different platforms where it is relatively suitable. So if data is stored and represented as XML documents, then it should be possible to query the contents of these documents in order to extract, synthesize and analyze their contents; also the presentation, transformation and searching of data become simpler.

To be more specific, XML is a markup language defined by the W3C that provides a way to create an extensible format for describing structured data in electronic documents and to express rules about this data [37].

In recent years, XML has achieved remarkable success in handling structured data access on the web and has become increasingly recommended and supported by major software vendors [2]. Such success opens the door widely for new challenges, as the object-oriented approach becomes increasingly used by web programmers and within the design community.

One of the main challenges is the possibility for integrating data from various sources into a single unified framework and the support for homogeneous access from the web [2].

The central idea of the work is to a develop a common model of the data processed by the system in XML format and to use it consistently across all parts of the system (communication, validation, transformation, presentation, storage and retrieval) so that at any given point of the information flow there is a correspondence between the structure of information processed and the model.

In the following sections a more detailed explanation will be given of the aims and objectives of this research and of the research methodology. In addition, more details on the XML and related technology will be presented in Chapter 2.

1.1 Goals of the Research

Basically, this research is being done to achieve the main goals which can be summarized as follows:

- Integrated architectures for processing XML data typically incorporating several functions: communication (Web server) validation (XML Parser), transformation (XSL Processors), presentation (HTML), storage (SQL) and retrieval (XQuery) [83].

- Currently all these functions are addressed separately by existing approaches. *Bourret* [2, 149] adopted an existing solution and also generated relational schemas from XML ones and vice versa to simply hardcode a path through the object-relational mapping, which has a number of optional features. *Kajal T. Claypool* and *A. Rundensteiner* [47] presented the core of a modelling framework - a set of cross algebra operators that covers the class of linear transformations, and two different techniques for composing these operators into larger transformation expressions. *Chawathe* [52] discussed how XML data is queried, referenced and transformed using stylesheet language XSLT (eXtensible Stylesheet Language Transformation) and referencing mechanisms XPath and XPointer.

D. Lee and *Wesley W. Chu* [49] presented the semantic knowledge, which needs to be captured during transformation to ensure a correct relational schema and also adopted an algorithm that can derive such semantic knowledge from a given XML DTD (Document Type Descriptor), representing it as semantic constraints in relational database terms.

Shanmugasundaram et al [56] presented a general technique for querying XML documents using a relational database system, which enables the same query processor to be used with most relational schema generation techniques, and allows users to query seamlessly across relational data and XML documents. *DeHaan et al* [137] adopted a translation of XQuery expressions drawn from a comprehensive subset of XQuery to a single equivalent SQL (Standard Query Language), using a novel dynamic interval encoding of a collection of XML documents.

Sushant Jain et al [96] adopted an algorithm that translates the XSL (eXtensible Stylesheet Language) into an SQL query. The translation is based on a representation of the XSLT program as a query tree, which encodes all possible navigations of the program through the XML tree. More explanation on the existing approaches is presented in Section 2.3.5.

As we mentioned earlier, currently all these functions are addressed separately by existing approaches, causing overlapping, inflexibility and inefficiency of the solutions due to the lack of a common model of data and very specific data processing. To incorporate additional functions and achieve maximum utility, the systems are integrated so they implement several principally different data processing functions such as communication, validation, transformation, presentation, storage and retrieval and use several different tools for realising each one of them as a separate layer.

- Since the use of a common data model for processing data in a particular applied domain (e.g., personal financial information, corporate finances, cultural objects, etc.) has wide applicability, a potential solution, which addresses the problem, has many applications and could be quite useful in practice.

1.2 Terminology explanation

This section will introduce some terminology which is fundamental to this research. Explanations of the terms Integrated, Generic, Offline, Online, Prototype, Framework and Class of problems are provided as follows:

- INTEGRATED: The systems are integrated because they implement several principally different data processing functions (communication, validation, transformation, presentation, storage and retrieval) and use several different tools for implementing each one of them as a separate layer (web server for communication, XML parsers for validation, XSL processors for transformation, HTML browsers for presentation [6], databases for storage and XQuery interpreters for retrieval).

- GENERIC: This applies to a class of problems, not just one specific problem. The solution we are going to develop is generic because it will be equally well applicable to the financial and the cultural domain as long as the models of data to be processed are similar and the information processing can be formulated using the model.

- OFFLINE: All generic features of the solution to be developed will be outside of the integrated system and will be used to generate the specific components of the integrated system.

- ONLINE: All specific generated components will be used in the integrated system and displayed on the web. Online data exchange is performed in XML format.

- PROTOTYPE: This is an experimental implementation of an integrated system for XML data processing, which covers all the main functions needed for data processing (communication, validation, transformation, presentation, storage and retrieval) and uses all main components of the online part of the system (web servers, XML parsers, XSL processors, HTML browsers, databases and XQuery interpreters). The domain of virtual museum exhibitions has been selected as a domain of the prototype because of its relative simplicity, good standardization and wider applicability.

- FRAMEWORK: This is used in two different senses in the text - in its wider sense, framework could be understood as a combination of a theoretical model and technical implementation; in its narrowest sense, it can be understood as a technical environment for performing specific operations using particular technology for doing this (e.g., data transformation using XSLT is a framework for data processing).

- CLASS OF PROBLEMS: The interest is in how far the solution we have developed is generic, that is the extent to which it will be equally well applicable to the cultural domain as well as other domains, not just for the museum prototype domain. It would be expected that the algorithms we have developed for mapping the data can be used in other domains where the models of data and information processing are similar. It will also be assessed whether our algebra can be used in other domains.

1.3 Aims and Objectives of the research

Basically, the main aim of this research is to combine different technologies for XML data processing such as storing (SQL), presenting (HTML), querying (XQUERY) and transporting (Web services) using a common framework, which is both theoretically (XML algebra) and technologically well grounded. To be more specific, the objectives of the research are as follows:

- To describe and review the current use of XML and related technologies for processing data. This is presented in Chapter 2.

- To review existing algebra models of XML. This is presented in Chapter 3.

- To develop an algebraic model for data management, which unifies in a single framework data presentation, data communications and data processing based on an XML Schema. This is presented in Chapter 4.

- To design software architecture for data-persistent processing of XML data using generic software components (offline) for building multi-tier architectures (online), which utilize relational database, client-server technology and Web services. This is presented in Chapter 5.

- To implement a prototype system which demonstrates a new approach in the domain of virtual museums and joint exhibitions on the Web. This is presented in Chapter 6.

- To evaluate the work described in this thesis. This is presented in Chapter 7.

For this purpose, we will develop a common model of the data processed by the system in XML format, and use it consistently across all parts of the system (communication, validation, transformation, presentation, storage and retrieval) so that at any given point of the information flow, there is a correspondence between the structure of information processed and the model.

The approach is generic since both our model (XML algebra) and our problem solver (the architecture of the integrated system) is XML Schema driven. In the theoretical approach a prospective single algebra model is used everywhere because it is generic. Furthermore, from a technological perspective a single XML Schema is used everywhere for all kinds of offline generation and online component transformation.

Starting with the XML Schema of the data, we first develop domain-specific XML algebra suitable for data processing of the specific data and then we use it for implementing the main offline components of the system for data processing (XML Schema → SQL schema generation, XML Schema → HTML presentation and XML Schema → XML query interpreter).

In the online transformation multiple frameworks will be used to solve the separate problems in XML processing and to integrate them into a single architecture:

- XML Schema-driven integrated architecture for distributed information processing over the Web. The proposed solution uses an XML Schema for all kinds of transformation such as online and offline. The offline component is described above but the following will outline the online transformation components.

    - XML Schema driven storing (XML → SQL transformation).
      The XML Schema prepares for the storing of XML data in a database. In other words, the XML data will be transformed to SQL data and then stored in the database.
    - XML Schema driven querying (XQUERY → SQL transformation).
      In this component all specific queries using SQL are executed against the database, which contains the transformation data.
    - XML Schema driven generation (SQL → XML transformation).
      XML Schema will drive the generation of XML data from the stored data in SQL format. For the purpose of the integrated architecture XML data will be processed and this is why the data is retrieved and transformed into XML format.
    - XML Schema driven presentation (XML → HTML transformation).
      This is an example for presenting XML data.

1.4 The Choice of Methodology

According to Fraenkel and Wallen [151] there are a number of research methodologies such as experimental, correlational, descriptive, causal-comparative and survey research. Experimental research can truly test hypotheses concerning cause and effect relationships and communication. In an experimental study, the researcher manipulates at least one independent function, data or variable and observes the effect on one or more dependent functions, information or variables. In other words, the researcher determines which data will get which processing and transformation. The data are generally referred to as experimental and they control other data. Ideally, in experimental research the group of data to be studied is randomly formed before the experiment [153]. An experiment has a meaning in a conceptual framework. Starting from observations, a hypothesis has to be formulated and the experimental design will be organized according to the theory. The experimental design is constructed to verify a hypothesis, itself built according to empirical observations to reach the deductive demonstration as proposed in experimental methodology. It is necessary to generate the model according to the theory, which will be validated or invalidated.

Our research on the experimental study of Web architectures for data processing is based on mapping of XML Schema and the combination of different technologies for XML data processing such as the storing, presenting, querying and transporting formatting of data. Therefore, we will use the experimental research methodology, which is more appropriate for this research. The research method is focused on experimental implementation and development of relevant techniques for different types of transforming XML data, to generate the offline components and to test all these techniques and components by processing XML Schema. The main methods for the strategy to achieve the goals are as outlined below:

- A formal data model of XML data will be used as a universal algebra. Its complexity as well as the algorithms for various transformations will be studied using standard methods of discrete mathematics.

- A software prototype of the integrated system will be built using a J2EE [92] technological stack (Servlets, Web services, online and offline XSL transformations) using common XML schema and specific transformation algorithms.

To evaluate the software we will introduce different types of algorithms for transforming XML documents into a different format with XSL stylesheet transformation such as the offline and online transformation and also to implement and test all these algorithms with many types of the examples, checking that the actual result is as expected.

The proposed model integrates different approaches for processing XML data. The hypothesis is therefore that an integrated approach to processing XML data incorporating several principally different functions such as communication, validation, transformation, presentation, storage and retrieval will lead to an increased effectiveness and flexibility in information management.

1.5 Contributions

The originality of the work can be seen on different levels, such as in the theoretical approach (algebra model) and in the technical solution. Furthermore the central idea behind the research is to use a single theoretical model of the data to be processed by the system (XML algebra) controlled by one standard metalanguage specification (XML Schema) for solving a class of problems (generic architecture). The proposed work combines theoretical novelty and technological advances in the field of Internet computing. In the following paragraphs more details are given on the contributions.

- New domain-specific XML algebra for generic distributed problem solving.

- XML Schema-driven software architecture (online).

  Here the following components will describe the online components transformation:

  - XML Schema-driven storing (XML → SQL transformation).

    The XML Schema enables the storing of XML data in relation databases.
  - XML Schema-driven querying (XQUERY → SQL transformation).
  - XML Schema-driven generation (SQL → XML transformation).

    XML Schema will drive the generation of XML data from the data stored in SQL format.
  - XML Schema driven presentation (XML → HTML transformation).

- XML Schema-driven generation of architecture components (offline).

  Here we will describe the generation of the following offline components:

  - SQL schema generation.

    This component is related to the generation of SQL schema from the XMLSchema.

  - XSL Stylesheet generation.

    This component is related to the generation of all online stylesheets.

  - The XQuery interpreter generation.

    This component is related to querying the data.

- Algorithmic analysis of the algebraic transformations.

The contribution will be expanded and reviewed in Chapter 7.

1.5 Outline of this Thesis

The remainder of this dissertation is organized as follows: In Chapter 2, an overview of XML as universal data specification language and technologies for processing data is presented. The XML and JAXP Technologies related to the content of this dissertation are reviewed. In Chapter 3, an overview of XML algebra is presented, in other words XML algebra data models are reviewed. In Chapter 4 we present a framework for our XML algebra and XQuery in a specific domain, giving the theoretical basis for our work. Chapter 5 deals with the mapping of XML documents and develops the technological solution for the system. In Chapter 6, we present the prototype of the system to test the integrated approach. The work is evaluated, the contribution is assessed and the conclusion is presented in Chapter 7 followed by a discussion. A number of appendices are included. Appendix A contains the XMLSchema for our system. Appendix B contains java classes used in the implementation. Appendix C contains the implementation of the transformation of XQueries to SQL queries. Appendix D contains some of the more important UML diagrams such as analysis of use cases, activity diagram (showing more details on the use case) and sequence diagrams showing how the system processes use case activity.

Chapter 2:

XML as a Universal Data Specification Language and Technology for Processing Data

2.1 Introduction

In this chapter we provide some background on XML and related technologies for processing XML data that are related to the content of this dissertation. By giving an overview about the evolution of XML standard throughout the life of the Web, we introduce the main XML features and technologies such as SAX [8], DOM [5] and JDOM [4] parsing. The reason for reviewing different types of parsing techniques is to choose a suitable one for the validation of our XMLSchema. Also we review the XSL stylesheet transformation for processing the XML document and to use the XSL stylesheet for all kinds of transformation and storing XML documents in a database. The remainder of this chapter is reserved for a review of XQuery so as to use the XQuery language for querying XML data. Furthermore, we will review the DTD and XML schema and see which is more suitable for our work and in addition we describe the protocols such as HTTP [31] and SOAP [30].

2.1.1 Brief history

Generally, the word *markup* is used to describe the means of making explicit an interpretation of a text. Historically, the word is used to describe the annotation or other marks within a text, intended to instruct a compositor or typist how a particular passage should be printed or laid out.

Throughout the web's history, efforts have been made to represent these means into rules that can be built into a programming language. As a result of these efforts, many types of markup languages have been presented. The most common and the most used markup language today is HTML, which is used widely on the Internet. The common feature for all markup languages is that they present data in the same way. As an example of this in HTML, the start tag can be used to tell the browser that the coming stream of characters is text until the end tag is shown.

Going back to the very beginning of markup languages, the first markup language was created in 1969 by IBM [1, 59], namely Generalized Markup Language (GML).
At the time of creating GML, the main aim was to introduce a Meta language[1]. Over the years, a new version of GML has evolved, the Standard Generalized Markup Language (SGML).

Despite the fact that SGML was used significantly in commerce, it never had the same level of success as HTML due to its main downfalls: the cost of implementation and its complexity. To overcome such downfalls, the World Wide Web Consortium (W3C) joined forces in 1996 to develop a new version of the markup language that had both the power of SGML and the simplicity of HTML. XML was the result of these efforts. As is widely known, XML is a subset of the Standard Generalised Markup Language (SGML) defined in the ISO standard 8879:1986: designed and optimised for delivery over the web and to make it easy to interchange structured documents over the Internet [60].
To be more specific, XML (eXtensible Markup Language) is a meta markup language defined by the W3C that provides a way to create extensible format for describing structured data in electronic documents and to express rules about this data. In the following sections we will give more detailed explanations about XML and its technical features supported with some examples.

2.1.2 XML vs. the XML Processor

The main core of XML technology is based on the XML processor, which is a software module used to read XML documents and provide access to their content and structure. In other words, the XML processor reads every piece of information of a document[2]. In general, XML documents are made up of storage units called *entities*. These *entities* contain either parsed or unparsed data. Parsed data is made up of *characters*, some of which form the *character data* in the document, and some of which form markup. Figure 2.1 is an example showing the basic structure of an XML document.

[1] A language used to describe other languages
[2] It is assumed that an XML processor is doing its work on behalf of another module called application - www.w3c.com.

Figure 2.1: An example of an XML Document

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <Students>
    <Student1>
       <Student-no>001</Student-no>
       <Student-name>Bob Mark</Student-name>
       <Student-address>
           <Street> 19 Stanhope</Street>
           <City>Newcastle</City>
           <Postcode>NE4 5JF</Postcode>
        </Student-address>
    </Student1>
    <Student2>
        <Student-no>002</Student-no>
        <Student-name>Alex Mark</Student-name>
        <Student-address>
            <Street> 07 Avison</Street>
            <City>Newcastle</City>
            <Postcode>NE4 5PY</Postcode>
         </Student-address>
    </Student2>
 </Students>
```

Functionally, the XML processor checks if the document follows all the syntactic rules defined in the XML specification. However, for a document to be valid it must follow all the specifications defined by DTD such as namespace or scheme for the document. Briefly, XML provides a mechanism to impose constraints on the storage layout and logical structure.

2.1.3 XML's design goals

Since its initial development by the W3C Generic SGML Editorial Review Board, XML has a number of main design goals. These goals as described by the XML Working Group [3] for XML are:

- XML shall be straightforwardly usable over the Internet.
- XML shall support a wide variety of applications.
- XML shall be compatible with SGML.
- It shall be easy to write programs which process XML documents.

[3] Originally known as the SGML Editorial Review Board.

- The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
- XML documents should be human-legible and reasonably clear.
- The XML design should be prepared quickly.
- The design of XML shall be formal and concise.
- XML documents shall be easy to create.
- Terseness in XML markup is of minimal importance.

Simply, by achieving these goals, the proposed markup language will be readable by both humans and machines. Furthermore, this markup language can deal with a general standard for data interchange.

2.1.4 XML's main usage

Basically, the main usage of XML technology can be summarised into three main categories.

- Retrieving the data via file-based or remote procedure methods.
- Extracting the required information (XML query language or scripted language coding with standard XML packages).
- Analysing single subset or an entire database.

Such features came as a result of continuous efforts to achieve the main goals formed by the World Wide Web Consortium (W3C) in 1996, to satisfy the main characteristics of the proposed generic markup language (XML) for ease of implementation and interoperability with both SGML and HTML [16].

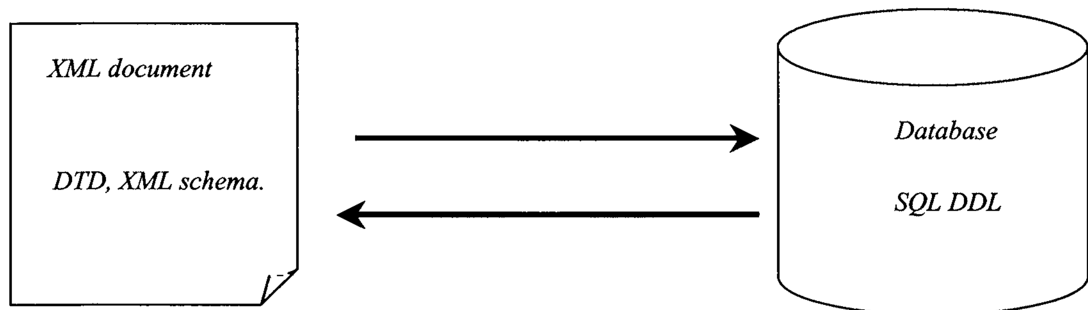
2.2 XML vs. DB

Generally, XML and its related technologies have many things in common with real databases. However, XML lacks some data base features such as storage, indexes, security, transactions and data integrity, multi-user access, triggers, queries across multiple documents, and so on [2].

Despite these disadvantages, XML still takes the lead in describing a user-defined database format; the markup describes the structure and type names of the data although not the semantics.

Thus, XML documents are a wise choice as a database in an environment with small amounts of data, few users, and modest performance requirements. In such cases, integration requirements are recommended to be very limited.
In order to handle data transfer between XML documents and a database, it is necessary to map XML document schema (DTD, XML Schema, RELAXNG, etc.) The software may use an XML query language (such as XPath, XQuery, or a proprietary language) or simply transfer data according to the mapping (the XML equivalent of SQL standard (SELECT * FROM Table)) [2]. Figure 2.2 is a diagram simplifying the main idea behind the XML schema mapping.

Figure 2.2: XML Schema/Database mapping



However, transforming XML documents into other documents needs certain elements to be specified, certain rules to be applied and software tools to be involved in the process in order to satisfy XML goals.

In the following sections, an explanation about XML and JAXP technologies such as XML parsing techniques and XML transformation will be given in more detail. Firstly we explain the principles of stylesheet transformation by focusing on the XSL stylesheet, XML Query languages, document type definitions (DTDs) and XML Schema [9], which specify the elements that are or are not allowed in an XML document and secondly, we explain protocols such as HTTP and SOAP.

2.3 XML and JAXP Technologies

JAXP, Java API for XML Processing, is an application to parse and transform XML documents using a pure Java API that is independent of a particular XML processor implementation. This allows flexibility in swapping between XML processors without making application code changes. Also JAXP supports the processing of XML documents using DOM, SAX and JDOM.

2.3.1 XML Parsing Techniques (SAX, DOM and JDOM)

Basically, XML parsing is an activity needed to validate the structure of an XML document. It is necessary to ensure that the XML document conforms to the DTD or XML schema of the various types of documents. Likewise, a parser is required to obtain the data items from an XML document. It reads the XML documents, then extracts the actual data and provides application programs with access to their content and structure.

In the last few years, there have been several efforts to employ the XML parser. The entire implementation in this research work uses Apache Xerces-J XML parser [3], written in the Java programming language. Thus it is very suitable in the Java environment used in this research and it supports the SAX and DOM approach. There are two types of parsing technique, a *tree constructing parsing* and *event-driven parsing:*

- *Tree constructing parsing* reads the input XML document in order to construct a tree and delivers it to the application for further processing. An application program interface (API) called Document Object Model (DOM) is recommended by W3C to support this type of parsing. The latest DOM is Level 3 version 1.0 [5]. The DOM API is designed to be compatible with a wide range of languages, including scripting languages and more sophisticated languages.

- *Event-driven parsing*
  - *Event-Based* APIs report parsing events directly to the application through callbacks.
    - Start Document
    - Start Element
    - End Element
    - End Document
  - Application enables handlers to deal with events.
  - Event-Based API provides a simpler, lower-level access to an XML document.
  - It is possible to parse large documents much larger than a whole memory system
  - Application builds up data structure using callback event handlers.

2.3.1.1 SAX API

The Simple API for XML (SAX) [8] was developed by the members of the XML-DEV mailing list and defines a framework of events for parsing XML documents. The SAX parser reads in XML content by returning constituent parts (e.g. elements, attributes and text) in the form of events. The SAX API is more lightweight than DOM and the fact that only events have to be filtered rather than loading the entire content into memory, makes SAX a much more efficient parser in terms of time and memory. However, this efficiency comes at a cost because essentially the XML structure does not persist and is not stored in the memory.

In addition, each event takes place as the document is parsed, not after the parsing has occurred. It allows a document to be handled sequentially without having to first read the entire document into memory. While the parser reads the XML document, the SAX API can quickly filter the needed data items and collect them. Also the converter discussed here only deals with element contents and attributes.
So after the parser finishes reading the XML document, the collection process has also finished.

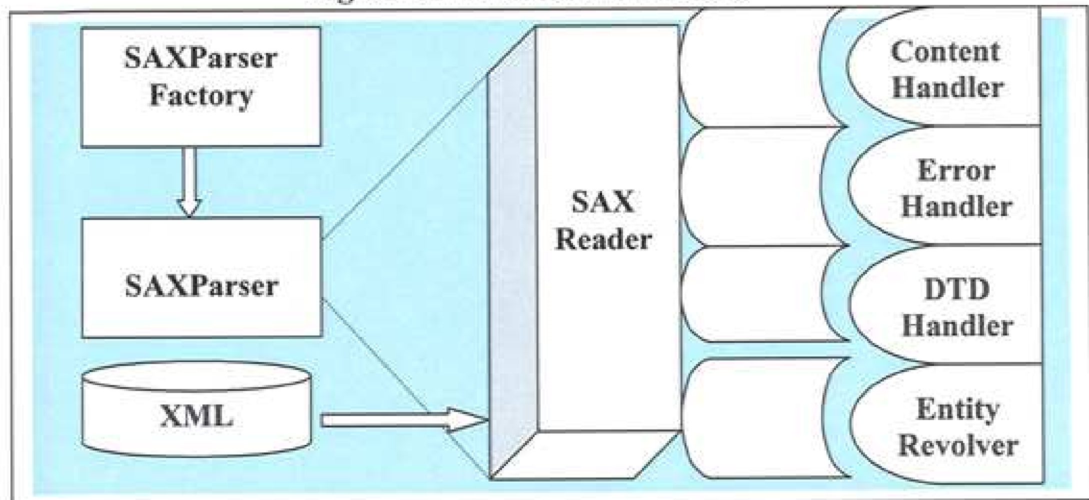
SAX 2.0 specifies several event handlers, such as **DTDHandler**, **ErrorHandler**, **ContentHandler**, **Lexical Handler** and **DeclHandler.** Table 1 simplifies the main event handlers used by the SAX.

Table 1: Main Event Handlers

Handler name	Type	Function
ContentHandler	Required	All standard contents reading events
ErrorHandler	Optional	Captures errors during reading the XML file
DTDHandler	Optional	Handles events in a DTD
LexicalHandler	Optional	Handles comments, entities and CDATA nodes
DeclHandler	Optional	Handles declaration of attributes and entities in DTD

Furthermore it specifies the events of grammatical structures that are found in 95% of the world's XML applications [5]. Figure 2.3 shows the Event SAX handlers as presented in Table 1.

Figure 2.3: Event SAX handlers



2.3.1.1.1 How SAX Works

The following paragraphs introduce different examples to show how SAX works and describes parsing the XML document [13]. Table 2 shows events in parsing the example below.

- XML Document.

```
<?xml version="1.0"?>
<Student>
   <student_id>Y0904</student_id>
   <student_name>Ali omer</student_name>
</Student>
```

An event-based interface will break the structure of this document down into a series of linear events as presented in Table 2.

Table 2: Events in Parsing an XML document

No	Events	Methods
1	Start document	startDocument()
2	Start element: *Student* Start element: *Student_id* characters: *Y0904*	startElement()
3	End element: *Student_id* Start element: *Student_name* characters: *Ali Omer*	endElement() characters()
	End element: *Student_name* End element: *Student*	endElement()
4	End document	endDocument()
5	StartPrefixMapping	StartPrefixMapping()

There are five event methods from the **ContentHandler** interface, such as a Start and End element that are used for XML source processing. When SAX finds an xmlns attribute the **startPrefixMapping ( )** callback is triggered to obtain the namespace definition. The namespaces are stored in the form of URI and prefix. According to the specification, the URI is the primary key.

While SAX is traversing the tree structure, a context node is maintained. The context node is the current node position of the parser cursor. Each time the parser moves to a new node, the current node is recorded. When the attribute value or element content is found, then recorded location steps are concatenated to form a location path.

- **SAX role on checking XML structure**

The parser has specific functions that are activated when processing the XML document. Parsing can be performed on the operational source XML and events on:

- startDocument()

  This function marks the beginning of the document and reports to the document handler. The purpose of the function is for initializing certain parameters.
- startElement()

  This function is called up whenever the parser encounters the start of an element.
- endElement()

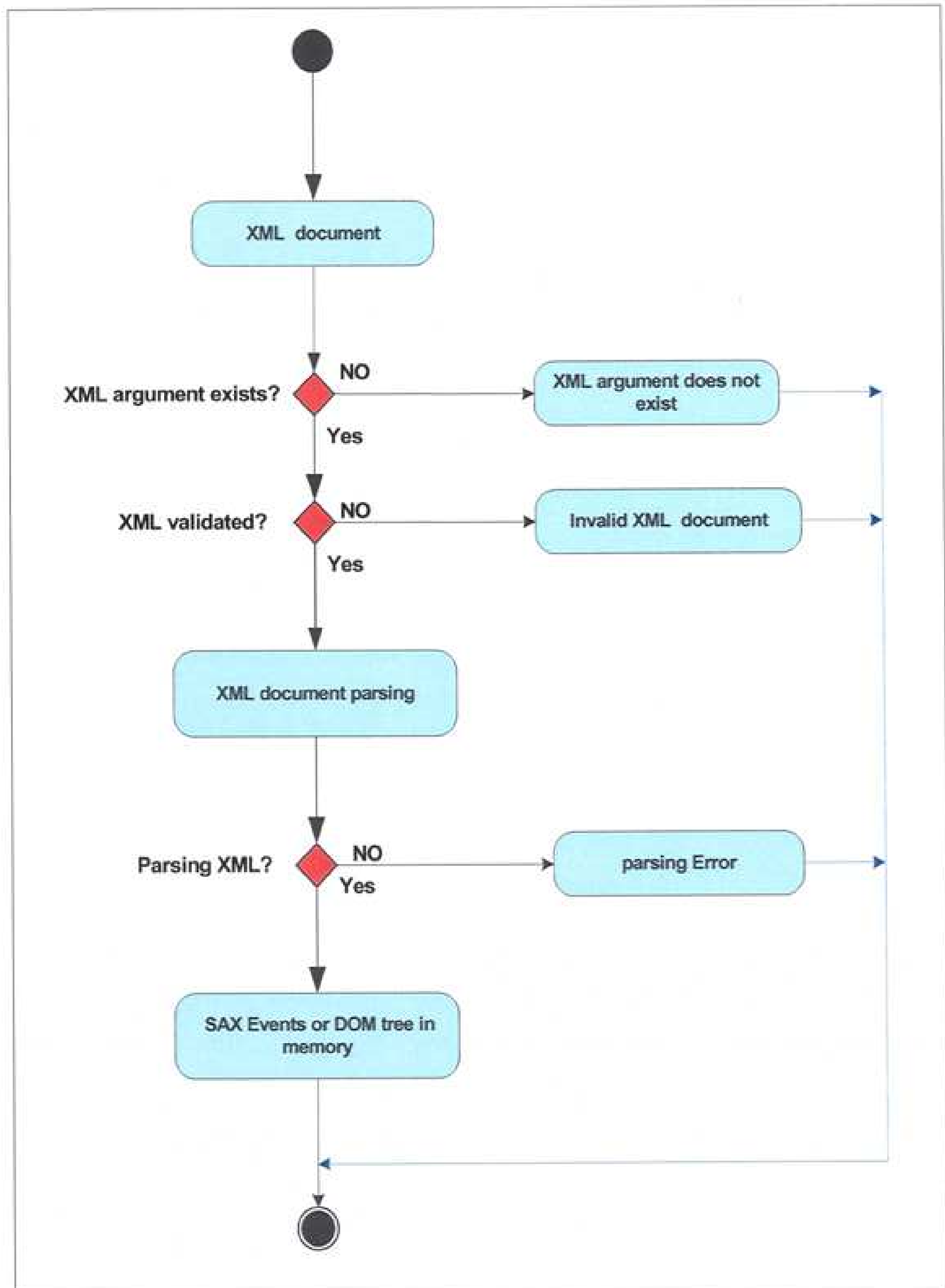
  This function marks the end of an element. In the document handler various flags and variables are initialized.
- characters()

  This function gives the data enclosed by the tags. Also it compares the character data location path and data under elements node with the source structure.
- endDocument()

  This function marks the end of the document.

The activity diagram in Figure 2.4 shows the decisions required for the technique of parsing XML documents in general.

Figure 2.4: Diagram to show Parsing of an XML Document

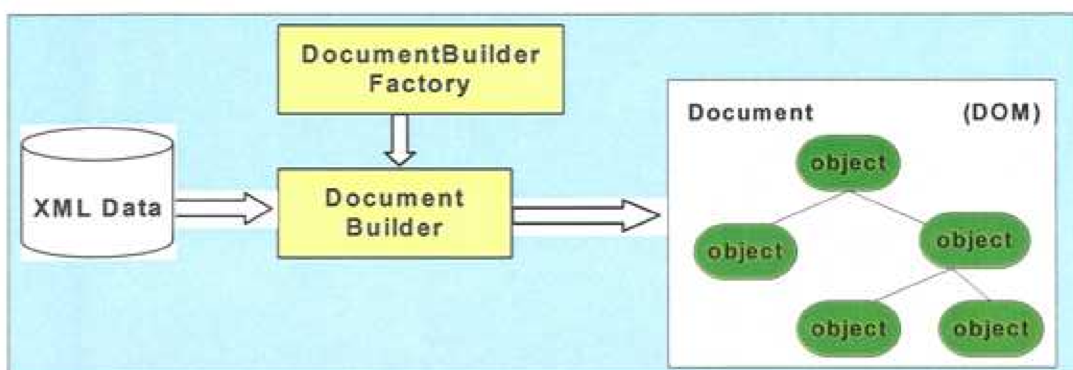


2.3.1.2 DOM APIs

Basically, the Document Object Model (DOM) [5] is a specification that comprises a set of interfaces that allow XML documents to be parsed and manipulated in the memory. These interfaces are defined by the W3C who provide application-programming interfaces (APIs) for every major programming language using Interface Definition Language. To be precise, the DOM is not designed specifically for Java, but to represent the content and model of documents across all programming languages and tools. Bindings exist for JavaScript, Java, CORBA and other languages, allowing the DOM to be a cross-platform and cross-language specification. DOM has been specifically designed by the W3C to model XML documents in a tree structure, beginning with a root element and comprising any number of child and sibling nodes. When an XML document is parsed by DOM, the entire content of the document is read into the memory and providing it is a well-formed document, is then represented in the tree structure. The elements can then be accessed and traversed from the root node using the tree relationships.

The main interfaces for a DOM application are as follows **Node:** this is the base type, representing a node in the DOM tree. **Document:** represents the entire XML document as a tree of Nodes (the DOM parser will return Document as a result of parsing the XML). **Element:** represents elements of the XML document. Every pair of tags <someTag>…</someTag> forms an element in the DOM tree, and all XML tags between the start-tag and the end-tag will be represented as child elements. **Attar:** represents an attribute of some XML element; the interface enables setting/getting the value of that attribute. **Text:** is used to represent the text content of an element (i.e. the text between tags that is not part of any child element). Figure 2.5 shows the DOM APIs in action:

Figure 2.5: JAXP APIs in Action



2.3.1.3 JDOM APIs

JDOM [4] is a way to represent an XML document for easy and efficient reading, manipulation and writing.

Specifically, in the spring of 2000, Jason Hunter and Brett McLaughlin in order to modify an XML document in a tree structure mode devised JDOM. JDOM has now been developed as an open-source API for XML and has been accepted by the Java Community Process.

The objective of JDOM is to provide a fast and more lightweight API for Java-based applications. JDOM is not a standalone parser and relies on either DOM or SAX to build a JDOM representation of the XML in memory.

However, once in the memory JDOM provides a class-based API for manipulating the XML content and this is much more efficient than DOM, which uses interfaces instead. Also, XML components have their own class types in JDOM below and are not as strongly typed as DOM. This is because DOM uses a common node interface from which every element is derived. In summary the JDOM API has the advantages of DOM of being able to manipulate XML in memory but combines this with the performance of the SAX parser by being purely a Java implementation.

As an example the following classes show how the methods for an XML element allow the user to get and manipulate its child elements and content, directly access the element's textual content, manipulate its attributes and manage namespaces. Also for an XML document, methods allow access to the root element as well as the DocType and other document-level information.

```
org.jdom.Element root = new org.jdom.Element ("XMLdoc")
org.jdom.Document doc = new org.jdom.Document (root)
```

In addition, JDOM also provides a full document view with random access but, surprisingly, it does not require the entire document to be in the memory. Additionally, JDOM supports easy document modification through standard constructors and normal set methods.

2.3.2 Summary of comparison of different XML Parsing Techniques (SAX, DOM and JDOM)

Table 3 provides a review summary of different XML parsing techniques, such as SAX, DOM and JDOM discussed so far [7].

Table 3: Comparison of different XML Parsing Techniques (SAX, DOM, JDOM)

Parsing / Categories	SAX (Simple API for XML)	DOM (Document Object Model)	JDOM (Java and DOM)
Techniques	Event-driven	Object-based	Reads and writes DOM documents and SAX events
Storing	Does not store data from XML document during parsing	Constructs in-memory copy of document tree	Open source project
Modifying	No support for modifying or writing XML document data	In-memory document tree can be modified	It is not built on DOM or modelled after DOM
Processing	Document data becomes available as it is parsed	Entire document must be parsed before processing by client application	Can use any DOM or SAX parser
Performance	Faster	Slower	Slower

2.3.3 XML Transformation

Having information stored in XML format is not enough if information is to be addressed to people rather than computers. As stated earlier, XML describes the contents of the data not the way it is presented. Therefore, there should be a way of taking an XML document and making it appear in legible format.

More specifically, a stylesheet description language should be used in order to present the same information in a different format when put on a computer screen or printed on paper or when projected through loudspeakers for the visually impaired. The W3C is trying to produce a new stylesheet language called eXtensible Stylesheet Language (XSL), which is going to be as extensible as XML, and appropriate for any document type. In other words an XSL is a language for expressing stylesheets. It consists of three parts: XSLT [11], XPath [10] and XSL formatting objects. Figure 2.6 is an example showing the basic structure of an XSL stylesheet. In Chapter 6 we will walk through an XML schema-driven transformation by using an XSL stylesheet for all kinds of transformation.

Figure 2.6: A Fragment of XSL sylesheet

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <!-- The bulk of the XSLT document goes here  -->
    </html>
  </xsl:template>
</xsl:stylesheet>
```

2.3.3.1 Why is an extensible document transformation method needed?

In general, any application that has the capability to work with XML documents will need to display the structure of its related data into a variety of formats as specified by the user on different occasions, due to the nature of working in heterogeneous environments. Accordingly, transforming a document from one data structure to another is needed. Such a transformation process is essential, especially when dealing with XML documents.

Of course, designing “traditional” software transformation tools for that purpose can deploy such a task. Consequently, the power of having a cross-platform and an XML independent language would be lost. Precisely, isolating content from formatting needs to be considered, especially when dealing with Web based documents.

Therefore, any method for transforming XML documents into different formats such as HTML, flat files or WML needs to be designed so that it can be used with different platforms/languages.

2.3.3.2 eXtensible Stylesheet Language Transformations (XSLT)

In fact, XML by itself is not intended for data presentation purposes. Originally, XML was created to meet the challenges of *data exchange* on the Web applications or between applications and users, not for data presentation purposes.

To deal with presentation issues, XML needs to be used in conjunction with *Stylesheets* to be easily viewed on the web. For this purpose, the Extensible Stylesheet Language was created. XSL (eXtensible Stylesheet Language) is being developed as part of the W3C stylesheets activity [12, 140]. It has evolved from the CSS language to provide even richer stylistic control, and to ensure consistency of implementations. It also has document manipulation capabilities beyond styling.

Basically, the XML stylesheet language (XSL) has three major subcomponents as follows:

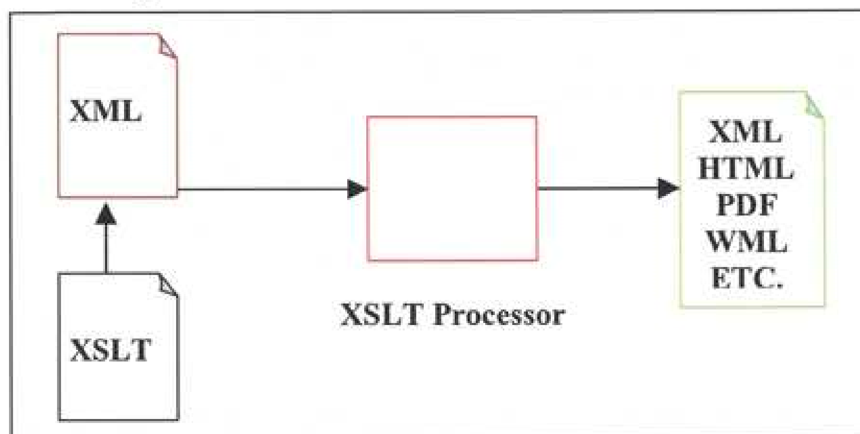
- XSLT [11] which is a transformation language, used to transform XML documents into other formats.
- XML Path Language (XPath) [10] is a language for addressing parts of an XML document; this is essential for cases where we want to say exactly which part of a document is to be transformed by XSL.
- XSL-FO (XSL Formatting Object), used as an XML vocabulary for specifying formatting semantics. W3C developed and standardized the XSL. The XSLT document can be reused later for any transaction between sources and destination relationship with given XML and HTML structures.

Besides the XML document transformation activity, XSLT is used as a general programming language like C, C++ or Java and can be used to code other XML transformation methodologies, such as data binding and tree modification. However, XSLT has the advantage of being more lightweight than these languages [6].

2.3.3.3 The Transformation Process

The process of transforming an XML document into another format, such as HTML, XML or WML, relies on three kinds of processing engines. First, a parser capable of loading an XML document must be present to load the source document into a DOM tree structure. Second, the XSLT document must be loaded and a tree structure will be created for it as well. This tree structure will normally be optimized to accommodate XSLT processing and is specific to the processor being used. An XSLT processor is then needed to take the XML document structure, match up nodes within the document against "templates" found in the XSLT document, and then output the resulting document. The third tree structure (the resulting document) is dynamically created based on information contained in the XSLT document. A simple diagram of this transformation process is shown in Figure 2.7.

Figure 2.7: The XSLT Transformation Processor



Generally, in the process, XSLT assumes three documents are in use: the source document, the XSLT Stylesheet document and the result document. An XSLT processor will perform a tree transformation from the source document and construct a new document based on instructions in the stylesheet. The stylesheet document is an XML document that uses the XSLT vocabulary to express transformation rules. Furthermore the XSLT stylesheet contains one or more templates. A template is a collection of literal result elements and the XSLT instructions. XSLT instructions are well-known elements that alter the processing of the template. XSLT instructions are always qualified by the XSLT namespace URI (http://www.w3.org/1999/XSL/Transform), which is typically mapped to the namespace prefix XSL.

2.3.4 Review of approaches related to the transformation of XML documents

The review gives us a choice of existing solutions for application to XML documents. As can be seen, *Ronald Bourret* [2, 97, 149] adopted an existing solution with integrated and generic solutions and also generated relational schemas from XML ones and vice versa to simply hardcode a path through the object-relational mapping, which has a number of optional features. Moreover, he gives a high-level overview of how to use XML with databases.

Kajal T. Claypool and *A. Rundensteiner* [47, 113] presented the core of a modelling framework - a set of cross algebra operators that cover the class of linear transformations, and two different techniques for composing these operators into larger transformation expressions. They also presented an evaluation strategy to execute the modelled transformation and thereby transform the input schema and data into the target schema and data. The strategy assumes that data model wrappers are provided for each data model.

Chawathe [52] discussed how XML data is queried, referenced and transformed using stylesheet language XSLT and referencing mechanisms XPath and XPointer.

Yuan Wang, David J. DeWitt and *Jin-Yi Cai* [109] adopted an algorithm that integrates key XML structure characteristics with standard tree-to-tree correction techniques, for computing the optimal difference between two versions of XML documents.

M. Klettke and *H. Meyer* [48] presented an algorithm that finds some kind of optimal mapping based on statistics such that the XML becomes the standard for the representation of structured and semi-structured data on the Web. Relational and object-relational database systems are a well-understood technique for managing and querying such large sets of semi-structured data.

D. Lee and *Wesley W. Chu* [49] presented the semantic knowledge, which needs to be captured during transformation, to ensure a correct relational schema and also adopted an algorithm that can derive such semantic knowledge from a given XML DTD, representing it as semantic constraints in relational DB terms. By combining existing transformation algorithms and constraint-preserving algorithms, one can transform XML DTD to relational schema.

Shanmugasundaram, E. Shekita, J. Kiernan, R. Krishnamurthy, E. Viglas, J. Naughton and *Igor Tatarinov* [56] presented a general technique for querying XML documents using a relational database system, which enables the same query processor to be used with most relational schema generation techniques, and allows users to query seamlessly across relational data and XML documents. The proposed technique can thus serve as the infrastructure for investigating issues common to different relational schema generation techniques, such as document caching, updates and information retrieval style querying.

Francis Norton [135] presented an XSL as a validation to test XML document and returning an XML document that contains a list of invalid elements or is a valid document or empty.

DeHaan D., D. Toman, M. P. Consens and *T. Ozsu* [137] adopted a translation of XQuery expressions drawn from a comprehensive subset of XQuery to a single equivalent SQL query using a novel dynamic interval encoding of a collection of XML documents.

Rajasekar Krishnamurthy, Venkatesan T. Chakaravarthy, R. Kaushik and *Jeffrey F. Naughton* [134] adopted a generic algorithm that translates path expression queries to SQL in the presence of recursion in the schema in the context of schema-based XML storage shredding of XML as relations.

Sushant Jain, Ratul Mahajan and *Dan Suciu* [96] adopted an algorithm that translates XSLT into SQL. The translation is based on a representation of the XSLT program as a *query tree*, which encodes all possible navigations of the program through the XML tree.

2.3.5 Characteristics of our Approach

As a result of the review given in section 2.3.4, it is shown that none of them introduces an algorithm to generate a generic XSL stylesheet for transforming XML data, to generate an SQL schema and to enable the XQuery interpreter generation of XMLSchema. Furthermore, several existing approaches propose mappings between XML documents and relational databases, but the query translation problem from XML queries to SQL queries has received less attention. But in our thesis we will introduce certain general techniques as a technology solution for different problems such as (a) generating SQL schema from XMLSchema, (b) generating XSL stylesheet from XMLSchema, and (c) XQuery interpreter generation.

Each of the techniques proposed in this thesis works by XMLSchema-driven generation of architecture components with an XSL stylesheet. As can be seen the input is XMLSchema and XSL stylesheet and the output is generic stylesheets. These stylesheets can be used for generating other types of data such as SQL queries from XQueries, SQL data, SQL schema and HTML format. We will present algorithms for all kinds of generation, and the algorithms will show how we can generate these components automatically. More details are presented in Chapter 5. In addition to the characteristics of our approach it is also necessary to compare the DTD and XML schema as presented in the following sections.

2.3.6 Document and XML document

The word "*document*" refers not only to traditional documents, but also to the myriad of other XML "data formats". These include vector graphics, e-commerce transactions, mathematical equations, object meta-data, server APIs and many other kinds of structured information.

2.3.6.1 Document Type Descriptors (DTDs)

Basically, the Document Type Descriptors (DTDs) [81] describe the structure of the XML documents and are like a schema for XML documents. A DTD shows the contents of an XML element by specifying the names of its sub-elements and attributes. The sub-element structure is specified using the * sign, declaring that the message can occur zero or more times inside the note element; + sign, declaring that the element message must occur one or more times inside the note element; ? (optional), and | (or). All values are assumed to be string values unless the type is ANY, in which case the value can be an arbitrary XML fragment. Each element can also have many attributes. There is a special attribute of type ID, which can occur once for each element. The id attribute uniquely identifies an element within a document and can be referenced through an IDREF attribute in another element. IDREFs are untyped in the sense that they can point to the id field of any element. There is no concept of a root of a DTD; an XML document conforming to a DTD can be rooted at any element specified in the DTD [81]. Figure 2.8 shows an example DTD specification, which specifies the schema for a class of XML .

Figure 2.8: An Example of DTD

```
<!ELEMENT item (title? , item*, metadata? ) *>
<!ATTLIST item identifier ID #REQUIRED
isvisible CDATA #IMPLIED
parameters CDATA #IMPLIED
idntifierref CDATA #IMPLIED
a-dtype NMTOKEN 'isvisible boolean' >
<!ELEMENT organization (title? , item*, metadata? ) *>
<!ELEMENT organization identifier ID #REQUIRED >
<!ELEMENT organizations (organization* )>
<!ATTLIST organization default IDREF #IMPLIED >
```

2.3.6.2 XML Schema

Functionally, the Document Type Definition (DTD) of XML documents can serve only to validate their syntactical correctness, and is not well suited to support complicated computations on their content. Therefore, new models are proposed for document content and content processing. The purpose of the XML schema language is to provide an inventory of XML markup constructs with which to write schemas. The purpose of a schema is to define and describe a class of XML documents by constraining and documenting the meaning, usage and relationships of their document parts. The schema specifies data types, elements and their content, attributes and their values, entities and their contents and notations. Schema constructs may also provide for the specification of implicit information such as default values.

Schemas document their own meaning, usage and function. Thus, the XML schema language can be used to define, describe and catalogue XML vocabularies for classes of XML documents [41,138]. Figure 2.9 shows a part of our XML Schema specification.

Figure 2.9: A Fragment of XMLSchema for the Proposed System

```
<element name="Archive" type ="Archive"/>
      <complexType name= "Archive">
        <complexContent>
          <extension base="Exhibition">
            <all>
              <xelement name="arch_id" type ="Integer"/>
              <element name="purpose" type ="String"/>
              <element name="accessibility" type ="String"/>
              <element name="exhibition_id" type ="Integer"/>
            </all>
          </extension>
           <element name="attended_by" minOccurs="1">
             <complexType>
               <choice minOccurs="1" maxOccurs="unbounded">
                 <element ref="expert"/> </choice>
             </complexType>
           </element>
        </complexContent>
      </complexType>
```

2.3.6.3 Comparison of DTD and XMLSchema

As a result of our review of DTD and XMLSchema, we can conclude that DTD defines the schema of an XML document [120]. It describes the structure of an XML document in terms of elements, attributes and its data types, valid ordering, nesting and so on. DTD has some major shortcomings like shallow support for data types, no support of namespaces and not being in XML format itself, while the XML Schema [116] is another emerging standard that takes care of the problems of DTD.

XMLSchema can be used to define and describe a class of XML documents by constraining and documenting the meaning, usage and relationships of their document parts. The schema specifies data types, elements and their content, attributes and their values, and entities and their contents and notations. Also, the XML schema language can be used to define, describe and catalogue XML vocabularies for classes of XML documents. Because of this we find the XMLSchema is more suitable for our work and we will use the XMLSchema for all kinds of generation and transformation components as well as in our XML algebra.

In addition to compare DTD and XML schema it is also necessary to review XQuery data model and storing XML documents as presented in the following sections.

2.3.7 XQuery Data Model

Generally, the XQuery [35, 55] data model defines the information contained in the input and output from an XQuery processor. It is based on the XML information set with additional features including, among other things, support for XML Schema data types. In essence, an instance of the XQuery data model may consist of nodes, simple values or sequences [71]. A node can be one of the following types: *document node, element node, attributes node, namespace node, and comment node, processing instruction node, text node* and *reference node.* In the following we will discuss in more detail the XQuery and path expression [87].

2.3.7.1 XML Query Language

Generally, as increasing amounts of information are stored, exchanged and presented using XML, the ability to intelligently query XML data sources becomes increasingly important. One of the main strengths of XML is its flexibility in representing many kinds of information, including information traditionally represented in databases and information traditionally represented as documents. To exploit this flexibility, an XML query language must include the features that are necessary for retrieving information from these diverse sources [40]. A number of XML query language features have been proposed [19, 25, 29, 44, 45, 53, 54, 126]. We will take XQuery, which was recently proposed by the W3C, as an example to discuss important query features for XML documents.

Basically, the XQuery is designed to meet the requirements identified by the W3C XML Query Working Group. XQuery is designed to be a small, easily implementable language in that queries are brief and easily understood and also flexible enough to query a broad domain of XML information sources, including databases and documents. The main principle forms of XQuery expressions consist of the following: Path expressions, element constructors, FLWR expressions, expressions involving operators and functions, conditional expressions, quantifiers and variable bindings. The FLWR (FOR-LET-WHERE-RETURN) expressions provide a SQL like syntax for extracting data, performing selections on the extracted data, and returning results in a structure of the user's choice.

The XQuery syntax the way it uses clauses like FOR, LET, WHERE and RETURN is approximately similar to FROM, WHERE and SELECT in SQL. The differences between a FOR and a LET clause is that FOR binds a variable to each element specified by the path expression while the LET binds a variable to the whole collection of elements.

We consider the clauses in the FLWR expression in the sample query:

```
For std IN <Students>
Return <student>
Where <Student-no=002>
```

The FOR clause iterates over the sequence of nodes returned by the path expression, binding the variable std to each of the student nodes returned. Once the variable is bound, we can further access any of its sub-parts by using XPath expressions.

```
WHERE student-no= "002"
```

The WHERE clause selects those student nodes where the student-no child element is equal to "002".

```
RETURN student
```

The RETURN clause, in this FLWR expression, constructs student elements, including the student-no as a child element [39].

The syntax for a FLWR expression can be expressed concisely as follows: FOR and LET clauses generate a list of tuples of bound expressions [38]. In Chapter 4 we will introduce our XQuery framework and also we will generate XSL stylesheets for transforming XQuery to SQL queries [137].

2.3.7.2 Path Expressions

Basically, the path expression XPath [35, 36] provides a way to address specific parts of an XML document by providing a path to the content of interest in the document tree. For example, suppose we want the address of the Student2 in the XML document shown in Figure 2.1.

In the preceding query, to refer to the student address, which is a child of the student's root element in the document "students.xml", we use the path expression: Students/Student2/Student-address.

The expression above will return the address of Student2 in the students.xml with the result as shown below.

```
<Student-address>
    <Street> 07 Avison</Street>
    <City>Newcastle</City>
    <Postcode>NE4 5PY</Postcode>
</Student-address>
```

2.3.8 Storing the XML document in Database

Generally, there are several experimental methods that have been developed and adopted in order to accomplish effective and genuine translating of the XML document into relational schemes and store it in the database [115,141] using two different methods. The first is mapping from DTD and the second is the Edge Model method. When mapping from DTD, since the Document Type Definition (DTD) describes the structure of the XML document, then DTD can be used as the basis to generate relational schemas for the XML document [42]. W3C simplify DTD to reduce its complexity without losing any information from the structure. Then W3C transform DTD into a directed graph where the elements are represented as nodes and their containment relationships are represented by arcs. In order to generate relations from elements, the DTD graph is searched to generate a spanning tree for each element.

The Edge Model [43], a directed graph of XML document, is stored in a single relational table called the 'Edge Table'. Every node (XML element) in the directed graph is assigned an id. Each tuple in the edge table corresponds to one edge in the directed graph and contains 1) the ids of the source and the target of the edge, 2) the tag of the target element and 3) an ordinal number that is used to record the position of the nodes. When an element has only one text child, the text is stored with the edge.

The example below shows an edge table for the following schema.

Edge_Table (sourceID, ordinal, flag, tag, targeted, data).

The Object Exchange Model originally proposed for semi-structured data [66, 117] is extended to handle XML data.

In the OEM [118] model the XML data is modelled as a directed labelled graph in which the nodes represent the data elements and the edges represent the element-subelement relationship. However, there are limitations to this model such as lack of definition of classes or types, lack of support of encapsulation and the difficulty of expressing cardinality and option properties.

Finally, in an object repository [43], the XML document is decomposed into elements and stored as objects in the repository. The format of an element object is: length, tag, parent, prev, next, flag, child, attar, text.

As shown previously, an element object is used as its identifier, where the length field records are the total length of the current object and the tag field is the tag name of the XML element. The parent field records the id of its parent node; the sibling list of an element is implemented as a doubly linked list via the prev and next pointers. Also the flag field keeps the information indicating whether the current object contains any child node, attribute, or textual content and the last three fields are optional.

2.3.9 Transport

The transport layer defines capabilities of the communication protocol together with its encoding and security aspects. The supported protocols are HTTP [31], FTP [26], XML-REC [32] and SOAP [30]. But in our work we will use just two types of protocols - HTTP and SOAP.

2.3.9.1 HTTP

Basically, HTTP is the HyperText Transfer that is used to deliver virtually all files and other data resources on the World Wide Web. Usually HTTP takes place through TCP/IP sockets. The HTTP client comes equipped with a browser that sends requests to an HTTP server and elicits a response in return. HTTP servers by default listen on port 80, but they can use any port. The Universal Resource Locator (URL) is a standardized naming convention for identifying a Web document or file, in a sense the address of a link.

The result is called the Web because it is made up of many sites, all linked together, with users travelling from one site to another by clicking a computer's pointing device on a hyperlink.

Generally, practical information systems require more functionality than simple retrieval, including search, front-end update and annotation. HTTP allows an open-ended set of methods to be used to indicate the purpose of a request. It builds on the discipline of reference provided by the Uniform Resource Identifier (URI) [15] as a location (URL) [17] or Uniform Resource Names (URN) [28], for indicating the resource on which a method is to be applied. Internet Mail [20] and the Multipurpose Internet Mail Extensions (MIME) [18] are designed to be compatible with older Internet mail standards. HTTP is also used as a generic protocol for communication between user agents and proxies/gateways to other Internet protocols, such as SMTP [24], NNTP [23], Gopher [14], WSDL [58] and WAIS [21], allowing basic hypermedia access to resources available from diverse applications and simplifying the implementation of user agents.

2.3.9.2 SOAP (Simple Object Access Protocol)

SOAP [30] is a lightweight protocol for exchange of information in a decentralized, distributed environment. Also, it is an XML-based protocol that consists of three main parts:

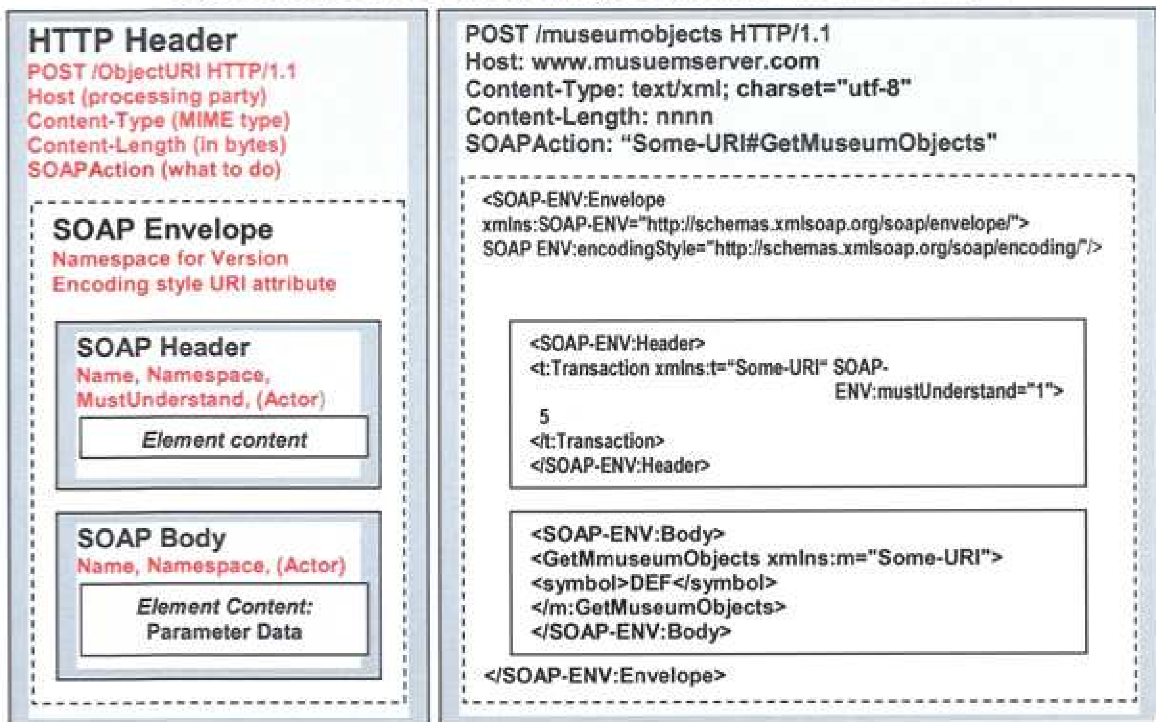
- *Envelope*, which defines a framework for describing what is in a message and how to process it.
- A set of *encoding rules* for expressing instances of application-defined data types.
- A convention for representing *remote procedure calls and responses*.

SOAP can be used possibly in combination with a variety of other protocols; however, the only bindings defined in this document describe how to use SOAP in combination with the HTTP protocol. In the following subsections more explanation on the main parts of SOAP is given:

2.3.9.2.1 Message Structure: The SOAP Envelope Contained in a HTTP message

A simple way to describe the SOAP [33, 34] message structure can be seen from the diagram shown in Figure 2.10. According to this diagram we can see on the right-hand side an example for a request message. In the message structure description we will give an example of values, which will be indicated between brackets.

Figure 2.10: Basic SOAP Message Structure with an Example



As is shown in Figure 2.10, **HTTP Header** (the first line) contains the send method (POST)[4], the Object URI (**MuseumObjects**) that is used by the HTTP server software to identify the target of the request and the HTTP1.1. The next line indicates the target host for the message (www.museumserver.com).

Then we see the Content-Type (must be text/xml) of the payload[5], i.e. the SOAP message, the character set (utf-8) used, and the Content-Length of the payload, in bytes (nnnn, for "some" number). Finally, and most important, there is the SoapAction, that tells the receiving application what to do, composed of a URI, a # sign, and an identifier that must equal the first element of the SOAP Body.

The XML part of the message is the **SOAP Envelope** (SOAP-ENV: Envelope). As we can see, the XML structure contains several specific tags and attributes, which are identified by the SOAP-ENV namespace prefix. This namespace is defined in the first attribute of the SOAP Envelope itself, which also functions as a kind of version indicator. Next is the encoding Style URI, which points to a link where the encoding rules are defined.

[4] Post is used for building applications, because it can transmit arbitrary data.

[5] It should be remarked that Soap's payload is part of the Envelope.

The optional element in the SOAP specification is **SOAP Header**, but this is very important in ebXML. Here, we see part of the *extensibility* design goal being accomplished: SOAP Headers offer a flexible way to extend a message with certain characteristics (e.g. for security). Again a namespace prefix (t) is assigned, since in general, all elements and attributes used in a SOAP message must be qualified by a namespace. In fact SOAP Headers are "optional", meaning that an application can ignore such a header if it is not recognized. However, an important attribute here is *mustUnderstand.* If this attribute has a value of "1", the header is said to be "mandatory", so the receiving application must return an error message if it does not recognize it. This way, it can be assured that any modification to the message semantics that is essential for correct processing will not go unnoticed, which would probably produce erroneous results.

The SOAP Header may contain any user-defined elements (in this case it just contains an identification value, "5"). As we mentioned above, SOAP 1.1 also introduced the concept of *intermediaries*, applications where a SOAP message "passes through" before going to its final destination. In the message, "actor" represents the attributes of these 'intermediaries'. This way, it can be indicated if certain information is intended for a particular intermediary.

Finally, the mandatory **SOAP Body** (SOAP-ENV: Body) contains the data that have to be passed to the service. The first element (GetMuseumObjects) is the name of the method being called containing the XML data as input for the method. It should be clear by now that the SOAP Request/Response model cleanly maps onto HTTP Requests and Responses. Again, there is a HTTP Header and a SOAP Envelope with a Header (in case there was a recognized Header in the request) and a mandatory Body, containing the processing result.

2.4 Conclusion

In this chapter of the literature survey a great deal was discovered and investigated. Many of the XML technologies that are relevant to this research were introduced and discussed. The chapter began with an introduction giving the background on XML and JAXP technologies and the motivations for its creation and its many benefits. Also we discussed the XML parsing technologies with examples and comparison of three types of parsing (SAX, DOM and JDOM). We also discussed the XML Stylesheet Language Transformation (XSLT) and the XSL Formatting Object Language (XSL-FO).

In general, the XSL stylesheet is used to convert the XML data into different formats such as XML, HTML or WML while the XSL-FO is used to describe the presentation of the output document in more detail. The XSL transformations are accomplished by applying a XSL stylesheet to a document. This stylesheet contains the conversion rules for the input document and this is used by the XSLT processor to generate the output document. Also we review several algorithms for transforming the XML document and discussed what they did. In addition we compared the DTD and XMLSchema and discussed the XML Query language and the storing of XML documents in a database. Finally, we discussed the transport protocols HTTP and SOAP.

Chapter 3:

Review of XML Algebra

In this chapter we provide a review of XML algebra models by giving an overview of the definition of algebra, relational algebra, operators, query algebra and semi-structured algebra. The remainder of this chapter is reserved for XML algebra models such as IBM [62], Niagara [67], YATL [65], Lore [78] and AT&T [77, 83].

3.1 Definition of Algebra

Algebra is a language which is used by mathematicians and scientists. An algebra or algebraic structure is a formal mathematical system consisting of a set of objects and operators on these objects. In algebra, building blocks are *terms, expressions, equations, identities, inequalities and operations.*

Given this, algebra is needed to help mathematicians to solve certain types of problems quicker and easier, by using letters and other kinds of symbols in place of numbers in equations and other arithmetic statements. In addition, algebra as a language is used instead of English because it is precise and concise. Also, it allows the finding of unknown numbers from information given. On the whole, algebra is a bridge between arithmetic and more broadly generalized mathematical situations.

With respect to computerized algebra, query algebra is part of relational algebra and is a closed and complete language in the spirit of relational algebra, supporting both value-oriented and object-oriented query processing in a single language. Furthermore, in the following sections we will explain relational algebra, the algebra operators and query algebra.

3.1.1 Relational Algebra

Relational algebra is a notation for representing the types of operations which can be performed on relational databases. In other words, a relational algebra is an algebraic language based on a small number of operators, which operate on relations (tables). It is the intermediate language used by a RDBMS. Queries are expressed by applying special operators to relations. Relational algebra is a procedural language used to describe how queries can be constructed. The expression below shows some relational algebra expressions and Table 4 simplifies the main relational algebra [80].

Expression::= relation | monadic-expression | dyadic-expression

Monadic-expression::= selection | projection | renaming

Selection::= $\sigma_{selection\text{-}condition}$ (relation-name)

Projection:: = π attribute - list (relation - name)

dyadic-expression:= U | − | × | ∩ | ÷ | $\oplus|_{join\text{-}condition}$

Table 4: Simplification of the main relational algebra

Relational Algebra		
Expression	::=	relation | monadic-expression | dyadic-expression
Monadic-expression	::=	selection | projection | renaming
Dyadic-expression	::=	expression dyadic-operation expression
Selection	::=	$\sigma_{selection\text{-}condition}$ (relation-name)
Projection	::=	π attribute - list (relation - name)
Renaming	::=	$\rho_{attribute\text{-}list}$ (relation-name)
Dyadic-operation	::=	U | − | × | ∩ | ÷ | $\oplus\|_{join\text{-}condition}$

These operations were originally defined for relations. Each of these creates a new relation from an existing relation or set of relations.

3.1.2 Operators

Algebra operators are symbols, called operators, which signify relationships between numbers or variables. Operators serve the same function as verbs in the English language. Some examples of operators in the language of algebra are:

+, -, /, *, =, ≠, >, <.

3.1.3 Query Algebras

Basically, most database query algebras are based on the algebra for the relational model proposed by Codd in 1970 and then formalized in [64, 76]. Tuples (or records) in the relational model represent real world entities. A tuple has a fixed integral number of named attributes or fields. These fields were restricted to contain only atomic (scalar, non-composite) values. This is known as the first normal form (1NF) restriction. A relation is a set of tuples with an identical layout. This layout is called the relation's schema. The relations are manipulated using relational algebra, which has the expressive power of relational calculus. The main standard relational algebra operators are Union, Difference, Cartesian Product, Projection and Selection.

3.2 Semi-Structured Algebras

Semi-structured data [86] is data in which the schema (information about the structure and types) is either completely absent or provides weak constraints on the data. Also semi-structured documents are used to implement any type of data sources known e.g., relational data, object-oriented data and others. XML documents can be implemented as semi-structured data sources in the Web context. In this section we will focus on work done on developing algebra for XML queries [93]. All the XML algebras we will discuss have operators similar to object algebra and extended-relational operators. Most of the operators have been modified so that they can be applied to the XML data model, which we will describe shortly.

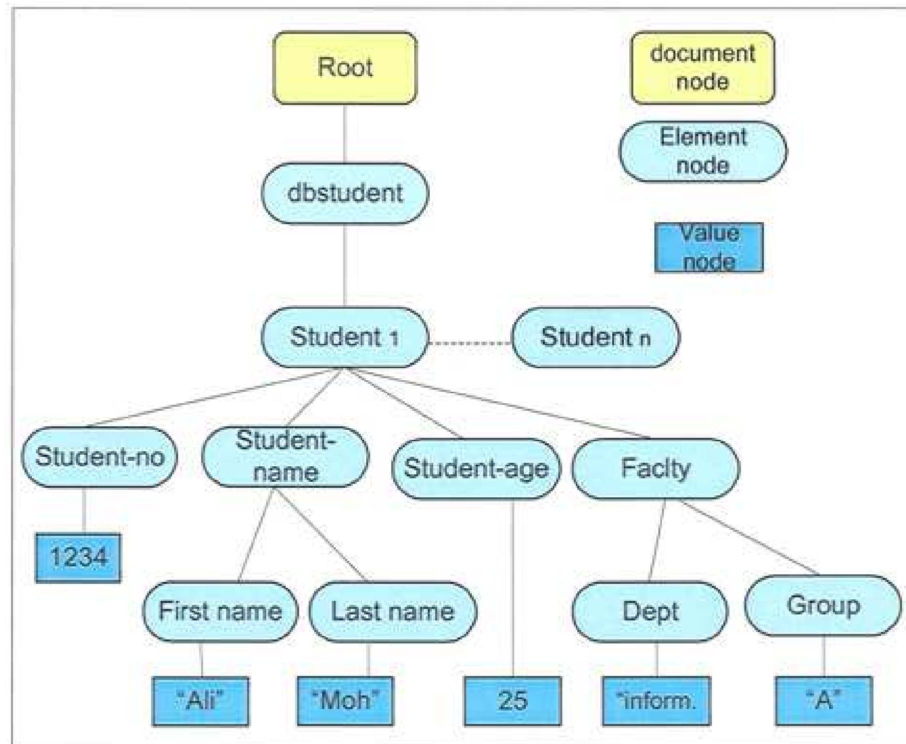
In September 1999, in order to establish standards for XML databases [63], the W3C included standards for an XML data model, query language algebra with more details on certain algebras and XQL [72, 75]. Several proposals for XML Algebras were submitted to the W3C, of which the AT&T algebra was selected and the working draft was published in February 2001 [62, 65, 69, 77, 84]. In this section we will discuss various proposals submitted to the W3C as well as recent work done on XML algebras after the working draft was published. The XML Query Working group published the XML Query Data Model draft in May 2000 [70]. The XML Query Data Model formally defines the information contained in the input to an XML Query Processor. In other words, an XML Query Processor evaluates a query on an instance of the XML Query Data Model. Figure 3.1 shows an XML document and associated DTD and Figure 3.2 simplifies the tree-based data model.

Figure 3.1: XML Document and its Associated DTD

```
<dbgroup>
 <student 1>
   <student-no>1234</student-no>
   <student-name> Moh</student-name>
   <student-age>25</student-age>
   <Faculty>
      <dept>informatics/dept>
      <group> A</group>
   </Faculty>
 </student>
        :

 <student n>
   <student-no>1237</student-no>
   <student-name>Ali</student-name>
   <student-age>30</student-age>
   <Faculty>
      <dept> engineering </dept>
      <group> B</group>
   </Faculty>
 </Student n>
</dbgroup>
```

```
<!DOCTYPE dbgroup [
<!ELEMENT dbgroup (student+)>
<!ELEMENT student(student-name, student- age, faculty)>
<!ELEMENT student-no CDATA #REQUIRED>
<!ELEMENT student-name (#PCDATA)>
<!ELEMENT student-age (#PCDATA)>
<!ELEMENT Faculty (dept )>
<!ELEMENT dept (#PCDATA)>
<!ELEMENT group(#PCDATA)>
]>
```

Figure 3.2: An XML Data Model



In the following sections we will discuss several algebra approaches. Firstly we discuss the IBM algebra, which was proposed in September 1999 by developers from IBM, Microsoft and Oracle [62]. The IBM algebra has some operators that are similar to the object algebra operators.

Secondly, we will discuss recent algebras proposed in 2001 by researchers from the University of Wisconsin, namely the Niagara algebra [67]. The Niagara algebra was developed after the authors unsuccessfully tried to implement some of the existing algebras including the IBM algebra. Thirdly, we will discuss other system-specific algebras such as YATL [65] and Lore [68]. The YATL algebra was proposed in May 2000 and was developed for the YAT XML-based integration system. Lore algebra is a DBMS designed specifically for managing semi-structured information that was originally proposed in 1997 by researchers from Stanford University.

Finally we describe the AT&T algebra [66], which is an updated version of the algebra originally proposed in June 2000 by researchers from AT&T. The W3C algebra [44] was proposed in September 1999.

3.2 XML Algebra Models

In this section we review the XML algebra models and concentrate on existing approaches to formulate an algebraic framework model. We first investigate the XML algebra models.

3.3.1 IBM algebra

The IBM algebra data model was presented by a group of researchers from IBM, Oracle and Microsoft in September 1999 [62]. This model provides an algebra that operates as a graph-based data model.

Basically, the IBM model is a logical model, which means it is independent from the underlying hardware and the physical storage synopsis and syntax. In addition to its standard operators, there are reshaping operators presented by this model. By using such operators, a new XML document can be created from a fragment of XML documents. Generally, the path navigation operator ϕ is expressed as follows.

ϕ [edge *type, name*] (*Vertex – expression)*

Edge type can be *element*, *attribute* or *reference*, while *vertex–expression* represents the collection of vertices in the data model graph from which edges with the given edge type and name should originate. As a result, the ϕ returns these sets of edges. The following examples show different expressions used by IBM algebra that represents various operations, such as the selection and join operators.

- **Selection operation**

  Basically, the selection operation returns a collection of vertices that satisfy some predicate. This is expressed as below:

  Syntax: *σ[condition (e)](e: expression)*

  Example: Find all students studying in the informatics department.

  σ[Value (x/student/faculty/dept) ='informatics'](x: child(/dbstudent))

  In this example, the selection operator (σ) returns a collection of vertices, which satisfies the predicate (dept = 'informatics') and the *child* is the property of an edge that returns the vertex referred to by the edge.

- **Join operation (⊗)**

  In general, Join is a query operation, which performs a selection operation that meets certain criteria from a collection of documents.

  Syntax: *(a: expression) ⊗[condition (a, b)](b: expression)*

Example: assume that the student-no is stored as an element in the first collection of documents and as an attribute in the second collection and spelt differently. Find all students in the first and second collection of documents that have the same value and type for the student-no.

In addition to these fundamental operations, other algebra operations are supported by the IBM algebra model such as the *sort* operation (Σ), which allows reordering of the sequence, or to order any unordered collection. Also, the *expose* operation is a special map operation, which exposes certain *edges* that originate at the set of vertices specified by the given expression. There are also other types of algebra operations such as *iteration, quantification, aggregation* that are used with the IBM algebra model.

Correspondingly, other algebraic standards comparison operators such as (<>, <, >, >=) can also be used to construct the *condition* section. In addition, Boolean operators (and, or, not) can be used to construct more complex conditions. Likewise, the navigation can be used along with *condition* section to specify the collections.

Despite the fact that its lack of optimisation rules out reducing the cost of query execution [62], the IBM algebra data model has a unique feature compared to many other logical data models in expressing its operators and queries in a clear and simple manner from a user perspective.

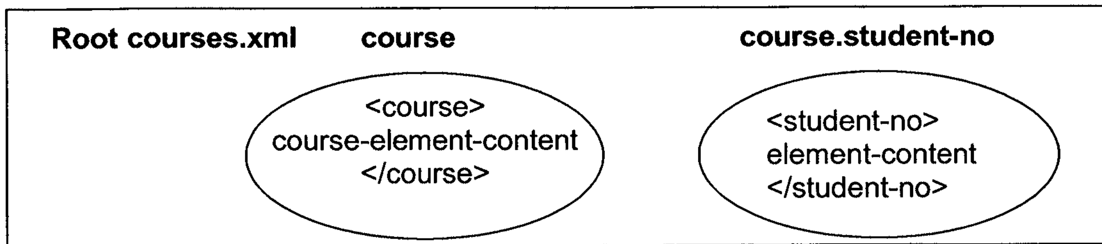
3.3.2 Niagara Algebra

Niagara algebra is similar to the IBM algebra data model [67]. It provides an algebra operator, which operates as a graph-based data model.

Basically, the developer of Niagara algebra in 2001 discusses some of the drawbacks of the IBM algebra model and attempts to implement and sort out these problems. These drawbacks were the main motive for the Niagara Algebra development team to create their model. The first drawback of the IBM algebra model is that the algebraic framework assumes that at the time of writing the query, the type is known for each vertex (attribute, element and reference). Secondly, the IBM algebra model defines a large collection of operators and clearly preserves all variable bindings appearing in a query, by creating multiple clauses for each query and assigning intermediate results to these bindings. As a result, implementing a query using the IBM model leads to a complex structure and, as we mentioned earlier, the deficiency of optimisation rules.

One of the innovative ideas proposed by its developers is that in Niagara algebra, instead of having the operators operate on a set of vertices, they operate on a set of bags of vertices. Each bag contains the vertex being operated upon as well as all the vertices already visited along the path to that vertex. XML data models have a collection of bags of vertices and vertices in a bag have no order. Figure 3.3 represents the XML data model based in Niagara algebra.

Figure 3.3: Set of Vertices



Niagara currently uses XML-QL as its query language. The backend query engine of Niagara uses its own Logical Algebra. XML-QL queries are parsed and converted to this Logical Algebra, which is then fed into the query execution engine for processing. Table 5 simplifies the main operators used by this algebra:

Table 5: Niagara Algebra Operators

Operator	Symbol	Description
Select	σ	Selects tuples based on a certain condition.
Follow	ϕ	The parent element to its child.
Vertex	ν	Creates new vertex.
Expose	ε	Projects the required elements.
Join	$\bowtie$	Joins the two collections based on a predicate.
Group	γ	Grouping of elements based on their values.
Rename	ρ	Entry point annotation of the elements of bag.
Source	S	Source operator used to specify the XML document.

In addition to these fundamental operations, other algebra operations are supported by Niagara algebra model such as *union* ($\cup$), *intersection* ($\cap$), *difference (-)* and *cartesian product* (x).

To further explain the operators in Table 5, we will interpret the tree data model shown in Figure 3.4 from the bottom to the top with the main operators used by Niagara algebra as:

Source (Courses.xml, Students.xml)

Follow (Root courses, course) ==> course (2)

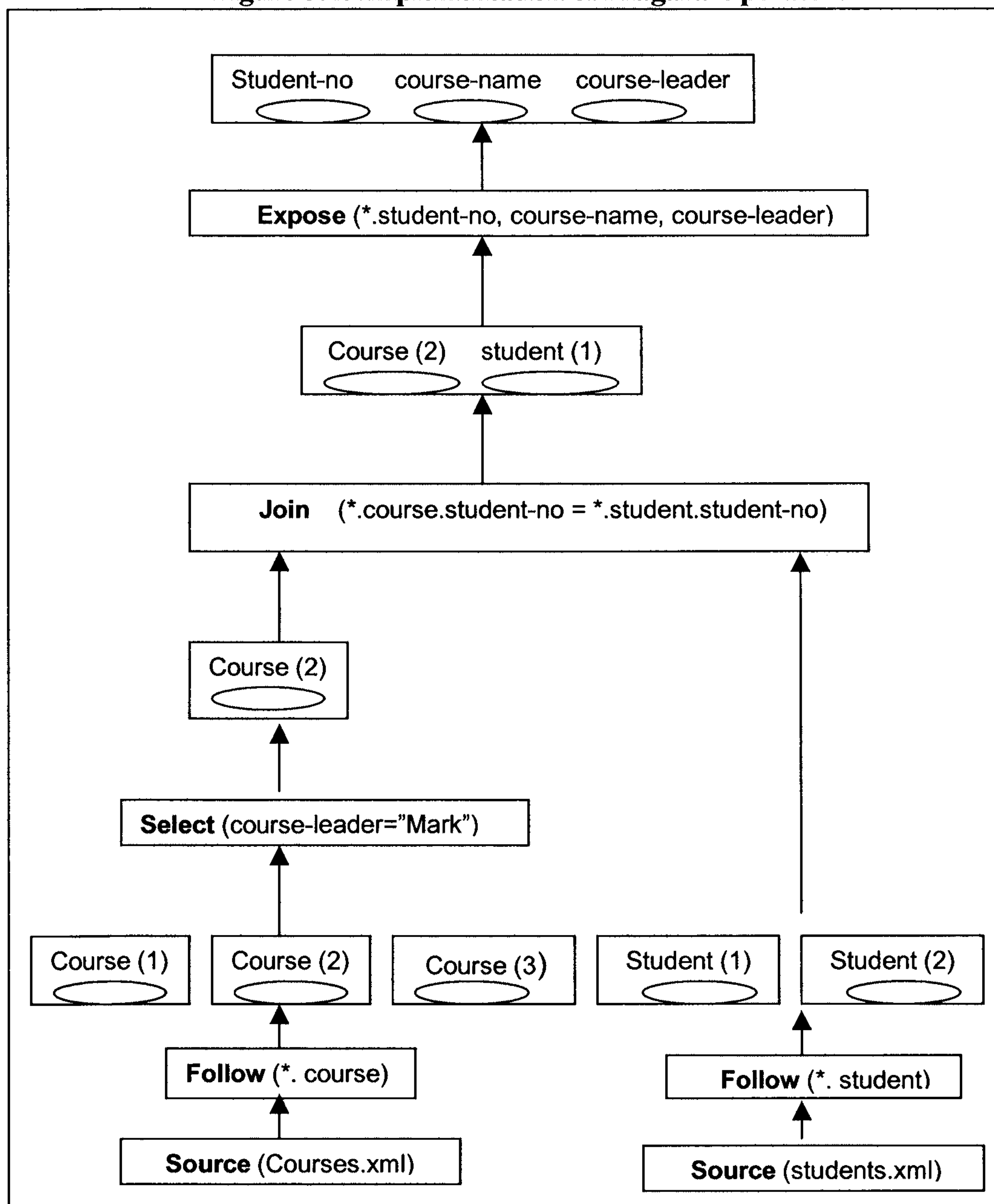
Follow (Root students, student) ==> student (1)

Select (courses.course. course-leader="Mark") ==> course-leader="Mark"

Join (courses.course.student-no= students.student.student-no)=>course (2), student(1)

Expose (student-no, course-name, course-leader) => course (2), student (1), "Mark"

Figure 3.4: Implementation of Niagara Operators



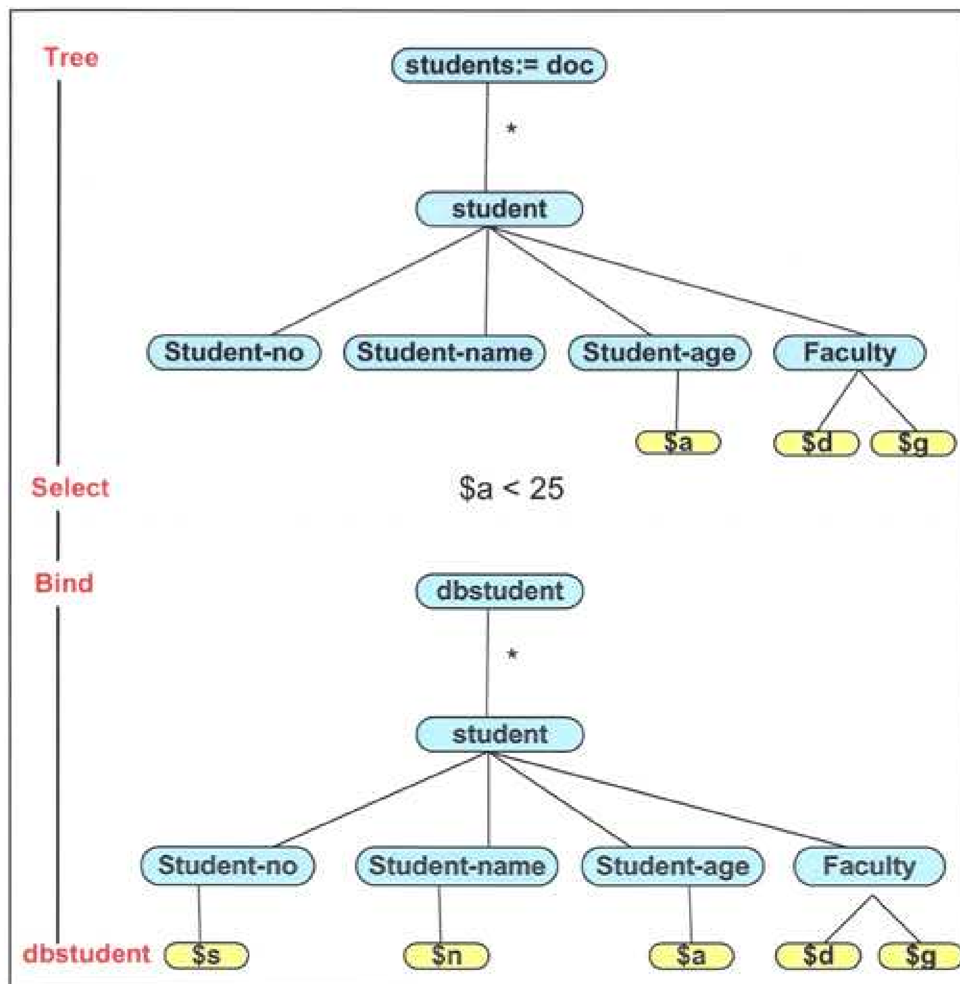
3.3.3 YATL Algebra

YATL (YAT Language) algebra is an advanced and modified version of YAT algebra; it was developed in 2000 [65]. Basically, the same relational algebra rules and operators are presented by the YATL algebra model but in addition, the YATL algebra has two distinctive features compared to other algebra models. The main feature is its ability to integrate data from different sources, such as semi-structured web repositories and object databases. To support integration, the researchers have provided two new operators, firstly *Bind* and secondly a *Tree* operator. The Bind operator is used to extract relevant information from different sources and produce a new structure called *Tab*, which is equivalent to a 1NF relation. Consequently, other standard operators such as *Join*, *Select* and *Project* can be applied on the *Tab* structure.

Overall, the *Bind* operation provides support for type filtering and horizontal and vertical navigation. Another distinctive feature proposed by the YATL authors is the *Tree* operation. By using such an operation, transforming a relational structure to a *Tree* structure is possible. To be more specific it is generating a new XML structure from a *Tab* structure. In fact, *Tree* is the sequence of *Group*, *Sort* and *Map* operations that are object algebra operators. Furthermore, YATL algebra is a logical model, which means it is independent from the underlying storage structure.

However, the main drawback of YATL algebra is that the researchers have developed it in the context of an integrated system using their own data model and type system, but have not provided a well-defined list of the operators and explicit rules of optimisation. Despite this it is a logical model, meaning it is independent from the underlying storage structure. The following examples in Figure 3.5 show how both YATL operators (*Bind* and *Tree*) can be used to transform a tree structure to a relational structure and vice versa. Finally, the inverse operation to *Bind* called *Tree* generates a new XML structure.

Figure 3.5: Query Evaluated by YATL Algebra



3.3.4 Lore Algebra

Lore (Lightweight Object Repository) [78,79] is a DBMS designed specifically for managing semi-structured information. The Lore data model is a very simple self-describing, nested object model called OEM (Object Exchange Model) [74]. As the OEM database is self-describing there is no notion of a fixed schema. This means the schema may change dynamically with no restrictions.

Basically, Lore algebra developers have focused their efforts on optimising the Lore queries processor from the cost point of view (cost based query), involving especially the efficient evaluation of path expressions. Each query is transformed into a logical query plan, using logical operators such as *select*, *project, discover, name,* chain, *glue*, etc. In addition, each logical query plan can give rise to more than one physical query plan and a cost-based approach is used to select the best physical plan. Functionally, each simple path expression in the query is represented as a d*iscover* node.

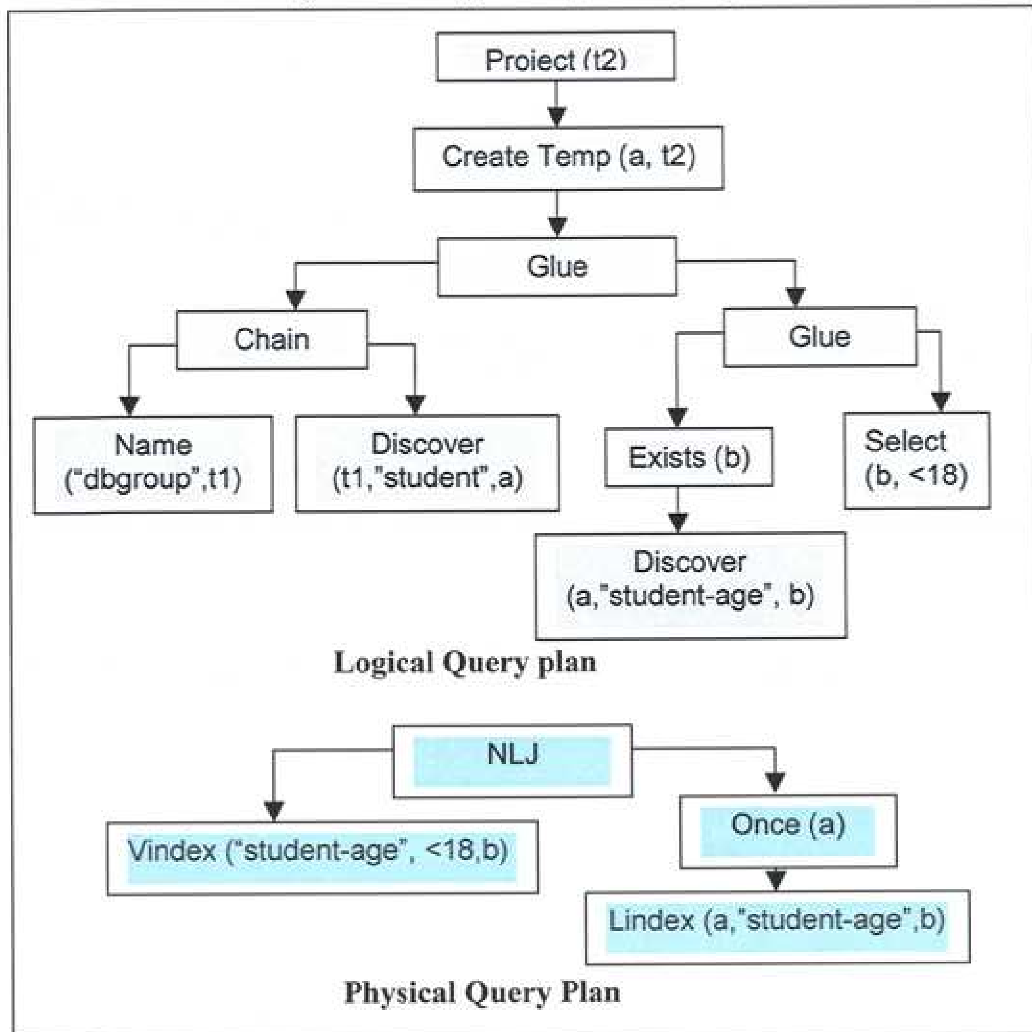
Usually, these path expressions are grouped together into a single path expression as a tree of d*iscover* nodes connected via c*hain* nodes.

Consequently, it is the responsibility of the *Chain* operator to optimise the entire path expression represented in its left and right sub (plans). Places where independent components meet are called *rotation points* because in the creation of physical query plans the order between independent components can be rotated to get vastly different physical query plans. These rotation points are represented as a *glue* node. The *CreateTemp* and *Project* nodes at the top of the plan are responsible for gathering the satisfying evaluations and returning the appropriate objects to the user. The following diagram in Figure 3.6 can simplify the query process in Lore DBMS [79].

Yet, the Lore system has a major downfall: it has some physical operators inherent in its design. These physical operators for indexing are used to represent very specific types for maintaining the Lore system. Basically, there are three index operators in this system, *Vindex, Lindex* and *Once* as shown in Figure 3.6. The *Once* operator allows the name to be passed to the parent only if it has not appeared earlier. Also, the Vindex operator uses the value index to match all objects, which satisfy the condition *"student-age< 18"*.

Furthermore, the Nested Loop Join (*NLJ*) operator is used to pass bound variables among these physical operators. Figure 3.6 shows for one example application a logical query plan and a physical query plan. More details on these operators are provided by McHugh and Widom [73].

Figure 3.6: Logical/Physical Query Plans



3.3.5 AT&T Algebra

The AT&T algebra data model [66,77,83] has been inspired by systems such as SQL, *OQL* and Nested Relational Algebra (NRA). The purpose of AT&T algebra is twofold. First, the algebra is used to give semantics for the query language, so the operations of AT&T algebra should be well defined. Second, the algebra is used to support query optimisation.

The researchers have also implemented types that can be used to detect certain kinds of errors at the compile time and to support query optimisation. In addition the authors have also implemented a type checker and an interpreter for the algebra in OCaml, which is a functional programming language [85]. The AT&T algebra operations are based on the iteration operation. The simple iteration operation is *for*, an expression that iterates over elements in a document given the path expression.

The selection operation is a *for* expression with a *where* predicates clause and the *join* operations are nested *for* expressions [83].

The symbol (/) gives the path expression. The *project* operation is also built from the *for* expression, the *nodes* function and the *match* expression. The following examples below show equivalent expressions.

```
Example 1:
object1/objInfor
Is equivalent to the expression:
for v in nodes(object1)  do
  match  v
    case a : objInfor [AnyComplexType]  do  a
    else  ( )
```

Above *AnyComplexType* is the type of the element. In this example, the *nodes ()* function returns a forest consisting of the content of the element *object1,* namely, a title element and three author elements. The *For* expression binds the variable **v** successively to each of these elements. Then the *match* expression selects a branch based on the type of **v**.

```
Example 2:

collection1/object/objInfor
Is equivalent to the expression:
    For b in nodes (collection1) do
       Match b
    case b: object [AnyComplexType]  do
       for d in nodes(b) do
          match d
            case a : objInfor[AnyCompltxType]  do  a
            else ()
       else ()
```

Above the *For* expression iterates over all book elements in *collection1* and the variable **b** to each such element. For each element bound to **b**, the inner expression returns all the author elements in **b**, and the resulting forests are concatenated together.

3.3.6 Tree Algebra for XML

TAX (Tree Algebra XML) [132] is for manipulating XML data, modelled as forests of labelled, ordered trees. TAX has been developed as a natural extension of relational algebra with a small set of operators. TAX is complete for relational algebra extended with aggregation, and can express most queries that can be formal in popular XML query languages. The TAX group introduces the notion of a *pattern tree*, which identifies the subset of nodes of interest in any tree in a collection of trees. The pattern is fixed for a given operation, and hence provides the needed standardization over a heterogeneous set. All algebraic operators manipulate nodes and attributes identified by means of a pattern tree (sub-tree). As such, they can be applied to any heterogeneous collection of trees and they show that most operators in relational algebra can be carried over into the tree domain. The TAX developers assume that each node has a virtual attribute called *pedigree* drawn from an ordered domain. Operators of the algebra can access node pedigrees much like other attributes for the purposes of manipulation and comparison. Pedigree plays a central role in grouping and sorting and elimination of duplicates.

All operators in TAX take collection of data trees as input, and produce a collection of data trees as output. The main algebra operators are selection, projection, product, grouping, aggregation, renaming, reordering and copy-and-paste.

However, the TAX approach has certain drawbacks, in particular for operations such as joins the pattern trees need to be explicitly defined to identify how matching across the two relations is to be done. The operations are therefore relatively procedural.

3.3.7 W3C XML Algebra

The XML Query Working Group of the W3C, established in September 1999, is working towards developing these standards for XML databases. This section briefly describes the Data Model proposed by the W3C and discusses the work of major research groups in developing a Query Language and Algebra for XML [44]. The W3C established the XML Query Working Group in September 1999. The mission of the XML Query working group is to provide flexible query facilities to extract data from real and virtual documents on the Web. The XML Query Working Group published its Requirements Document [69] in January 2000, followed by updates in February 2001 [44].

This document specifies goals, requirements and usage scenarios for the W3C XML Query Data Model, algebra and query language. The Working Group also published a Data Model draft in May 2000 [70] and updated it in 2001.

3.3.7.1 W3C Data Model

The XML Query Data Model formally defines the information contained in the input to an XML Query Processor. In other words, an XML Query processor evaluates a query as an instance of the XML Query Data Model. As the XML documents are tree-structured, the XML Query Data Model is defined using conventional tree terminology. The Data Model defines the structure of various types of tree nodes, functions to construct tree nodes, called *constructors*, and functions to access nodes. The W3C algebra operations are based on the iteration operation. The simple iteration operation is *for*, an expression that iterates over elements in a document given the path expression. In fact, the standard is largely based on the AT&T algebra with some additional features and revisions in the spirit of the Niagara algebra.

3.4 Comparative analysis of Different Algebraic models

Based on our review in this chapter, it can be seen from Table 6 that although all algebras differ from each other, some have common features, e.g. all algebra models studied support standard algebra operations. IBM algebra, Niagara and AT&T are standalone XML algebras. In addition, they have the ability to act as graph-based logical data models. These factors have been carefully taken into consideration by W3C in the proposal for the standard XML algebra [67, 69, 77].

Alongside these common characteristics, some of these models have unique features. For example, the Lore system has a powerful feature for manipulating dynamic data structures. Similarly, AT&T algebra has a distinctive ability for detecting errors at query compile time with its well-defined list of the operators and explicit optimisations. Also, YATL algebra has the ability to integrate data from different sources. To use the advantages of these features, the algebra model needs special operators to perform such operations.

These operators should be uniquely defined for their own data model. For instance, in the YATL system there are the *Bind* and *Tree* operators, used for structure transformation purposes [88]. Also, IBM algebra has its *Reshaping* operators, used to create new XML documents from a fragment of XML documents.

As can be seen so far, both YATL and Lore systems suffer from major critical drawbacks, where integration in both systems is exclusively dependent on their own data model. Therefore both systems have to be excluded from further consideration due to their lack of facilities for integration. Apart from the ability to achieve simplicity in its query structure, the IBM model has a major downfall: its deficiency in optimisation rules as mentioned before. On the other hand, both the Niagara and AT&T systems support standard algebraic operators. Also, they both have the ability to act as a standalone XML algebra system, allowing manipulation of algebraic expressions in a simple and powerful way.

Furthermore, Tree Algebra XML manipulates the XML data, modelled as forests of labelled ordered trees. TAX has developed as a natural extension of relational algebra with a small set of operators. All operators in TAX take collection of data trees as input, and produce a collection of data trees as output.

All the above considerations have already been accounted for by the team working on the standard XML algebra [77, 83]. In fact, the standard is largely based on the AT&T model with some additional features and revisions in the spirit of Niagara. So, the standard is a good starting point for us in an effort to develop domain-specific algebra and we will make use of it. However, it does not make sense to implement the full algebra defined by the standard for one single specific task or even for a class of similar tasks. Because of this we will develop a restricted version of the standard algebra instead, which suits representative classes of problems. Of course, such an approach will surely lead to a non-universal model, but at the same time, it will be feasible and will definitely produce a more effective solution for classes of problems. Table 6 provides a review summary of algebraic methods discussed so far.

Table 6: Comparison of Different Algebraic Models

Algebraic Model	IBM	YATL	Niagara	Lore	AT&T	TAX	W3C
Data Model	Standalone XML algebra	Logical Data Model, graph based	Standalone XML algebra	Integrated DBMS	Standalone XML algebra	Logical Data Model, tree Based	Standalone XML algebra
Approach	Represents the collection of vertices	Integrating data from different sources	Operates on a set of bags of vertices	Cost-based query optimization	Based on the iteration operation, NRA (Nested Relational Algebra)	Operates on a collection of trees	Based on the iteration operation
Distinctive Features	- Not system specific, XQuery support. - Provides an algebra operator that operates as graph-based data model.	- Ability to integrate data from different sources can be used efficiently in querying distributed data. - YTAL has Tree operations for transforming relation structure to Tree structure.	- Simple and powerful algebraic expressions , optimised rules	- Optimised from cost-based perspective, dynamic schema structure. - Each query is transformed into a logical query plan using logical operators such as *Select, Project, Discover*...etc	- Built in types for detecting errors at query compile time. - SQL & OQL and NRA support XQuery. - Based on the iteration operation.	- Provides an algebra operator that operates as collection of trees based data model. - The relation data model is the declarative expression in terms of algebraic expressions over collection of tuples	- NRA support the XQuery. - Common for a query language to exploit types with NRA
Drawbacks	- Deficiency of optimisation rules, complex query structure, data type should be known at query compile time	- The integration is exclusively based on YATL data model and type system, and does not provide a well-defined list of the operators and explicit optimisations.	- The algebraic framework assumes that at the time of writing the query, the type is known for each vertex (attribute, element and reference). -Implemen-ting a query using IBM model will lead to a complex structure.	- The physical operators are designed specifically for its own data model	- a complex query once generated, is difficult to optimize	- Some operations such as joins are relatively procedural - All the operator in this model operates on sub-tree	- Query plan based on this algebra, once generated, is difficult to optimize
Special Operators	Reshaping Operators	Bind, Tree operators	-	Vindex, Lindex, Once	-	Copy-and-paste, node deletion, node insertion	-
W3C	Proposed	-	Proposed	-	Proposed	-	Proposed

3.5 Conclusion

The aim of this review has been to compare several approaches to XML algebra, concentrating on existing ones and discussing the advantages and formulating an algebra framework model. Such a data model must have a well-defined standard SQL support. To put it more simply, XML algebra is the support for standard relational algebra operations and nested relational algebra.

Of course, W3C's recommendation for such a data model to be used as standard XML algebra is crucial in our case. Accordingly, the targeted framework will be able to deal with XML documents, firstly by performing XML document translation into relations and vice versa, then secondly being able to handle different related algebra tasks.

In addition, a review and investigation of XML algebra models gives us the choice of an existing solution for application to our context of either (1) Lore algebra that adopts an existing standard for building an original solution, (2) the standard XML algebra of W3C, or (3) developing a more targeted solution, that is a domain-specific algebra.

Furthermore, all the previous considerations in the review have already been described by a team working on the standard XML algebra. In fact, the standard is largely based on the AT&T model with some additional features and revisions in the spirit of Niagara. So, the standard is a satisfactory starting point for us in the effort to develop domain-specific algebra and was therefore adopted as the basis for our way forward. However, it does not make sense to implement the full algebra defined by the standard for one single specific task or even for a class of similar tasks. Because of this we develop a restricted version of the universal algebra suitable for a representative class of problems. Of course, such an approach may lead to a non-universal model, but at the same time, it is feasible and does produce a more effective solution for particular classes of problems. On this basis, we will start with the XML Schema of the data to develop a domain-specific XML algebra more suitable for data processing of the specific data and then we will use it for implementing the main offline components of the system for data processing. A more detailed explanation of our XML algebra and XQuery framework will be presented in Chapter 4.

Chapter 4:

Our XML Algebra and XQuery Framework

In this chapter we will introduce our algebraic and XQuery framework for expressing queries over XML data. We present assumptions of the framework and also describe the input and the output of the algebraic operators, and define the syntax of these operators and their semantics in terms of algorithms. Furthermore we define the algebra relational operators and their semantics in terms of algorithms. These algorithms have been faithfully implemented. Examples show that this framework is flexible to capture queries. As can be seen, the input and output of our algebra is a tree, in other words the input and output is an XML document and the XML document defined as a tree. We also present algorithms for many of our algebra operators. These algorithms show how the algebra operators *join*, *union*, *complement, project*, *select*, *expose* and *vertex* form an XML tree or an element and attribute of XML document. Detailed examples show how the algebraic operators work.

4.1 Description of our algebra

Our XML Algebra is a tree structure that consists of algebra operators [90,132]. Each node in the tree has only one parent node, but a parent node can have multiple child nodes. The XML Tree is interpreted top-down, so the root of the tree at the top of the XML Tree is when the final XML document output is produced, whereas the leaves of the tree correspond to the different data sources accessed, which in our system prototype are assumed to be relational [89]. There are two types of operators in our algebra. Firstly, the algebra operators are *join, union, complement, project, select, expose* and *vertex.* Every operator has an output of a tree, which makes it distinct from other algebra operators. Secondly, the algebra relational operators are *universal, subsuming, equivalence,* and s*imilarity.*

The originality in our algebra is that it operates on a new version of the semi-structured data model, employing the tree structure as data sources and targets. The difference between our algebra and the other XML algebras is that the other XML algebras have complex collections of bags, collection of vertices, collection of tuples and collection of trees as input and output.

Our algebra is simple in that it has one tree as input if it is unary, two trees as input if it is binary and in both cases one tree as output. Furthermore, our algebra framework can be used in integrated architectures for distributed information processing and its components will be XML schema-driven. Also, our algebra is a new domain for generic distributed problem solving in areas which will benefit from the simple approach.
Other XML algebra define a large collection of operators and explicitly preserve all variable bindings appearing in a query, by creating multiple clauses per query and assigning intermediate results to these bindings. This leads to complex structures when representing a query that makes the operators hard to deal with in an implementation. Our algebra has simple data structure; simple representation of the data and definitions for operator input and output can map all queries into a single output (one tree). Moreover, in our algebra we can expose nodes and leaves from the tree which can then be expressed as new trees. The targeted framework will be able to deal with XML documents, firstly by performing XML document translation into relations and vice versa, then secondly, being able to handle different related algebra tasks.

Our algebra employs the data sources as the natural structure of a tree. There are many technologies, softwares and parsing techniques that can be used for implementing the tree and can also transform the tree into different data structures and provide a way to create new data from existing data. A collection of tables to represent a tree model is less natural to the user and more difficult to implement. In the following sections we will explain in more detail our XML data model, all algebra operators and algebra relational operators.

4.2 Our XML Data Model

Figure 4.1a shows an XML document in our domain is representing collection of exhibition information where each set of objects such as object1, object3 has an associated set of information such as objectInfor1, objectinfor2, objectInfor3 and objectInfor4 in which there is a topmost, unique element, collection, known as the root of an XML document (ancestor) [88]. All elements are enclosed within the topmost element, collection1, known as the root of the XML document. Object1 and object3 sub-elements reside within the root node.

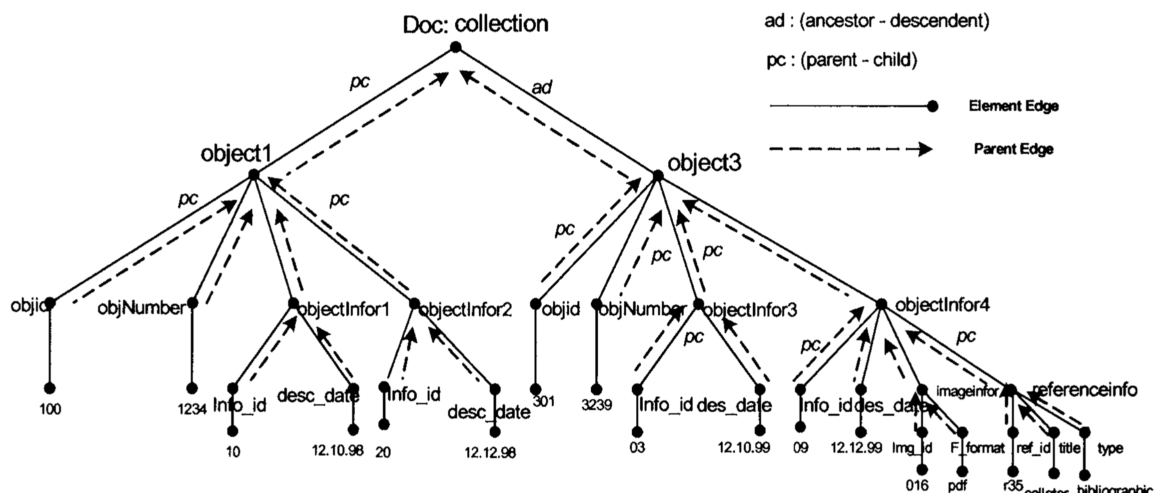
This nesting of sub-elements can go to an arbitrary level. Figure 4.1b depicts the corresponding tree model for the document in Figure 4.1a. Elements and attributes correspond to nodes in the XML tree.

Directed, named edges connect nodes, with the tag of the corresponding element or attribute name acting as the name of the edge. For each node of the tree, except the root node, there is a backward edge leading to the parent node. Note that a parent node appears only between those connected with child nodes or leaf nodes. A path consists of the sequence of nodes' names one needs to follow in order to arrive at a node from the root node.

Figure 4.1a: An XML document

```
<collection>
    <object1>
        <objectId>100</objectId>
        <objNumber>1234</objNumber>
        <objectInfor1>
            <info_id>10</info_id>
            <desc_date>12.10.98</desc_date>
        </objectInfor1>
        <objectInfor2>
            <info_id>20</info_id>
            <desc_date>12.12.98</desc_date>
        </objectInfor2>
    </object1>
    <object3>
        <objectId>301</objectId>
        <objNumber>3239</objNumber>
        <objectInfor3>
            <info_id>03</info_id>
            <desc_date>12.10.99</desc_date>
        </objectInfor3>
        <objectInfor4>
            <info_id>09</info_id>
            <desc_date>12.12.99</desc_date>
            <imageInfor>
                <img_id>016</img_id>
                <fileFormat>pdf<fileFormat>
            </imageInfor>
            <referenceinfor>
                <ref_id>r35</ref_id>
                <title>collector</title>
            </referenceinfor>
        </objectInfor4>
    </object3>
</collection>
```

Figure 4.1b: Tree data model for Figure 4.1a



4.2.1 Concepts of a tree model and data model definitions

This section introduces the terminology of the concepts of a tree data model such as root node, parent/child node, leaf, path, descendants and ancestor.

- Root (ancestor or parent): The top node of the tree is identified as the root node.
- Node (parent or child): An edge is a link from a parent node to a successor node, called a child node.
- Leaf (child): Child nodes, atomic values
- Path: A path from node V_1 to V_n sequence of nodes, where n =1...n, n = 20
- Descendants: Represent all nodes that are children of the current node or children of children of the current node and so on.
- Ancestor: Ancestors of nodes V are a parent, grandparent, etc, that is all nodes found on the path from node V to root node. In other words the ancestor represents all nodes that are parent of current node or parent of parent of current node and so on.

4.2.2 Definition of XML document as a Tree Structure

In the following we introduce the definition of XML document as a tree structure of the concepts.

- XML document → Tree
- Element → Root node, parent node, child node
- Leaf → child nodes, atomic values
- Attribute → atomic values

4.3 Test examples

We specify XML documents of the museum objects as examples, and we will apply our algebra operators and algebra relational operators on these examples of XML documents. An XML document consists of structures of nested elements starting with the root element. Each element has a tag associated with it.

In addition to the nested elements, an element can have attributes and atomic values or sub-elements. As can be seen from Examples 1, 2, 3 and 4, all are representing collection information such as collection1, collection2, collection3 and collection4 respectively. Each collection has a set of objects - object1, object2, object3 - and each object has a set of object information e.g. objectInfor1, objectinfor2, objectInfor3 and objectInfor4 and so on. Furthermore, there are unnamed elements in the XML documents, which can then be used to match other elements.

- **Example 1**

```
<collection1>
   <object1>
       <objectInfor1>
           <info_id>10</info_id>
           <desc_date>12.10.98</desc_date>
       </objectInfor1>
       <objectInfor2>
           <info_id>20</info_id>
           <desc_date>12.12.98</desc_date>
       </objectInfor2>
   </object1>
   <object3>
       <objectInfor3>
           <info_id>03</info_id>
           <desc_date>12.10.99</desc_date>
       </objectInfor3>
       <objectInfor4>
           <imageInfor>
               <img_id>016</img_id>
               <fileFormat>pdf</fileFormat>
           </imageInfor>
           <referenceinfor>
               <ref_id>r35</ref_id>
               <type>bibliographic</type>
           </referenceinfor>
       </objectInfor4>
   </object3>
</collection1>
```

- **Example 2**

```
<collection2>
    <object1/>
    <object3>
        <objectInfor3>
            <imageInfor>
                <img_id>016</img_id>
                <fileFormat>pdf</fileFormat>
            </imageInfor>
            <referenceinfor>
                <ref_id>30</ref_id>
                <title>tit23</title>
            </referenceinfor>
        </objectInfor3>
        <objectInfor4>
            <info_id>31</info_id>
            <desc_date>12.12.99</desc_date>
        </objectInfor4>
    </object3>
</collection2>
```

- **Example 3**

```
<collection3>
    <object1>
        <objectInfor1>
            <info_id>10</info_id>
            <desc_date>12.10.98 </desc_date>
        </objectInfor1>
        <objectInfor2>
            <info_id>20</info_id>
            <desc_date>12.12.98 </desc_date>
        </objectInfor2>
    </object1>
    <object3/>
</collection3>
```

- **Example4**

```
<collection4>
  <>
    <objectInfor1>
        <info_id>10</info_id>
        <desc_date>12.10.98 </desc_date>
    </objectInfor1>
    <objectInfor2>
        <info_id>20</info_id>
        <desc_date>12.12.98 </desc_date>
    </objectInfor2>
  </>
  <object3/>
</collection4>
```

4.4 Our algebraic operators and algebraic relational operators

Basically the algebra relational operators are *universal, subsuming, equivalence,* and *similarity*. The algebra operators are *project, select, join, complement, union, expose* and *vertex*. In the following sections we will give examples and explain in more detail all the algebra relational operators and the algebra operators.

The general approach to formalisms in this area is to emphasise the nodes in the tree for reference purposes as it is invariably the nodes upon which searches are made. It is, however, important to remember that the nodes are connected by edges for a full definition. We follow the approach of TAX [132] in defining a tree (T) as a collection of nodes or vertices (V) and edges (E):

T = (V,E)

Application of a formula F to a tree T returns another tree P as follows:

P = (T,F)

with the order of the nodes in T preserved in P.

In the calculus expressions below the formula F is represented by the predicate and the output tree by the target. A number of source trees are specified in the predicate.
Galanis [67] and Beech [62] adopt a similar approach.

4.4.1 Algebraic relational operators

This section introduces and elaborates four types of algebra relational operators namely *Universal*, *Similarity*, *Equivalence* and *Subsumption*.
This section introduces the definition of all the algebraic relational operators in terms of set theory. More details on the algebraic relational operators are found in the following sections.

- **Definition of Algebraic relational operators**

The algebraic relational operators can be classified as unary (one operand) or binary (two operands).

$T_2 \leftarrow Op\ T_1$

where T_2 and T_1 are two trees, Op is the relational operator.
n is parent node, child node, leaf node with atomic value.

Relational	The Definition of the Algebraic Relational Operators
Universal U Unary	A universal relational may be defined as: $U\ T_n \mid \forall n \in U$ The object holds all the nodes and edges in the museum system.
Similarity ~ Binary	$T_1 \sim T_2 \mid \forall n\ T_2.n \sim T_1.n$ The objects compared are two trees, holding similar data and structure. The essential structure of the two trees T_1 and T_2 is similar.
Equivalence ≈ Binary	$T_1 \approx T_2 \mid \forall n\ T_2.n = T_1.n$ The objects compared are two trees, holding the same data and structure. The two trees T_1 and T_2 are equivalent as they have the same edges and nodes.
Subsumption ⊂ Binary	$T_2 \subset T_1 \mid T_2.n \in T_1.n$ The objects compared are two trees in which we identify that the T_2 is a subsumption of T_1. The T_2 is a subsumption of T_1 as every node n in T_2 is also a node in T_1.

4.4.1.1 Universal element relational

The universal relational operator (∪) is unary containing all the information of the museum objects system (see chapter 6 for more details of this example application). Let ∪ = {Staff, Objects, Collection, Curator, Exhibition, Collector, ObjectInformation, Image, Documentation, Acquisition, Location, Reference, Collection, Visiting Groups, Display, Visitors, External, Institution, International, Private, State, Archive, Web Visitors, Virtual Exhibition, Address} be the universal relational of museum objects. The syntactical meaning of ∪ can include the following (with museum objects capitalised): Staff that are responsible for publishing the museum Objects and Collection (set of museum Objects). Next, the Curator is the person who manages the Exhibition. Then, the Collector is the person who collects some information related to the museum's objects. The ObjectInformation relating to museum objects is classified into six categories: Image, Documentation, Acquisition,

Location, Reference and Collection. The Visiting Groups are those who visit the Public Display (kinds of exhibition) and also the public can be visited by individuals (Visitors). The External person is a member of the board or staff and the board belongs to an Institution; the institution could be public, international, private or state. In addition, the museum archive will be considered as a type of exhibition. Finally, the proposed system will be visited via the web by different kinds of visitors (Web Visitors).

Those visitors will visit the virtual exhibition, which is a kind of exhibition. An address might be needed for the institutions and persons. The following example shows a fragment of museum objects as a universal relational.

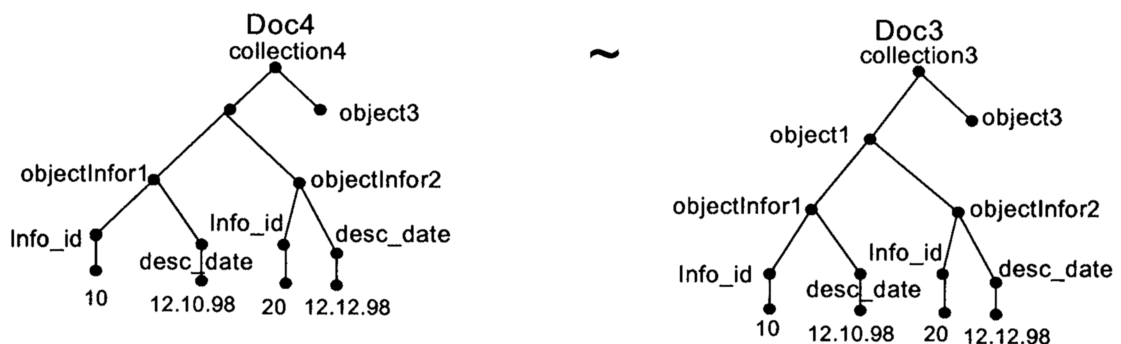
```
<museumObjects>
  <collection>
      <collection_id>20</collection_id>
      <num>01</num>
      <name> Alex Mark </name>
      <title>collector</title>
      <publishing_date>08.01.03</publishing_date>
      <remote>yes</remote>
      <features>exhibition</features>
      <purpose>research</purpose>
      <description>first exhibition</description>
      <url>http//:www.virtualexh.com</url>
      <exhibition_id>100</exhibition_id>
      <institution_id>200</institution_id>
      <staff_id>300</staff_id>
  </collection>
  <object>
      <object_id>200</object_id>
      <number>1234</number>
      <name>altarpiece</name>
      <title>lasmeninas</title>
      <registration_date>03.12.02</registration_date>
      <description>first exhibition</description>
      <staff_id>300</staff_id>
      <institution_id>200</institution_id>
  </object>
  <exhibition>
        <exhibition_id>10</exhibition_id>
        <title>private exhibition </title>
        <opening_date>10.01.03</opening_date>
        <description>educational</description>
        <opened>yes</opened>
        <curator_id> </curator_id>
        </institution_id>102</institution_id>
  </exhibition>
        .
</museumObjects>
```

4.4.1.2 Similarity Relation

A similarity relation (~) means that one tree is similar to another - for example the Doc4 tree is similar to the Doc3 tree in Figure 4.2. The node structure of the Doc4 tree is the same as that in the Doc3 tree. In other words the similarity relation means that the major structure of the two trees are similar. Also the similarity relation is binary. Figure 4.2 shows the two Doc trees and two XML documents are similar, as the nodes present in the Doc4 tree are similar to the nodes in Doc3 tree. Also, the Doc4 tree contains an unnamed node to match the object1 parent node in Doc3 tree and the object3 parent node is similar in both Doc trees. All child nodes and the leaf nodes in Doc4 tree and Doc3 tree are similar such as objectInfor1 and objectInfor2 child nodes in Doc4 tree are similar to the objectInfor1 and objectInfor2 child nodes in Doc3 tree. Furthermore the child nodes info_id and desc_date in Doc4 tree are similar to those child nodes info_id and desc_date in Doc3 tree. As the nodes present in Doc4 tree are similar to those in the Doc3 tree then we can thus identify that the Doc4 tree is similar to the Doc3 tree.

In general we can see the relation between any two XML documents or any two XML tree is one of similarity if the following holds: the root node in Docn tree and the root node in Docm tree are similar and the child nodes in both Docn tree and child node in Docm tree are similar. In addition if we find that any parent node or child node in Docn tree or Docm tree is an unnamed node, then this node matches the corresponding named node at the same level and position as depicted in Figure 4.2. As a result, we can conclude that the relation is similar.

Figure 4.2: Similarity relational



```
<collection4>
   < >
     <objectInfor1>
        <info_id>10</info_id>
        <desc_date>12.10.98 </desc_date>
     </objectInfor1>
     <objectInfor2>
        <info_id>20</info_id>
        <desc_date>12.12.98 </desc_date>
     </objectInfor2>
   </ >
   <object3>
   </object3>
</collection4>
```

```
<collection3>
   <object1>
     <objectInfor1>
        <info_id>10</info_id>
        <desc_date>12.10.98 </desc_date>
     </objectInfor1>
     <objectInfor2>
        <info_id>20</info_id>
        <desc_date>12.12.98 </desc_date>
     </objectInfor2>
   </object1>
   <object3>
   </object3>
</collection3>
```

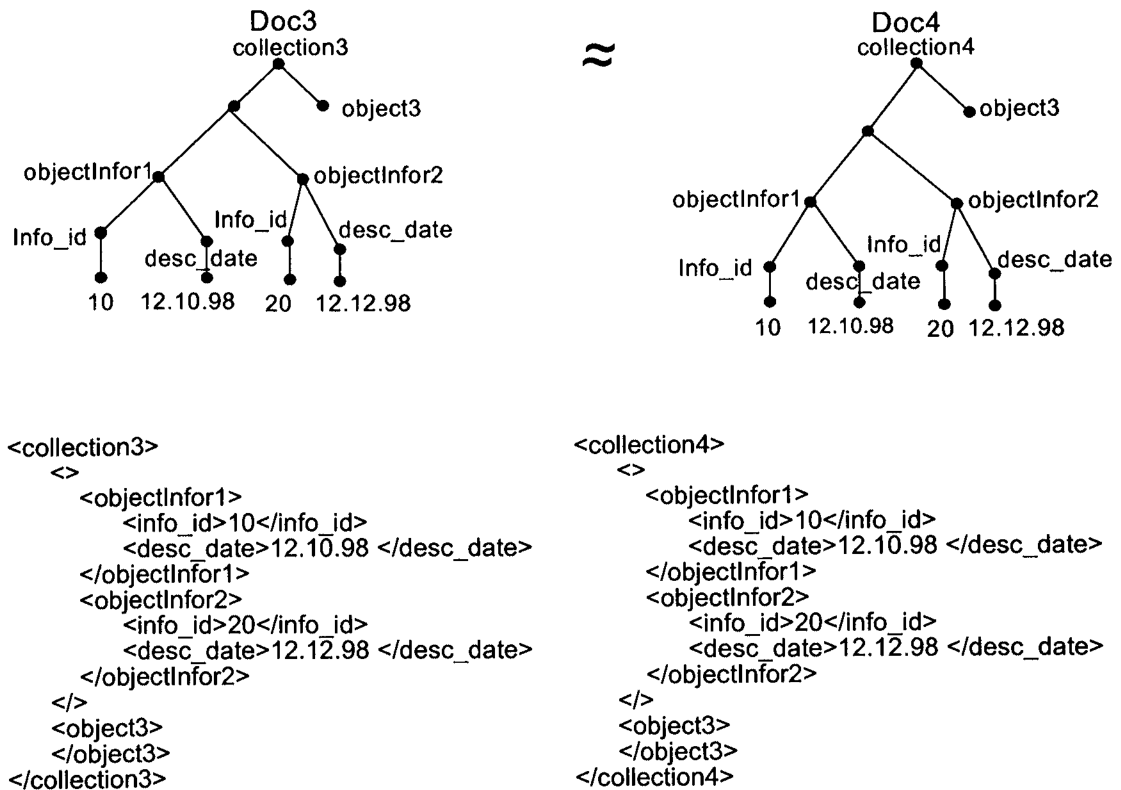
4.4.1.3 Equivalence Relation

Equivalence relation (≈) means that two trees or two XML documents are indistinguishable, as in Figure 4.3, where the nodes in Doc3 tree are equivalent to those in Doc4 tree. As the nodes present in the Doc3 tree are equivalent to those present in Doc4 tree then we can say that Doc3 tree is equivalent to the Doc4 tree. In other words the node structures of both trees are equivalent and the identities on a node-by-node basis are also equivalent. Also the equivalence is a binary relation.

As an example, Figure 4.3 shows the relation between the Doc3 tree and Doc4 tree as an equivalence relation. The nodes present in the Doc3 tree are equivalent to the nodes present in Doc4 tree, such as the collection4 root node in Doc4 tree is equivalent to the colection3 root node in Doc3 tree. The Doc3 tree contains an unnamed node which is equivalent to the unnamed node in the Doc4 tree. The child nodes objectInfor1 and objectInfor2 in both the Doc trees are equivalent. Also, the child nodes info_id and desc_date in Doc3 tree are equivalent to the child nodes info_id and desc_date in Doc4 tree.

In general we can state that the relation is one of equivalence between any two XML documents or any two XML trees if the following holds: the root node in Doc_n tree and the root node in Doc_m tree are equivalent, the parent node in Doc_n tree and that in Doc_m tree are equivalent and the child nodes in Doc_n tree and in Doc_m tree are equivalent. In addition if the parent node or child node in both Doc trees are unnamed, then the unnamed node matches the corresponding node at the same level and in the same position.

Figure 4.3: Equivalence relational



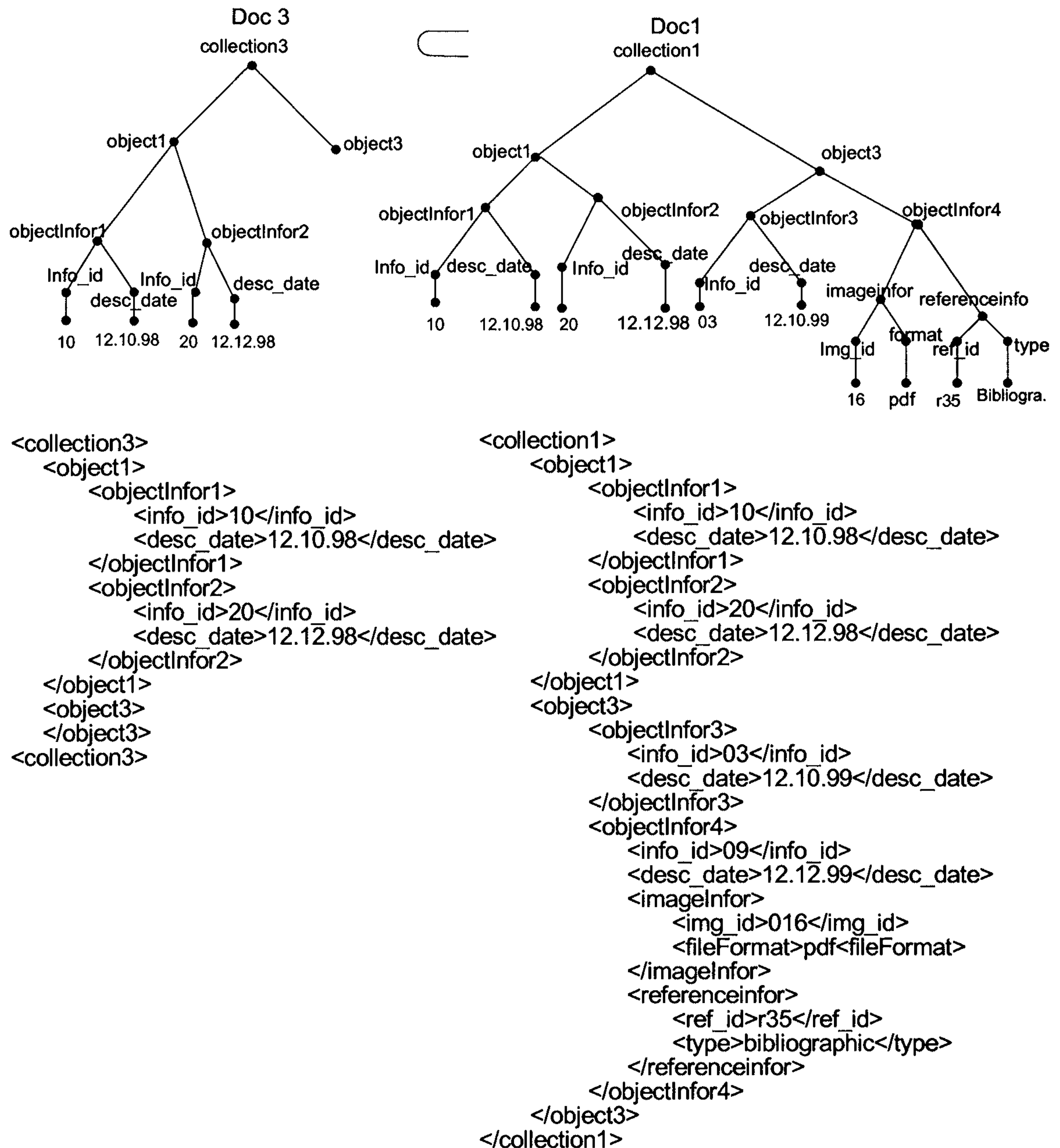
4.4.1.4 Subsumption Relation

A subsumption relation ($\subset$) means that one tree is a subset of the other, for example the Doc3 tree is a subset of the Doc1 tree. If the nodes and structures present in Doc3 tree are a subset of those present in Doc1 tree, then we can say the Doc3 tree is a subsumption of the Doc1 tree.

As an example, Figure 4.4 shows the relation between the XML documents collection3, collection1 and Doc3 tree, with Doc1 tree as a subsumption relation, as the elements in the collection3 XML document are a subset of the elements existing in the collection1 XML document and also the nodes present in Doc3 tree are a subset of the nodes in Doc1 tree.

In general we can conclude that the Doc_n tree is a subsumption or equal to Doc_m tree if the following holds: the root node in Doc_n tree exists in Doc_m tree; the parent node in the Doc_n tree exists in the Doc_m tree and the child node Doc_n tree exists in the Doc_m tree. In other words the Doc_n tree is part of the Doc_m tree.

Figure 4.4: Subsumption relational



4.4.2 Algebraic operators

All operators [80] in our tree algebra take one or more trees as input and produce one tree of data as output. This section introduces the definition of all the algebraic operators in terms of set theory. More details on the algebraic operators are found in the following sections.

❑ Definition of Algebraic operators in Terms of Set Theory

The algebraic operators can be classified as unary (one operand) or binary (two operands).

- **Unary Operators**

The format of a unary operator, with one tree as input and another as output, is:

$T_2 \leftarrow Op\ Y\ T_1$

where T_2 is the new tree,
T_1 is the old tree,
Op is the relational operator,
Y is a parameter, which may be a condition (C) or a tree expression (T_e).
C is in the form of a predicate,
T_e is in the form $P_1, P_2\ldots$
P_i (i>=1) is a full path from the root to a node n_j
The whole subtree with n_j as root is addressed by each tree expression.

Unary operators	The Definition of the Unary Algebra Operators
Project π	Produce a new tree T_2 in which $T_2 \subseteq T_1$ $T_2 \leftarrow \pi\ T_e\ T_1$ $T_2 \cong T_1 \mid n \in T_e$ The object returned is a tree with connections between nodes n in T_2 preserved as in T_1. The Projection operator was first defined for a relational model by Codd in 1972 [76] "The introduction of the relational algebra" and also "Relational Completeness of Database Sublanguages", and the proof of its equivalence to tuple oriented calculus was by Codd in 1970 [64]. The project operator in our algebra is defined in the same way as the project operator for the W3C algebra [77].

Select σ	$T_2 \leftarrow \sigma\, C\, T_1$ $T_2 \cong T_1 \mid n \in T_1 \wedge n \in C$ The object returned is a tree satisfying condition C with connections between nodes n in T_2 preserved as in T_1. The selection operator in our algebra is defined in the same way as the select operator for the W3C algebra [77].
Expose ε	$T_2 \leftarrow \varepsilon\, n\, p\, T_1$ $T_2 \cong T_1 \mid p \in T_1 \wedge n \in p$ The object found is the leaf node n in the path p. The tree returned is the leaf node n, its parent node and the link between them. The expose operators in our algebra are defined as the expose operators by Wan Liu & Binton Kane and are equivalent to that in their "Advanced database". Also, the expose operator is defined by Galanis & et al., the research group from the University of Wisconsin, and is equivalent to that in their "An algebraic framework for XML query evaluation" [67]. Moreover, the expose operator as defined by Beech et al. [62] and is equivalent to that in their "A formal data model and algebra for XML".
Vertex ν	$T_2 \leftarrow \nu\, p\, T_1$ $T_2 \cong T_1 \mid p \in T_1$ The object returned is a tree containing the node n specified by path p and all its successors in T_1. Connections between the nodes n and its successors in T_1 are preserved in T_2. The Vertex operators in our algebra are defined as the vertex operators by Galanis & et al [67] and is equivalent to that in their "An algebraic framework for XML query evaluation" and also as defined by Beech et al. and equivalent to that in their "A formal data model and algebra for XML" [62].

- **Binary Operators**

The format of a binary operator, with two trees as input and one tree as output, is:

$T_p \leftarrow T_m$ Op [Y] T_n

where T_p is the new tree,

T_m and T_n are the input (old) trees,

Op is the relational operator,

Y is an optional parameter, a condition (C) in the form of a predicate,

n is parent node, child node, leaf node with atomic value.

Binary operators	The definition of the Binary algebra operators
Join $\oplus$	$T_p \leftarrow T_m \oplus C\ T_n$ Consider a natural join of T_m and T_n with node n_a in common. Then $T_p \cong T_m \times T_n \mid C$ where C is $T_m.n_a = T_n.n_a$ The predicate C is applied to the product of T_m and T_n. The object returned is a tree containing nodes n present in T_m and T_n, linked where $T_m.n_a = T_n.n_a$ and with connections between edges and nodes n in T_p preserved as in T_m and T_n. Natural join is commutative $(T_m \oplus T_n) \cong (T_n \oplus T_m)$ Natural join is associative operator $(T_m \oplus T_n) \oplus T_q \cong T_m \oplus (T_n \oplus T_q)$. Natural join was first defined for databases by Codd [76] and is equivalent to that in his "Relational Completeness of Database Sublanguages". The natural join operator in our algebra is defined in the same way as the join operator for the W3C algebra [77].

Union $\cup$	$T_p \leftarrow T_m \cup T_n$ $T_p \cong T_m \cup T_n \mid n \in T_m \vee n \in T_n$ The object returned is a tree containing nodes n present in T_m or T_n with connections between nodes n in T_p preserved as in T_m or T_n. Duplicate nodes are removed as the output in a set of nodes. union is commutative operator $(T_m \cup T_n) \cong (T_n \cup T_m)$ union is an associative operator $(T_m \cup T_n) \cup T_q \cong T_m \cup (T_n \cup T_q)$ The union is disjoint: duplicates are purged not renamed. The union operator in our algebra is defined in the same way as the union operator for the W3C algebra [77].
Complement -	$T_p \leftarrow T_m - T_n$ $T_p \cong T_m - T_n \mid n \in T_m \wedge n \notin T_n$ The object returned is a tree containing nodes n in T_m that are not present in T_n with connections between nodes n in T_p preserved as in T_m. Complement is not commutative $(T_m - T_n) \neq (T_n - T_m)$ Complement is not associative $(T_m - T_n) - T_q \neq T_m - (T_n - T_q)$. A complement operator was first defined for databases by Codd [76] and is equivalent to that in his “Relational Completeness of Database Sublanguages”. The complement operator in our algebra is defined in the same way as the complement operator by Codd [76].

4.4.2.1 Join operator

A join (⊕) is a binary operator, which takes two trees as input, and combines them into one tree of data as output. This combination is made whenever a certain expression holds true, that is when the two tree collections are joined on a predicate. The predicate can be as generic as the conditions accepted by the selection operator. The query operation selects documents that meet stated criteria from a collection of documents. It may also extract components from selected documents and construct new documents from these components. In some cases, we may want to start with more than one source collection and perform the initial selection on documents from those collections that are related by some condition. Also the join operator has the property of being commutative.

More precisely Doc1 tree ⊕ Doc2 tree = Doc2 tree ⊕ Doc1 tree. Furthermore, it is an associative operator, which means

(Doc1 tree ⊕ Doc2 tree) ⊕ Doc3 tree = Doc2 tree ⊕ (Doc2 tree ⊕ Doc1 tree).

The algorithm is shown in Figure 4.5 for joining any XML documents or any two Doc trees in general.

Figure 4.5: An algorithm for Joining any two DOC trees or two XML documents

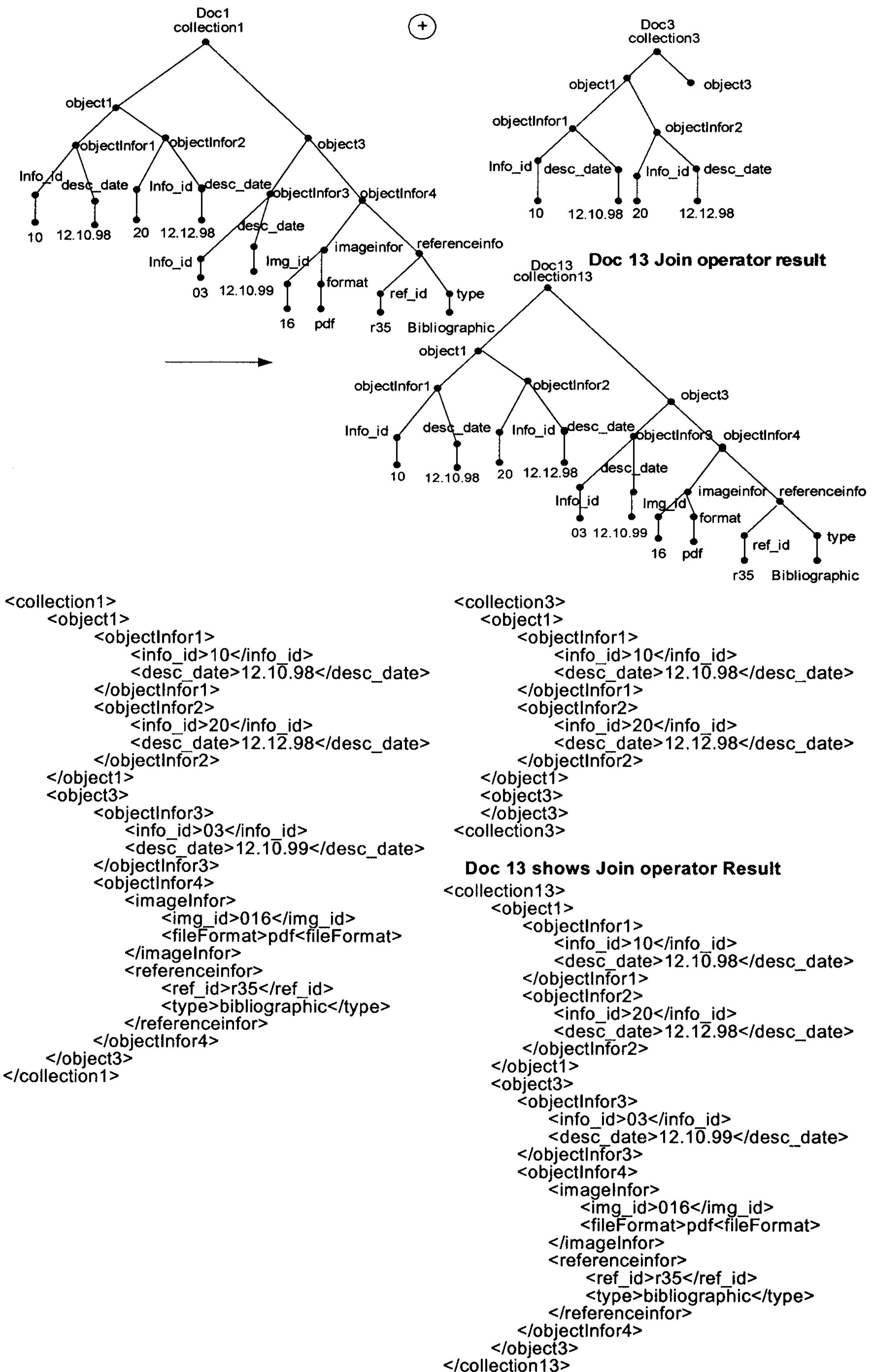
// Input two XML document or two DOC tree (DOC_n, DOC_m)
// Output DOC_{nm} Tree = (DOC_n Tree ⊕ DOC_m Tree)

1. start from the root node DOC_n Tree and root node DOC_m Tree
2. if the root node has a parent/child node
   - 2.1 perform depth-first algorithm
   - 2.2 if the parent node in DOC_n Tree has a child node and parent node in DOC_m Tree has child node
     - 2.2.1 while child node in DOC_n Tree agrees with the child node in DOC_m Tree repeat:
       - 2.2.1.1 join such a child node (concatenation without repetition)
       - 2.2.1.2 set the output in DOC_{nm} Tree
       - 2.2.1.3 if the child node in DOC_n Tree has a leaf node, agree with leaf node in DOC_m Tree
         - 2.2.1.3.1 join such a leaf node (concatenation without repetition)
         - 2.2.1.3.2 set a joined leaf node to the output in DOC_{nm} Tree
       - 2.2.1.4 join such a leaf node in DOC_n Tree to the leaf in DOC_m Tree
     - 2.2.2 join such leaf node in DOC_n Tree to the leaf node DOC_m Tree
   - 2.3 set the parent node into DOC_{nm} Tree output and terminate
3. set the root node to DOC_{nm} Tree and terminate
4. terminate

Figure 4.6 is an example of applying the algorithms to join any two XML documents or two XML trees. The root node collection1 in Doc1 tree data model is concatenated with the root node collection3 of the Doc3 tree data model. The parent node object1 of Doc1 tree data model is concatenated with the parent node object1 of the tree Doc3 tree data model. Furthermore, the parent node object3 of the Doc1 tree data model is concatenated with the parent node object3 in the Doc3 tree data model. The child node objectInfor1 and child node objectInfor2 of Doc1 tree data model appear in the output tree, because they belong to the parent node object1. The child nodes object1 of Doc1 tree data model appear in the output tree, because they belong to the parent node in Doc3 tree data model. The child nodes object3 in the Doc1 tree data model join with the child nodes object3 in the Doc3 tree and appear in the output tree.

In general the syntax of the join operators is DOC_n Tree ⊕ [condition] DOC_m Tree, where DOC_n is a tree ($n \in$ Integer) and DOC_m a tree ($m \in$ Integer).

Figure 4.6: An Example showing the Join Operator



4.4.2.2 Union operator

The purpose of the Union operator (U) is to combine two XML documents or two XML trees. It is a binary operator with two XML documents or two XML trees as input and one XML document or one XML tree as the output. This is depicted in the Algorithm shown in Figure 4.7. An example is given in Figure 4.8.

Furthermore, the union operator is commutative. More precisely, DOC1 tree U DOC2 tree = DOC2 tree U DOC1 tree. Also it is an associative operator, meaning that (DOC1 tree U DOC2 tree) U DOC3 tree = DOC1 tree U (DOC2 tree U DOC3 tree).

Figure 4.7: An Algorithm for the Union of any two DOC trees

// Input two XML documents or two DOC trees (DOC_n Tree, DOC_m Tree)
// Output DOC_{nm} Tree = (DOC_n Tree U DOC_m Tree)

1. start with the root node of DOC_n Tree and root node of DOC_m Tree
2. set root node DOC_n Tree or root node DOC_m Tree in new DOC_{nm} Tree
3. if the root node has a parent/child node in either DOC Tree
   - 3.1 perform depth-first algorithm
   - 3.2 if the parent node in DOC_n Tree = parent node in DOC_m Tree
     - 3.2.1 set the parent node in DOC_n Tree or DOCm Tree in new DOC_{nm} Tree
     - 3.2.2 while the parent node in DOC_n Tree has child node or parent node in DOC_m Tree has child node
       - 3.2.2.1 set the child node to the new DOC_{nm} Tree
       - 3.2.2.2 eliminate the duplicate child node
     - 3.2.3 repeat
   - 3.3 set a message "no parent node to new DOC_{nm} Tree" and terminate
4. set a message "no parent/child node to new DOC_{nm} Tree" and terminate
5. terminate

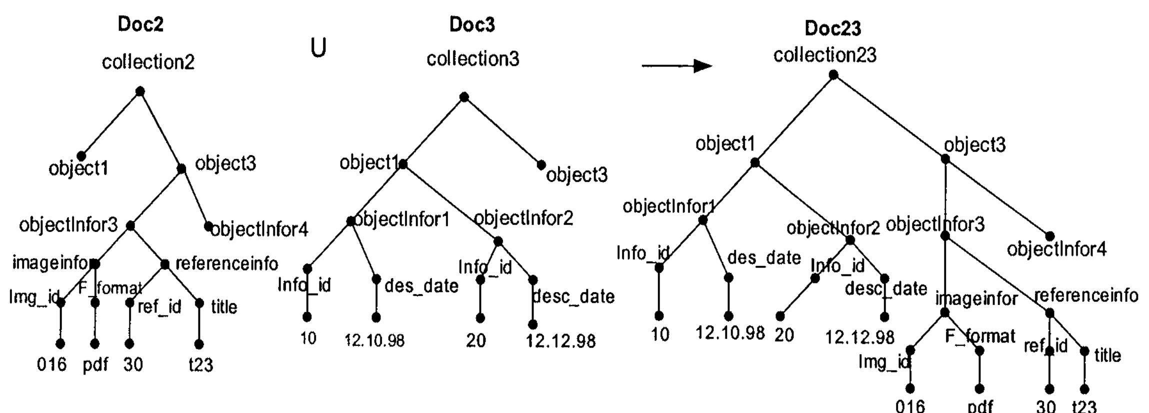
The output of the union operators is a new XML tree data containing all the elements such as root node, parent nodes and child nodes in the two input Doc trees data model without the duplication of any elements such as root nodes, parent nodes and child nodes.

In general the syntax of the union operator is DOCn Tree U DOCm Tree where DOCn is a tree (n ∈ integer) and DOCm a tree (m ∈ integer).

The union is not disjoint: duplicates are purged not renamed. As an example, the Doc tree in Figure 4.8 shows the union operators. The root node collection2 in XML Doc2 tree data merges with the root node collection3 in the Doc3 tree data, then by following the path from parent node, the object1 in Doc2 tree data model merges with the parent node object1 in the XML Doc3 tree data. The parent node object3 in Doc2 tree data model merges with the parent node object3 in the Doc3 tree data model. Because they do not exist in Doc tree, the child node objectInfor1 and child node objectInfor2 in Doc3 tree appear in the output tree. The child nodes object1 in Doc3 tree appears in the output tree with the leafs node and atomic values, because they do not exist in the Doc2 tree. The child nodes object3 in Doc2 tree and the leaf nodes and atomic values appear in the output tree, because they do not exist in the Doc3 tree data model.

In addition we can see that the union operator combines all elements and attributes in collection2 XML document and collection3 XML document after removing the duplicate elements and attributes. The output of the union operators is a new collection23 XML document which presents all the elements and attributes in collection2 XML document and collection3 XML document without duplicates.

Figure 4.8: An Example of a Union Operator



```
<collection2>
    <object1/>
    <object3>
      <objectInfor3>
          <imageInfor>
              <img_id>016</img_id>
               <fileFormat>pdf<fileFormat>
          </imageInfor>
          <referenceinfor>
               <ref_id>30</ref_id>
               <title>t23</title>
          </referenceinfor>
      </objectInfor3>
      <objectInfor4>
      </objectInfor4>
     </object3>
</collection2>
```

```
<collection3>
   <object1>
       <objectInfor1>
           <info_id>10</info_id>
           <desc_date>12.10.98</desc_date>
       </objectInfor1>
       <objectInfor2>
           <info_id>20</info_id>
           <desc_date>12.12.98</desc_date>
       </objectInfor2>
   </object1>
   <object3>
   </object3>
<collection3>
```

Collection23 shows the Union operator Result

```
<collection23>
     <object1>
         <objectInfor1>
             <info_id>10</info_id>
             <desc_date>12.10.98</desc_date>
         </objectInfor1>
         <objectInfor2>
              <info_id>20</info_id>
              <desc_date>12.12.98</desc_date>
         </objectInfor2>
     </object1>
     <object3>
         <objectInfor3>
             <imageInfor>
                  <img_id>016</img_id>
                  <fileFormat>pdf<fileFormat>
             </imageInfor>
             <referenceinfor>
                  <ref_id>30</ref_id>
                  <title>t23</title>
             </referenceinfor>
         </objectInfor3>
         <objectInfor4/>
     </object3>
</collection23>
```

Finally we can conclude that the difference between the union and the join operators is as follows:

The Union operator (U) is to combine two XML trees without a predicate. It is a binary operator with two XML trees as input and one XML tree as the output. The union is not disjoint: duplicates are purged not renamed. While the join is a binary operator, which takes two XML trees as input, and combines them into one tree of data as output. This combination is made whenever a certain expression holds true, that is when the two XML trees are joined on a predicate.

4.4.2.3 Complement operator

The complement operator (⊥) operates on two XML documents or two XML trees as input, as it is a binary operator and produces one XML document or one XML tree as output. The output of the complement operator is a new XML tree containing all element nodes (root node, parent nodes, child nodes, leaf node, atomic values) existing in the input Docl tree and not existing in the Doc3 tree. The algorithm in Figure 4.9 shows how we can complement the two DOC trees or two XML documents in general.

Figure 4.9: An Algorithm to Complement two XML trees

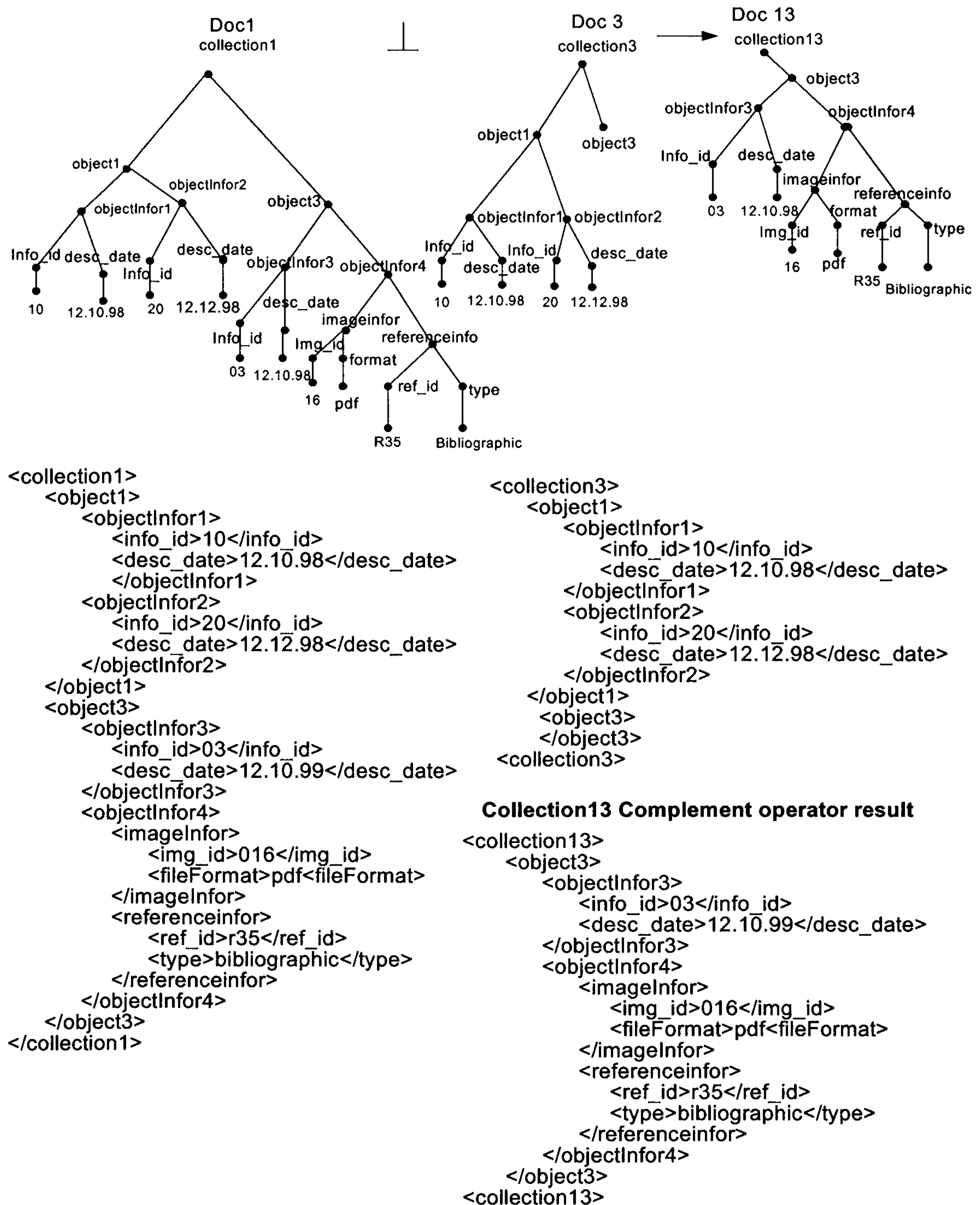
```
// Input two XML documents or two DOC trees (DOCn Tree, DOCm Tree)
// Output DOCnm Tree = (DOCn Tree - DOCm Tree)
  1. start from the root node of DOCn Tree
  2. if root node DOCn Tree and root node DOCm Tree have a parent/child node
        2.1 perform depth-first algorithm from root node
        2.2 if DOCn Tree has parent node not existing in DOCm Tree
            2.2.1 set parent node DOCn Tree to the new DOCnm Tree
            2.2.2 While parent node DOCn Tree has child node not existing in
                  DOCm Tree
                  2.2.2.1 set child node DOCn Tree to DOCnm Tree
                  2.2.2.2 If child node DOCn Tree has leaf node not existing in
                          DOCm Tree
                          2.2.2.2.1 set leaf node DOCn Tree to DOCnm Tree
                2.2.2.3 set null to DOCnm Tree
            2.2.3 repeat
        2.3 set null into DOCnm Tree
  3. set root node to DOCnm Tree and terminate
  4. terminate
```

The purpose of the complement operator is to compute the difference between the two inputs trees DOC_n Tree - DOC_m Tree

where DOC_n is a tree ($n \in$ integer) and DOC_m a tree ($m \in$ integer).

The complement operator is not commutative, more precisely DOC1 tree - DOC3 tree ≠ DOC3 tree - DOC1 tree, and also it is not associative. As an example, suppose we want the complement to the Doc1 tree data and Doc3 tree shown in Figure 4.10. We start by comparing the Doc1 tree and Doc3 trees. The root node exists in both tree successors, the parent node object1 exists in both tree data models and also the parent node object3 exists in both tree data models. But the parent node object3 contains two child nodes - objectInfor3 and objectInfor4 in Doc1 tree data model - that do not exist in the Doc3 tree data models. The child nodes of objectInfor4 in tree Doc1 data model do not exist in the Doc3 tree data models. The output returned is a new tree data model. In other words, the complement result between Doc1 and Doc3 trees is all the nodes present in the Doc1 that do not appear in the Doc3 tree.

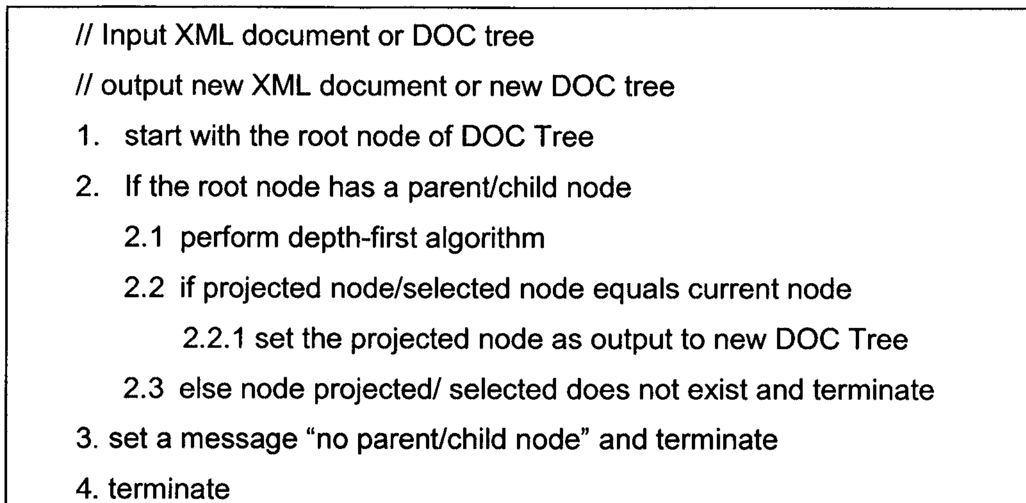
Figure 4.10: An Example of a Complement Operator



4.4.2.4 Projection operator

The input of the projection operator (π) is one XML document or XML tree, as it is unary. In general the algorithm in Figure 4.11 shows how the projection operator works and Figure 4.12 shows how we can project any nodes of the Doc tree or an element in XML document.

Figure 4.11: An Algorithm for projecting nodes of XML Tree

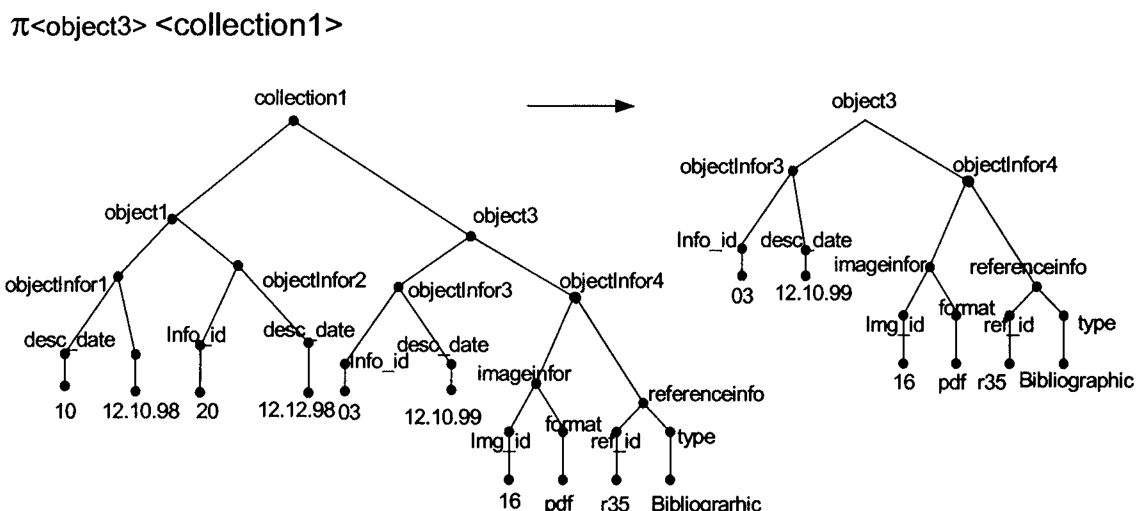


// Input XML document or DOC tree
// output new XML document or new DOC tree
1. start with the root node of DOC Tree
2. If the root node has a parent/child node
    2.1 perform depth-first algorithm
    2.2 if projected node/selected node equals current node
        2.2.1 set the projected node as output to new DOC Tree
    2.3 else node projected/ selected does not exist and terminate
3. set a message "no parent/child node" and terminate
4. terminate

As is shown in Figure 4.12, we take object3 as a parameter and the searching is performed at root node in the collection1 tree. The output of the projection operator is a new object3 tree or new XML document. For trees, projection may be regarded as eliminating nodes other than those specified. In the substructure resulting from node elimination, we would expect the (partial) hierarchical relationship between surviving nodes that existed in the input collection to be preserved. Projection in tree algebra takes one tree <C> as input and <P> as parameters. Projection starts with a search at the root node and follows the path from parent node to successor child nodes until it finds the node to project. The syntax of the projection operator is defined as follows: $\pi_{<P>}$ < C > where <P> is parameter and <C> is tree.

Figure 4.12: Projection Operator an Example

π<object3> <collection1>



```
<collection 1>
    <object 1>
        <objectInfor1>
            <info_id>10</info_id>
            <desc_date>12.10.98</desc_date>
        </objectInfor1>
        <objectInfor2>
            <info_id>20</info_id>
            <desc_date>12.12.98</desc_date>
        </objectInfor2>
    </object1>
    <object3>
        <objectInfor3>
            <info_id>03</info_id>
            <desc_date>12.10.99</desc_date>
        </objectInfor3>
        <objectInfor4>
            <imageInfor>
                <img_id>016</img_id>
                <fileFormat>pdf<fileFormat>
            </imageInfor>
            <referenceinfor>
                <ref_id>035</ref_id>
                <type>bibliographic</type>
            </referenceinfor>
        </objectInfor4>
    </object3>
</collection1>
```

Object3 shows project operator result

```
<object3>
    <objectInfor3>
        <info_id>03</info_id>
        <desc_date>12.10.99</desc_date>
    </objectInfor3>
    <objectInfor4>
        <imageInfor>
            <img_id>016</img_id>
            <fileFormat>pdf<fileFormat>
        </imageInfor>
        <referenceinfor>
            <ref_id>035</ref_id>
            <type>bibliographic</type>
        </referenceinfor>
    </objectInfor4>
</object3>
```

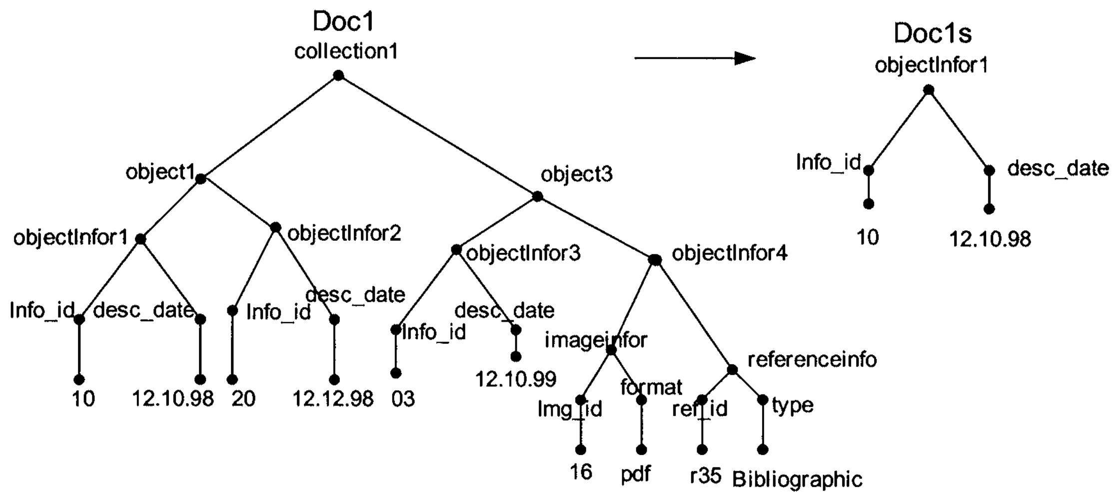
4.4.2.5 Selection operator

The purpose of a select (σ) operation is to filter out tuples in the XML algebra satisfying an expression given as a predicate. The select operator is a unary operator to take one XML document or one DOC tree as input and produce one XML document as output. The selection operator, allows us to determine a subset over a collection. It applies a given condition to each member of the collection and returns a result collection consisting of those members for which the condition evaluates true. Selection in a tree algebra takes collection nodes as input, and a <P> as parameter, and returns an output collection nodes <C>. Formally, the output σ <p><C> of the selection is a tree. We can take the parameter and start our selection by depth-first from the root node of the tree through successor child nodes until the selection condition is satisfied. This is depicted in the example shown in Figure 4.13.

As an example let us select all objects in objectInfor1 that have info_id = 10. The result of selection operators is a new tree with the root node objectInfor1. The single forward slash (/) signifies the parent-child relationship between elements or nodes.

Figure 4.13: Selection Operator an Example

$\sigma_{info_id=10}$ <collection1>/<object>/<objectInfor1>




```
<collection1>
    <object1>
        <objectInfor1>
            <info_id>10</info_id>
            <desc_date>12.10.98</desc_date>
        </objectInfor1>
        <objectInfor2>
            <info_id>20</info_id>
            <desc_date>12.12.98</desc_date>
        </objectInfor2>
    </object1>
    <object3>
        <objectInfor3>
            <info_id>03</info_id>
            <desc_date>12.10.99</desc_date>
        </objectInfor3>
        <objectInfor4>
            <imageInfor>
                <img_id>016</img_id>
                <fileFormat>pdf<fileFormat>
            </imageInfor>
            <referenceinfor>
                <ref_id>r35</ref_id>
                <type>Bibliographic</type>
            </referenceinfor>
        </objectInfor4>
    </object3>
</collection1>
```

objectInfor1 shows selection operator result

```
<objectInfor1>
    <info_id>10</info_id>
    <desc_date>12.10.98</desc_date>
</objectInfor1>
```

4.4.2.6 Expose operator

The Expose operator (ε) has one XML document or one Doc tree as input, as it is a unary operator, and produces one Doc tree as the output. The purpose of the *Expose* operator is to retrieve specific elements of the XML document or specific nodes of the Doc tree.

The Expose operator accepts as its parameters a *list of path expressions to be exposed* from the document on which it operates, with the path expression in entry-point notation. The algorithm in Figure 4.14 shows how we can expose a specific node of the Doc tree in general. Figure 4.15 also shows the result of exposing specific nodes of the Doc1 tree.

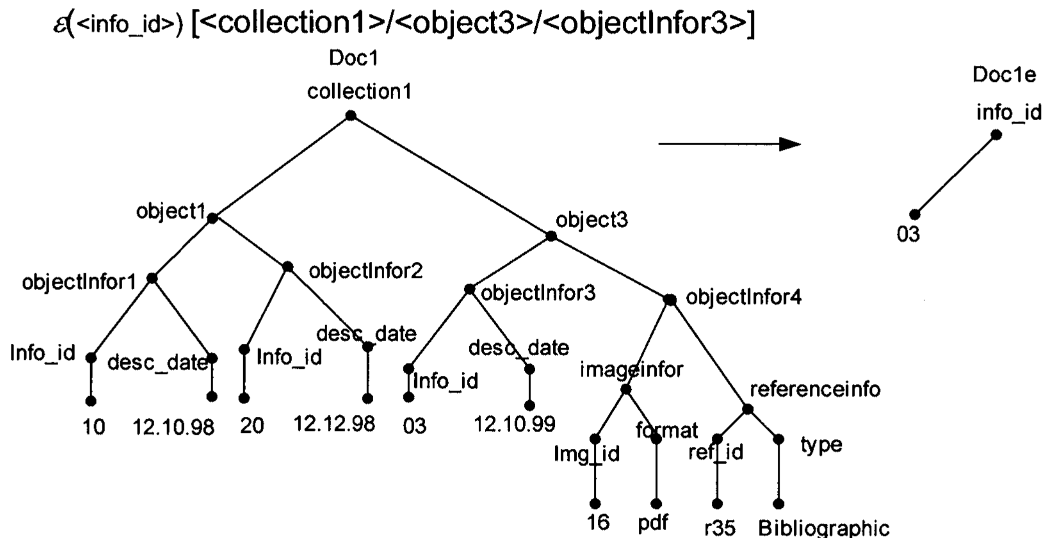
Figure 4.14: An Algorithm to expose specific elements of XML Document

```
// Input one DOC tree or one XML document
// Output one DOC tree or one XML document
1. start with entry point, it is the root node of DOC tree
2. perform depth-first algorithm
   2.1 if the parameter is equal to the specific node needed to expose
       2.1.1  return the specific node
       2.1.2  set specific node in new DOC tree
   2.2 else set exposed node or element does not exist and terminate
3. terminate
```

The Expose operator accepts as its parameter the element (parent, child) exposed from the tree data model on which it operates, with the path in entry point notation and follows the path from root node through successor child nodes until the element required to be exposed is found. The output of an Expose operator imposes a new ordering, the same as the order of its arguments. Once the nodes denoted by the path are reached, a new element content is constructed. In general the syntax of the Expose operation is: $\varepsilon_{(n)}\, p$ <C>
where n is parameter (parent, child), p is path and <C> is tree.

As an example, suppose we want to expose the child node info_id of the parent node objectinfor3 in the Doc1 tree in Figure 4.15. The entry point is the root node collection1 and following the path from parent node object3, child node objectinfor1 and child node info_id, we then retrieve the specific element. The output of the Expose operator is a new XML tree as Doc1e tree data model.

Figure 4.15: Expose Operator an Example




```
<collection1>
    <object1>
        <objectInfor1>
            <info_id>10</info_id>
            <desc_date>12.10.98</desc_date>
        </objectInfor1>
        <objectInfor2>
            <info_id>20</info_id>
            <desc_date>12.12.98</desc_date>
        </objectInfor2>
    </object1>
    <object3>
        <objectInfor3>
            <info_id>03</info_id>
            <desc_date>12.10.99</desc_date>
        </objectInfor3>
        <objectInfor4>
            <imageInfor>
                <img_id>016</img_id>
                <fileFormat>pdf<fileFormat>
            </imageInfor>
            <referenceinfor>
                <ref_id>r35</ref_id>
                <type>bibliographic</type>
            </referenceinfor>
        </objectInfor4>
    </object3>
</collection1>
```

Exposed operator result

```
<info_id>03</info_id>
```

4.4.2.7 Vertex operator

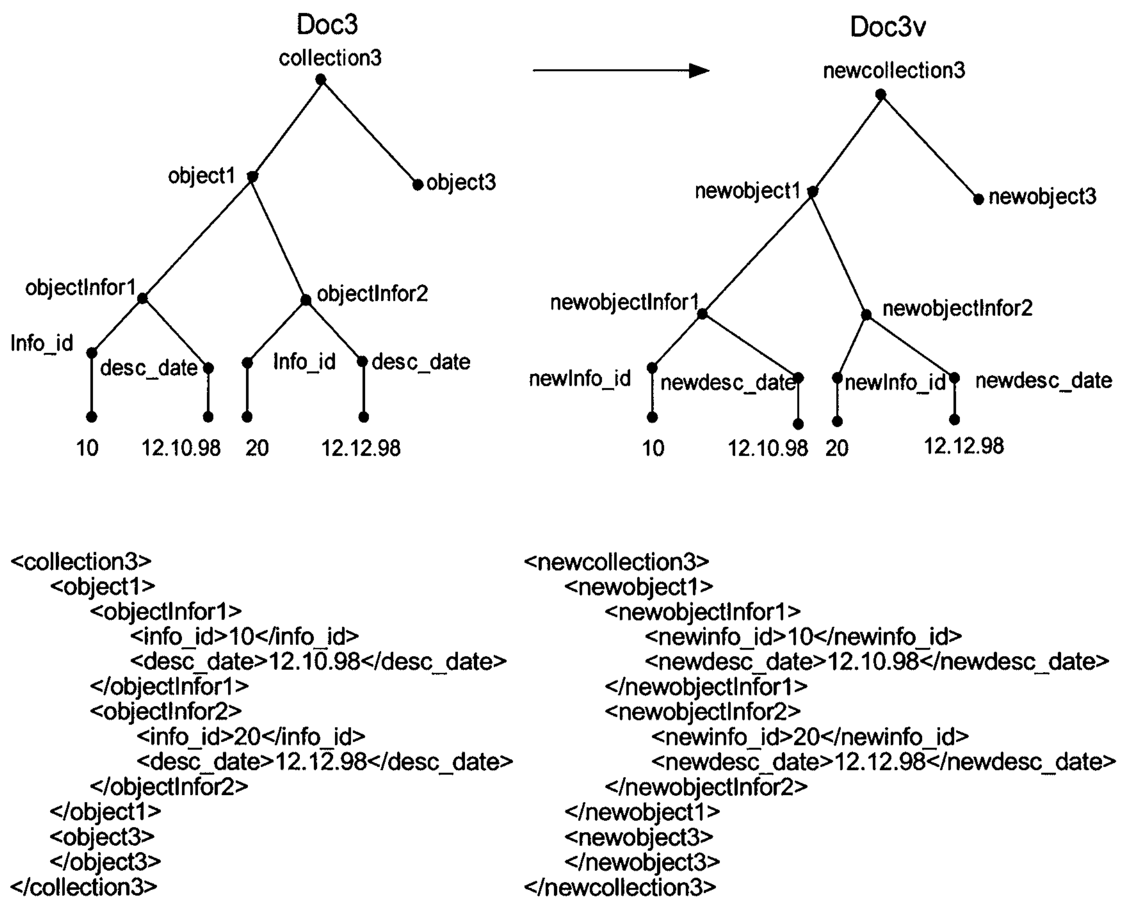
The vertex operator (v) has one XML document or Doc tree as input, as it is unary, and creates the actual XML vertex that will encompass everything created by an Expose operator. It arranges the element content according to the order indicated by its input. Figure 4.16 shows how we can use the vertex operator to create an actual XML vertex.

Furthermore we will use the notation (ν) for the Vertex operator and the notation (ε) for the Expose operator when creating the XML tree to which nodes can be connected, as well as the named edges that lead to the newly defined XML document or XML tree. These operators handle arbitrary complexity in the constructed XML elements or XML tree nodes. As can be seen from the example shown in Figure 4.16, a combination of a complex Vertex and Expose operation, the output can generate a new XML tree or new XML document. In the example we used the single forward slash (/) to signify the parent-child relationship between elements or nodes. In tracing a path through the tree the expression starts at the root node of the tree.

In general the syntax of the vertex is v[type] (expression) where *type* is the vertex element or node type and the *value* is the vertex expression.

Figure 4.16: Vertex Operator an Example

v<newcollection1>[ε(v<newobject1>[<collection1/object1>], v(<newobject3>[collection1/object3>])]



4.4.3 A complete algebra

As mentioned earlier our XML Algebra is a tree structure that consists of algebra operators. There are two types of operators in our algebra. Firstly, we have the algebra operators such as *join, union, complement, project, select, expose* and *vertex*. Every operator has one tree as output, which makes it distinct from other algebra operators. Secondly, the algebra relational operators are *universal, subsuming, equivalence* and *similarity*.

Codd [76] defined a relational algebra as being complete if it comprises the operators *join*, *difference*, *project*, *select* and *product*. The natural join is defined here from a relational product. No other joins or the product itself are available. So our algebra is relationally complete except when a user wishes to use an unrestricted product or other types of join. In fact the algebra operators we have mentioned are suitable not only for our domain but for other domains as well, provided that such domains do not need the product operators. The *product* operator joins two sources together without a condition. There are no parameters for this operator. This operator can be complex as the resulting table has M x N rows, where M and N are the number of rows of the input sources and P + Q columns, where P and Q are the number of columns of each of the inputs. Because of the large output table, executing a *product* is avoided when possible and in tree-based systems is unimportant. The *product* operation is needed to compare rows within a table with one another in relational database applications such as finance and health. The *Product* operation is not so necessary in tree operations where it is very unwieldy. So we can conclude that our algebra is not complete but it is adequate for tree-based applications.

4.5 Our XML Query Language

Basically, XQuery [91, 95, 131, 150] is a language that lets us retrieve data items from an XML-formatted document. Furthermore XQuery is a language containing one or more query expressions. XQuery supports conditional expressions, element constructors, FOR, LET, WHERE, RETURN (FLWR) expressions, expressions involving operators, function calls and quantifiers, type checking and path expressions. Some XQuery expressions evaluate to simple nodes such as elements and attributes or atomic values such as strings and numbers [147].

The syntax for retrieving, filtering and transforming records uses FOR, LET, WHERE, RETURN clauses.

A FLWR expression creates some bindings, applies a predicate and produces a result set. XQuery does not conform to the same conventions as SQL. But XQuery and SQL share some similar concepts. Both languages provide keywords for projection and transformation operations (SQL SELECT or XQuery RETURN). SQL supports joins between tables and XQuery supports joins between documents.

Furthermore, the XQuery model we have developed is based on the existing standard (W3C XML Query) for querying information in XML trees. So, the standard is a satisfactory starting point for us in our effort to develop a domain-specific XQuery and was therefore adopted as the basis for our way forward but is too cumbersome for many applications. Our XQuery is a simplified and restricted version of the W3C XQuery for querying XML data. This is because we have developed a new specific tree algebra and we need a restricted version of XQuery suitable for a representative class of problems to satisfy the requirement. In the following sections more details are given on XQuery.

4.5.1 Syntax

The types of syntax grammar for XQuery consist of core expressions and types [129]. The expressions mainly include Boolean expression, path expression, and tree expression. The tree expression is constructs such as for...where...return, or let...for return and if... then...else and so on. Furthermore the path expression is composed of basic steps that are separated by a forward slash. The single forward slash (/) signifies the parent-child relationship between elements. The double forward slash (//) signifies the ancestor-descendant relationship. Both are called containment queries because they restrict the path of an XML tree. In the following subsections we will formulate two types of queries on the algebra operators: simple queries and complex queries and also those related to syntax.

- **Expressions**

constant *Const* unit Doc = 1
variable *Var*
document Doc0,...Docn
expression type ::= boolean expression | tag expression
| path expression | tree expression
boolean expression *<BoolExpr>* ::= and | or | not
tag expression *<TagExpr>* ::= startTag | endTag
startTag ::= '<' tagName '>'
endTag ::= '< / ' tagName '>'
tagName ::= QName | Variable
QName ::= *Name*
Name ::= Identifier
path expression *<PathExpr>* ::= path
| *'/' path*
| *'//' path*
| *<Expr> '/' path '/' <Expr>*
| *<Expr> '//' path '// <Expr>*
path ::= '/' | '//'
tree expression *<TreeExpr>* ::= *tree*
tree ::= *root node* | *parent node*| leaf node
expression *<Expr>* ::= *Const*
| *Var*
| if *<BoolExpr>* then *<TreeExpr>*
else *<TreeExpr>* /* conditional */
| for *Var* in *<PathExpr>* do *<TreeExpr>*
| where *< BoolExpr>*
| return *<TreeExpr>*
| let *Var* = *<PathExpr>* do *<TreeExpr>*
| ( ) /* empty tree expression *
<List Expr> ::= *<Expr>, <Expr>*

- **Variables**

X, obj, inf, coll, inst, exb

4.5.2 Simple Examples in Our XML Query Language

As mentioned above we will use our own algebra to formulate queries executed by the system (software) according to the XML transformation to produce the result. The simple queries are tree expression and path expression. The query 1 below is to join the two XML trees such as Doc1 and Doc3 trees. While a simple query 2 for projecting nodes of collection tree and simple query3 for selecting elements satisfying an expression e.g. select the objects nodes that have the (objNumber=1234) of object in collection tree. The simple query 4 shows how we can union the two XML trees such as Doc2 and Doc3 and the simple query 5 to complement the Doc1 and Doc3 trees. Furthermore, Query 6 is to expose specific nodes of a collection tree through the sub-trees object3 and objectInfor4.

1. <Doc1> ⊕ <Doc3>

   This query is presented in detail in Figure 4.6.

2. π<object3 ><collection1>

   This query is presented in detail in Figure 4.12.

3. σ objNumber=1234<collection>/<object>

   This query selects all the object elements in the collection where the objNumber is equal to "1234".

Expected result for the above XQuery

```
<object>
<objNumber>1234</objNumber>
<objectInfor>
    <info_id>10</info_id>
    <desc_date>12.10.98</desc_date>
</objectInfor>
<objectInfor>
    <info_id>20</info_id>
    <desc_date>12.12.98</desc_date>
</objectInfor>
</object>
```

4. <Doc2> ∪ <Doc3>

   This query is presented in detail in Figure 4.8.

5. <Doc1> - <Doc3>

   This query is presented in detail in Figure 4.10.

6. The query below will expose the objectInfor has info_id= 03 of object in the collection tree.

ε(<info_id=03>) [<collection>/<object>/<objectInfor>]

Expected result of the above XQuery

```
<objectInfor>
   <info_id>03</info_id>
   <desc_date>12.10.99</desc_date>
</objectInfor>
```

4.5.3 Complex Examples in Our XML Query Language

In this section we will formulate complex XQueries in our own XML algebra as examples to support the conditional expressions, element constructors, FOR, LET, WHERE, RETURN (FLWR) expressions and path expressions [150]. The following examples introduce XQueries with the expected results in more detail. The run of the programs developed later gave in all cases the expected result for these test queries.

1. The query below selects all the object1 nodes of the collection3 tree

```
For obj in <collection3> do
Return <obj>
Where obj = <object1>
```

Expected result of the above XQuery

```
<object1>
     <objectInfor1>
        <info_id>10</info_id>
        <desc_date>12.10.98 </desc_date>
     </objectInfor1>
     <objectInfor2>
        <info_id>20</info_id>
        <desc_date>12.12.98 </desc_date>
     </objectInfor2>
</object1>
```

2. The query below selects all the objectinfor2 nodes of object1 in collection1

```
For inf in <collection1>\<object1> do
Return <inf>
Where inf = <objectinfor2>
```

Expected result of the above XQuery

```
<objectInfor2>
    <info_id>20</info_id>
    <desc_date>12.12.98</desc_date>
</objectInfor2>
```

3. The query below selects all the objectInfor4 elements of object3 elements in collection1.

```
For inf  in <collection1>\<object3> do
Return <inf>
Where inf = <objectInfor4>
```

Expected result of the above XQuery

```
<objectInfor4>
    <imageInfor>
        <img_id>016</img_id>
        <fileFormat>pdf</fileFormat>
    </imageInfor>
    <referenceinfor>
        <ref_id>r35</ref_id>
        <type>bibliographic</type>
    </referenceinfor>
</objectInfor4>
```

4. The query below projects the object nodes in collection3 tree

```
For obj in <collection3> do
Return <object>
```

Expected result of the above XQuery

```
<object>
   <objectInfor1>
      <info_id>10</info_id>
      <desc_date>12.10.98</desc_date>
   </objectInfor1>
   <objectInfor2>
      <info_id>20</info_id>
      <desc_date>12.12.98</desc_date>
   </objectInfor2>
</object>
```

5. The query below exposes the imageInfor nodes of objectInfor4 of object3 in collection1 tree.

```
Let inf = <collection1>/<object3>/<objectInfor4> do
Return < inf >
```

Expected result of the above XQuery

```
<imageInfor>
    <img_id>016</img_id>
    <fileFormat>pdf</fileFormat>
</imageInfor>
```

6. The query below selects the objectInfor4 elements of object3 in collection1

```
Let inf=<collection1>/<object3> do
Return <inf>
Where inf = <objectInfor4>
```

Expected result of the above XQuery

```
<objectInfor4>
    <imageInfor>
        <img_id>016</img_id>
        <fileFormat>pdf</fileFormat>
    </imageInfor>
    <referenceinfor>
        <ref_id>r35</ref_id>
        <type>bibliographic</type>
    </referenceinfor>
</objectInfor4>
```

7. The query below selects the objectInfor4 elements of object3 in collection1

```
For obj in <collection1> / <object3> do
If obj = <objectInfor4> then Return <obj>
else ()
```

Expected result of the above XQuery

```
<objectInfor4>
    <imageInfor>
        <img_id>016</img_id>
        <fileFormat>pdf</fileFormat>
    </imageInfor>
    <referenceinfor>
        <ref_id>r35</ref_id>
        <type>bibliographic</type>
    </referenceinfor>
</objectInfor4>
```

8. The query below projects the object nodes of collection1 tree

```
For obj in <collection1> // <object> do
Return <objectInfor>
```

Expected result of the above XQuery

```
<objectInfor>
    <info_id>03</info_id>
  <desc_date>12.10.99</desc_date>
</objectInfor>
```

9. The query below returns the exhibition information on exhibition

```
For exh in <exhibition> do
Return <exh>
Where exh/<opened>="yes"
```

Expected result of the above XQuery

```
<exhibition>
   <exhId>A001</exhId>
   <title>malitry</title>
   <openDate>01.03.04</openDate>
   <description>air show</description>
   <opend>yes</opend>
</exhibition>
```

10. The query below projects the institution information on institution

```
For inst in <institution> do
Return <inst>
```

Expected result of the above XQuery

```
<institution>
   <inst_id>35</inst_id>
   <name>British museum</name>
   <institution_type>private</institution_type>
   <country>Great Brittian</country>
   <description>yes</description>
   <url>http//:www.virtualexh.co.uk</url>
</institution>
```

4.5.2 Semantics of Our XML Query Language

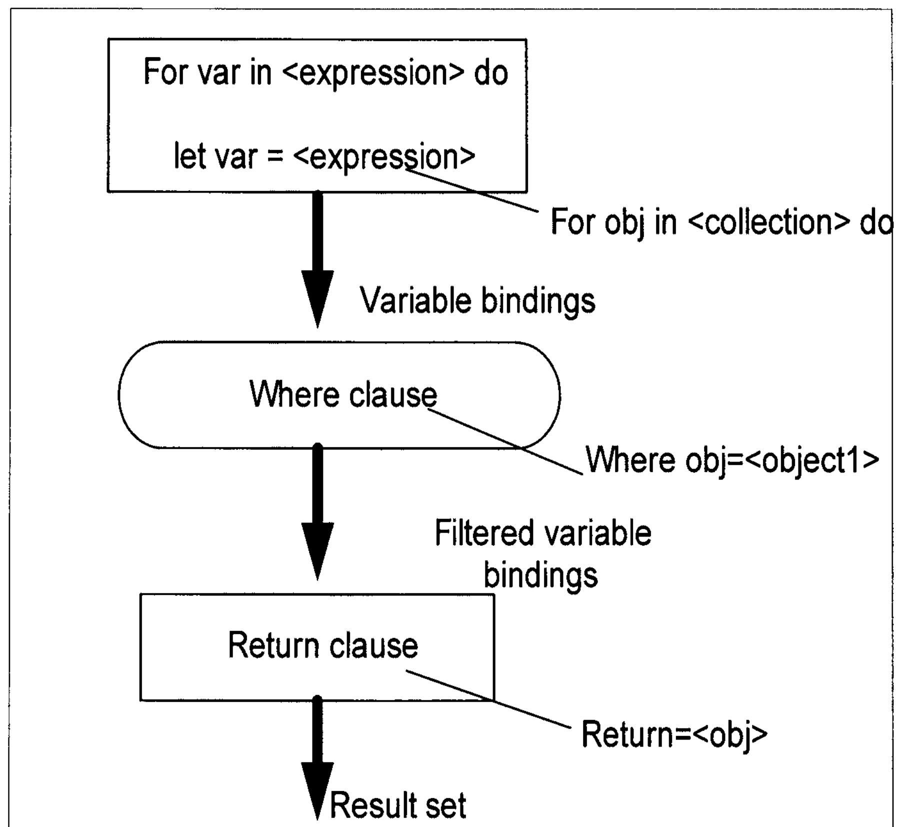
4.5.2.1 Concepts XQuery model

This section introduces some of the terminology of the XQuery data model such as root, node, XPath, loop and the conditions:

- Root (ancestor or parent)
- Node (parent or child, leaf)
- XPath
- Loop such as for, do, let
- Conditions such as If, then, else, where, return

- FLWR (For-Let-Where-Return) Expression

Figure 4.17: FLWR expression semantics



The FLWR expression can be explained with the diagram shown in Figure 4.17 in more detail, where F stands for FOR, L stands for LET, W stands for WHERE and R stands for RETURN. The FLWR expression must contain at least one for or let clause to evaluate expressions and assign (bind) the results of the expressions to variables. The difference between the for and let clause is that the for binds a variable to each element specified by the path expression while the let binds a variable to the whole collection of elements.

In addition the FLWR expression is useful for processing and restructuring data from one or more XML documents. The For loop iterates through each element in the document, while the where clause selects those elements where the condition is satisfied and then the return clause lists the result.

To interpret the diagram shown in Figure 4.17 we use the following example.

```
For obj in <collection> do
Where obj = <object1>
Return <obj>
```

In Figure 4.17 the variable Var is converted to a variable obj in the example. The expression in Figure 4.17 is the <collection> in the example. The for loop iterates through the <collection> element that contains objects elements, while the where clause in Figure 4.17 is interpreted as where obj=<object1> to select those elements whose <object1> appears in <collection> elements. The return clause is interpreted as return <obj> as the obj contains the elements of <object1>. Then the result of the query looks like:

```
<object1>
   <objectInfor1>
      <info_id>10</info_id>
      <desc_date>12.10.98</desc_date>
   </objectInfor1>
   <objectInfor2>
      <info_id>20</info_id>
      <desc_date>12.12.98</desc_date>
   </objectInfor2>
</object1>
```

Here is another example to interpret the XQuery with the let clause as in Figure 4.17. As we mentioned, a let clause binds variables directly to the expression as a whole. However, the for loop binds variables to each value of the sequence returned by the expression. The query returns the obj_id elements for each <object> in the <collection>.

In other words, the expression binds the variable inf to the sequences of the obj_id elements in the <collection>.

```
let inf = <collection>/<object>/<obj_id> do
return <inf>
```

The return clause lists the elements of <inf> and then the result of the query looks like:

```
<object>
  <info_id>10</info_id>
  <info_id>30</info_id>
</object>
```

4.6 Conclusion

We have presented a framework for representing XML algebra and XQueries over XML documents. The operators and the relational operators we have defined explore XML data as well as constructing new XML documents. In other words the operator works on one tree if it is a unary operator and on two trees if it is a binary operator producing one XML tree as output. The contribution of this chapter is that it introduces new algebra that operates on a novel specific data model because the input and output of our algebra works on XML trees. The algebraic relational operators work on XML documents and XML document defined as a tree. Furthermore, our XML algebra is generic, since our XML algebra model is XML Schema-driven. Also our algebra has a good data structure and a simple presentation of the data. There are two types of operators in our algebra: firstly, the algebraic operators are *join, union, complement, project, select, expose* and *vertex.* Every operator has an output of a tree, which makes it distinct from other algebra operators. Secondly, the algebra relational operators are *universal, subsuming, equivalence* and *similarity.* Our algebra framework can be used in an integrated architecture for distributed information processing and its components will be XML Schema-driven.

Moreover, we will use our algebra for implementing the main offline components of the system for data processing (XML Schema → SQL schema generation, XML Schema → HTML presentation and XML Schema → XML query interpreter). This will be shown in Chapter 5.

Chapter 5:

XML Mapping and Component Generation

In this chapter we will introduce the mapping of XML documents into a variety of formats and investigate how the XML document can be stored in different ways. Also, how the XMLSchema drives the architecture for the generation of components. In general, any application that has the capability to work with XML documents will need to display the structure of its related data into a different format specified for a particular occasion, due to its nature in working in a heterogonous environment [47]. Accordingly, transforming a document from one data structure to another is needed. Such a transformation process is essential, especially when dealing with XML documents.

Of course, designing "traditional" software transformation tools for that purpose can achieve such a task. However, the power of having a cross-platform and an XML independent language would be lost. Precisely, isolating content from formatting needs to be considered, especially when dealing with Web based documents. Therefore, any method of transforming XML documents into different formats such as XML, HTML, SQL, flat files or WML needs to be tailored so that it can be used with different platforms/languages.

Originally, XML was created to meet the challenges of *data exchange* on Web applications or between applications and users, not for data presentation purposes. To deal with presentation issues, XML needs to be used in conjunction with stylesheets to be easily viewed on the Web [1].

For this reason, the extensible stylesheet language was created. XSL (eXtensible Stylesheet Language) has been developed as part of the W3C stylesheets activity [96]. It has evolved from the CSS language to provide an even richer stylistic control, and to ensure consistency of implementations. It also has document manipulation capabilities beyond styling.

In the following we will introduce different types of methods for transforming XML documents into a different format with XSL stylesheet transformation such as the Offline and Online transformation (for definitions, see page 4 of thesis) - in other words to develop the technological solution for the system such as generation of components. Furthermore, we will introduce the mapping of XML documents.

5.1 Mapping

Actually, when the data are to be translated between XML documents and database there should be some means of mapping formulated for the data before they can be transferred either to the database or into the document [148]. Most of the techniques use relational mapping for transforming data between XML and the database [2,48,124,143]. In this chapter we will present different types of mapping for XML document, such as mapping from tree-to-tree [114] which means XML to XML and XML to XHTML. Moreover, we will cover mapping of the Tree to relation [97,130,133,136,144].

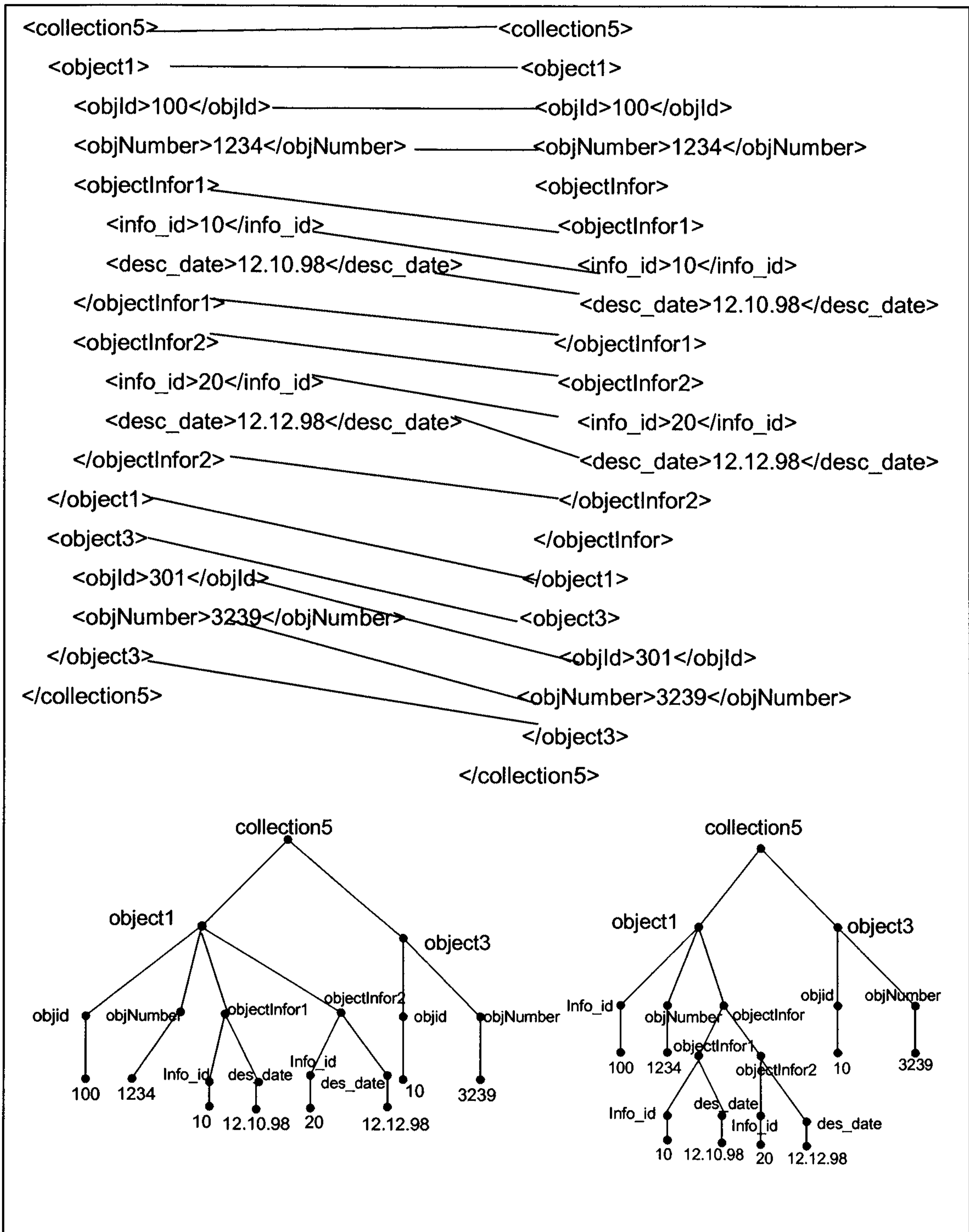
5.1.1 Mapping Tree to Tree

In the following subsections we will introduce different kinds of mapping from tree-to-tree such as XML to XML and XML to XHTML.

5.1.1.1 Mapping XML to XML

Basically, we need the mapping from XML to XML for merging and coalescing data from multiple diverse sources into different data formats and to manipulate the data from an existing XML document. As can be seen in the example, shown in Figure 5.1, this process is simplified by creating a set of mappings from one XML document (Tree) to another XML document (Tree). Mapping describes how the elements (parent node, child node) and attributes of an XML document are mapped to another XML document. The starting point of the root node <collection5> in the left XML document is mapped to root node <collection5> in the right XML document, and also the parent node <object1> is mapped to parent node <object1> in the right XML document. The child nodes of the left XML document <objId> and <objNumber> are mapped to <objId> and <objNumber> in the right XML document. The <objectInfor1> and <objectInfor2> are merged and mapped into the right XML document as <objectInfor>. The parent node <object3> in the left document is mapped to parent node <object3> in the right document. All data types in XML document are mapped between the same types as integer to integer, string to string, character to character, and so on.

Figure 5.1: An Example for mapping XML to XML



In general we present an algorithm shown in Figure 5.2 for mapping one XML document to another XML document. Let us walk through the algorithm using the XML document in Figure 5.1 as an example. The algorithm is invoked with the root node of the XML schema (collection5 in our example).

If the root node has no parents or child nodes, the task is terminated. If it has parents like object1 and object3 in our example, perform breadth-first traversing from the root node of the tree or XML document and record the level of all nodes and flag the nodes which point to an upper or the same level node (line 2.1 in Figure 5.2). Create an empty XML document, and set the start tag of the root node into the document (line 2.2 in Figure 5.2), then merge the nodes, which have the same parent node, child name and atomic value and set as the new element in the empty XML document (line 2.3 in Figure 5.2). If the node is an unnamed node it is necessary to merge any node in the same level and position (line 2.4 in Figure 5.2). Then the algorithm continues with the child nodes (line 2.5 in Figure 5.2) and generates XML data by applying the algorithms shown in Figure 5.3. At the end the algorithm writes an end tag element to the new XML that is created (line 2.6 in Figure 5.2) and then deletes the added flag (line 2.7 in Figure 5.2).

Figure 5.2: An Algorithm for mapping XML to XML document (Tree to Tree)

// Algorithm for mapping XML tree to XML tree
// Input: XML tree
// Output: XML tree

1. start with the root node of XML tree

2. if the root node has a parent/child node

    2.1 perform breadth-first traversing from the root node of the tree, record the level of all nodes and add a flag to edges which point to a node on a higher level or the same level node

    2.2 create an empty XML document, write some routine information and start with tag of root element into the document

    2.3 merge the nodes which have the same parent, child name and values under the new node and write it to empty XML document

    2.4 If the node is an unnamed node, it is merged with any node on the same level and position

    2.5 for all children C of root node generate XML data by applying algorithm 2 in Figure 5.3 for C (children)

    2.6 write *end tag* root element into XML document (i.e. </...>)

    2.7 delete the flags that have been added in step 2 and a new tree created

3. report no parent/child node and terminate

4. terminate

Figure 5.3: An Algorithm to generate XML data by depth-first Traversing

```
// Algorithm to generate XML data by depth first traversing
// Input: Node N of tree graph
// Output: XML data

1. if node N is an atomic object then
   1.1 Label:= name of edge which connects node N and its parent node
   1.2 Idvalue:= identifier of node N
   1.3 Atom Value:= string format value of node N
   1.4 write into XML document
2. if node N is a complex object then
   2.1 Label:= name of edge which connects nodes N and its parent node
   2.2 Idvalue:= identifier of node N
   2.3 write data into XML document
   2.4 for all child node C of node N do
       2.4.1 if edge E which connects nodes N and C has added flag then
             2.4.1.1 SubLabel:= name of edge E
             2.4.1.2 SubIdvalue:= identifier of node C
             2.4.1.3 write data into XML document
       2.4.2 else generate XML data by recursively applying the algorithm
       2.4.3 string:= </Label>
       2.4.3 write string into XML document
   2.5 end for
3. terminate
```

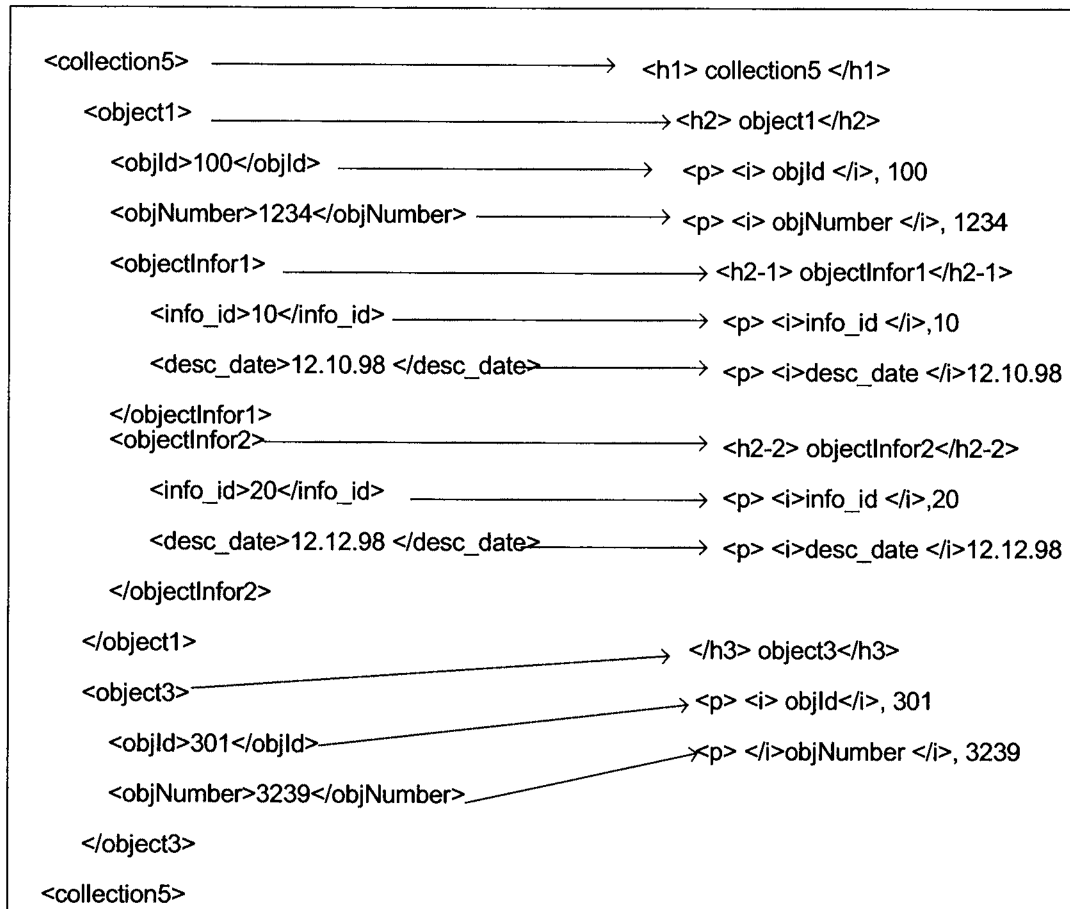
5.1.1.2 Mapping XML to XHTML

We need this type of mapping because the HTML combines data and presentation in the web. Furthermore, the HTML is the only language always understood by browsers. Since XML addresses only the data, we need to convert it into XHTML, in order to display and present it on the Web [27, 50].

It can be seen from the example shown in Figure 5.4 that the root node <collection5> in the XML document is mapped to <h1> in the HTML document, because they correspond. To depth-first in the XML document, parent node <object1> is mapped to <h2> in the HTML document. All the child nodes in the XML document are mapped to <p> in the HTML document like <objId> is <p> in the HTML, also the atomic value for <objId> is mapped to the child as <i> in the HTML document.

The parent node <objectInfor1> in the XML document is mapped to <h2-1> in the HTML document because they correspond. So all elements and attributes in the XML document are mapped to corresponding objects in the HTML document.

Figure 5.4: An Example for mapping XML to XHTML



In general we present the algorithm shown in Figure 5.5 for mapping an XML document to an XHTML document. Let us walk through the algorithm using the XML document in the left hand part of Figure 5.4 as example. The algorithm is invoked with the root node of the XML document (collection5 in our example). Since the root node has no parents or children, we set null and terminate the processing. If it has parents, perform depth-first (line 2 in Figure 5.5) and map all parent nodes to corresponding <h> in the HTML document, object1 into <h2> and object3 to <h3> in our example (line 2.2 in Figure 5.5).

We also map all child nodes in the XML document to <p> in the HTML document (line 2.2.1 in Figure 5.5). Finally write end tag body to the HTML document and terminate (lines 2.3 - 3 in Figure 5.5).

Figure 5.5: An Algorithm for mapping XML to XHTML (Tree to Tree)

```
// Algorithm for mapping XML Tree to XHTML Tree
// Input: XML tree
// Output: XHTML

 1. start from the root node of the XML tree

 2. if the root node has no parent node set "no parent node" and terminate
    Otherwise, perform depth-first traversing from the root node of tree

    2.1 map the root node in XML to <h1> in HTML document, as it corresponds to the
        <h1> node in HTML document

    2.2 for all parent nodes in the XML document, map to corresponding <h> in HTML
        document

        2.1.1 for all child nodes in XML document, map those corresponding to <p> in
              HTML document

    2.3 at end write end tag body into HTML document (i.e. </body>, then HTML
        document is created and terminate
3. terminate
```

5.1.2 Mapping Tree to Relation

In the following sections we will introduce the mapping from tree to relation such as nested relational tables [100].

5.1.2.1 Mapping XML to Nested Relational Tables

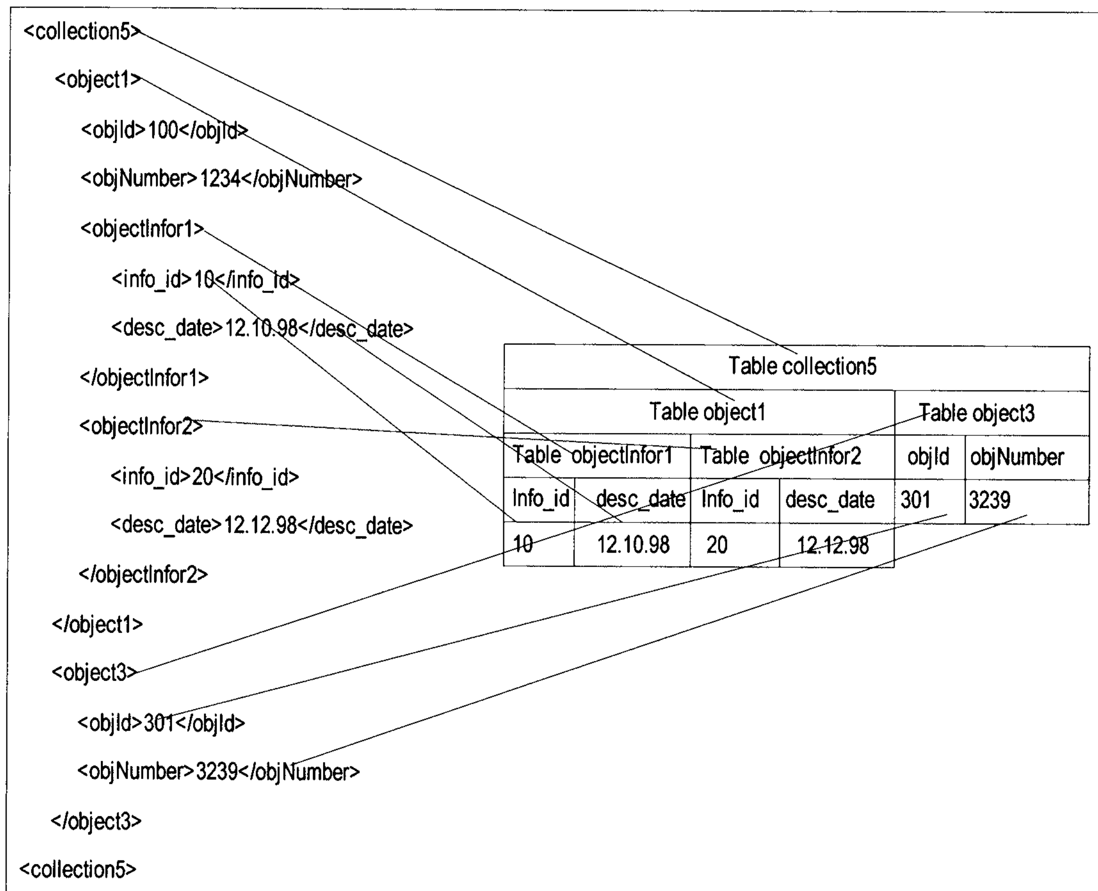
A nested relational database consists of a set of tables, where each table consists of a fixed collection of columns. An indefinite number of rows occurs within each table. However, each row must have a unique primary key, which is an identifier or handle for that particular type of data.

The purpose of mapping an XML document to nested relational tables is for simplicity and to be able to store an entire XML document as a single database.

An example is shown in Figure 5.6 for mapping an XML document to nested relational tables.

As can be seen from the example, the root node <collection5> of the XML document is mapped as <collection5> table name. Also the parent node <object3> is mapped as the object name to the nested table. Furthermore, the child element <objId>, <objNumber> in the XML document is mapped as columns in the table named object3. The parent node <object1> is mapped as the name to the table. The child element objectInfor1 is mapped as an inner table name and also the child elements <info_id> and <desc_date> are mapped as column names to the inner table. The child element objectInfor2 is mapped as an inner table name and also the child elements <info_id> and <desc_date> are mapped as column names to the inner table. This type of mapping is called nested relational mapping.

Figure 5.6: An Example for mapping XML to Nested Relational Tables



5.2 Developing a Technological Solution for the System

It is possible to code by hand an XSL stylesheet that validates an XML document against some or all constraints of XML schema. But the main goal of the following sections introduces general techniques as a technological solution for different problems such as:

1) SQL schema generation
2) XSL stylesheet generation, and
3) XQuery interpreter generating.

In other words we will introduce generic XSL stylesheets as interpreters for solving the different problems that we have mentioned. More details on these solutions will be dealt with in the following sections.

5.2.1 SQL schema generations

The DOM [139] is a specification that comprises a set of interfaces that allow XML documents to be parsed and manipulated in memory. These interfaces are defined by the W3C who provide Application-Programming Interfaces (APIs) for every major programming language using the Interface Definition Language. To be precise, the DOM is not designed specifically for Java [119], but to represent the content and model of documents across all programming languages and tools. Bindings exist for JavaScript, Java, CORBA and other languages, allowing the DOM to be a cross-platform and cross-language specification. DOM has been specifically designed by the W3C to model XML documents in a tree structure, beginning with a root element and comprising any number of child and sibling nodes. When an XML document is parsed by DOM, the entire content of the document is read into memory and providing it is a well-formed document, it is then represented as a tree structure. The elements can then be accessed and traversed from the root node using the tree relationships. The main interfaces for an application with DOM are as follows: 1) Node, this is the base type, representing a node in the DOM tree. 2) Document, representing the entire XML document as a tree of Nodes (the DOM parser will return Document as a result of parsing the XML). 3) Element, representing elements of the XML document. 4) Attribute, representing an attribute of some XML element; the interface enables setting/getting the value of that attribute. 5) Text, used to represent the text content of an element (i.e. the text between tags that is not part of any child element).

The DOM tree is composed of nodes, each of which represents a parsed document. Based on these interfaces, we will use XMLSchema parse file (DOM) as input for our algorithm and also the generated XSL stylesheet shown in Figure 5.10. Now we will present an algorithm that can be used automatically to generate the SQL schema. In particular, we present a translation algorithm that takes as input an XMLSchema and the generated XSL stylesheet for specifying the construction of an SQL schema as the output. In other words this algorithm is the technological solution to the problem of generating the SQL schema. The algorithm is shown in Figure 5.7 for the generation of SQL Schema from XMLSchema. In the following sections we will show how the algorithm works and how we can extend the initial SQL schema shown in Figure 5.9.

Figure 5.7: An Algorithm to generate SQL Schema from XMLSchema

```
// Algorithm generate SQL Schema (XMLSchema (DOM), XSL stylesheet)
// Input XMLSchema, XSL stylesheet
// Output SQL Schema  (SQL DDL)

1. start
2. if the input arguments of the algorithm (XMLSchema, XSL stylesheet) exist
   2.1 Build DOM and parse it
   2.2 if parsing XMLSchema is done then DOM will build dynamically
       2.2.1 perform XSL stylesheet transformation
       2.2.2 if transformation is done
             2.2.2.1 transformation processing (XMLSchema, XSL stylesheet)
             2.2.2.2 the XSL stylesheet template adds SQL clause CREATE TABLE to the start of each
                     separate SQL table followed by the matching complexType node of the DOM tree
             2.2.2.3 for each complexType create a separate SQL table
                     2.2.2.3.1 generate a primary key to each SQL table as table name and followed by
                               the string "_id"
                     2.2.2.3.2 generate the foreign key to the appropriate table name followed by the
                               string "_id" and references to the table name and it's the primary key
                     2.2.2.3.3 If the root node (complexType) has parent/child nodes
                               2.2.2.3.3.1 the XSL stylesheet template matches and maps all child
                                           nodes of DOM tree as column names to created SQL tables
                               2.2.2.3.3.2 the XSL stylesheet templates walk through DOM and match
                                           the attribute nodes and data types
                               2.2.2.3.3.3 map attribute nodes and data types to column names in SQL
                                           table as data types
                     2.2.2.3.4 report no parent/child node and terminate
             2.2.2.4 finish the execution, the SQL schema is generated
       2.2.3 report transformation errors and terminate
   2.3 report parsing errors and terminate
3. report reading errors and terminate
4. terminate
```

5.2.1.1 How the algorithm works

Figure 5.8 shows how the algorithm works and how the XML tree data semantics for the XML algebra map into SQL semantics, which is relational algebra. The following shows how the algorithm maps an XMLSchema to an SQL schema:

- For each complexType name in XMLSchema, create a separate table name and primary key column. As in our example the complexType names are collection, object and exhibition mapped as collection table name, object table name and exhibition table name respectively.

- To generate the primary keys of the tables that takes the table name followed by suffix "_id". In our example the table names are collection and object and the primary key generated, as we mentioned, is the table name followed by the string "_id". Therefore the primary keys are collection_id and object_id.

- To create a foreign key of table names that takes the table name followed by "_id", with the key word References followed by the table names and its primary key. In our example, the institution_id column in the collection table is a foreign key.

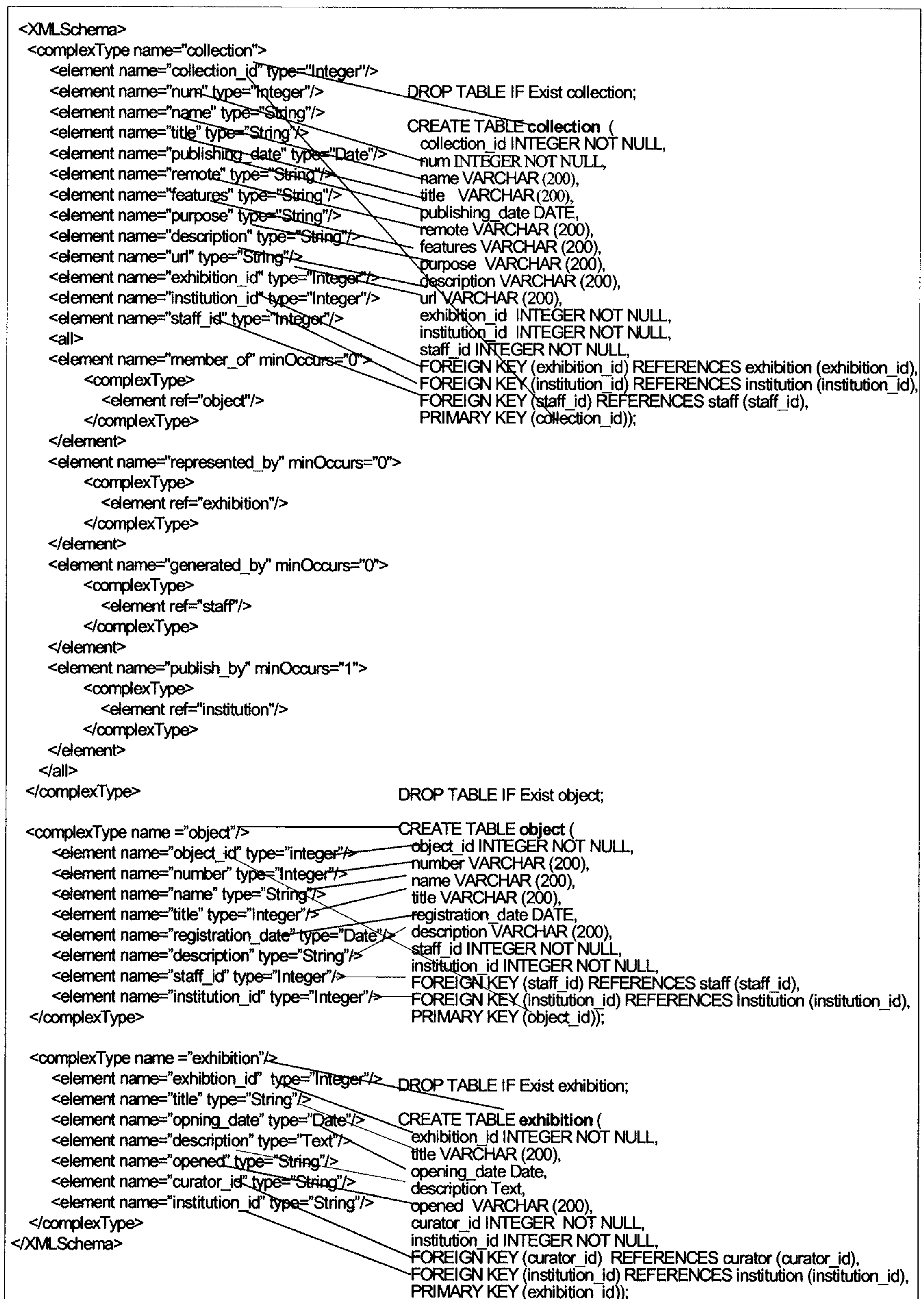
- For the single-valued attribute of that element type, and for each single-occurring simple child, create a column in that table.

- If XMLSchema has data type information, then set the data type of the column to the corresponding type such as VARCHAR, DATE and INTEGER as shown in our example. If the child node type or attribute is optional, make the column nullable.

- For each multi-valued attribute and for each multiply occurring simple child node, create a separate table to store values, linked to the parent node table through the parent table's primary key, as shown in the example.

- For each complex child node or element, link the parent element type's table to the child node type's table with the parent table's primary key.

Figure 5.8: Diagram for Mapping XML Schema to SQL Schema



5.2.1.2 Testing the algorithm

Figure 5.9 shows the SQL Schema of the initial database before we generate and use the generator (XSL stylesheet). The algorithm shown in Figure 5.7 is designed for extending the generation of the initial SQL schema. So we will apply the algorithm and see how we can extend the SQL schema shown in Figure 5.9.

To illustrate how the algorithm shown in Figure 5.7 works, let us walk through the algorithm, since the input of the algorithm is a fragment of our XML Schema containing information for the museum objects such as collection, object, objectInformation, exhibition and institution shown in Figure 5.11, and the generated XSL stylesheet (GenericSQL.xsl) shown in Figure 5.10. If the inputs of the algorithm exist, then we parse the XMLSchema, and while the parsing is executed (line 2.1-2.2 in Figure 5.7) the DOM tree will build dynamically. Then we perform the XSL stylesheet (GenericSQL.xsl) shown in Figure 5.10. If the transformation is done (line 2.2.2 in Figure 5.7), the algorithm is invoked with the root node of the XMLSchema as the complexType name (collection, object, objectInformation, exhibition, institution and staff in our example) then the XSL stylesheet templates will match the complexType node in DOM tree (i.e., <xsl:template match="collection">). For each complexType name will create a separate SQL table as a table name (line 2.2.2.3 in Figure 5.7). To generate the primary key take the table name followed by the string "_id" by using the XSL stylesheet template (2.2.2.3.1 in Figure 5.7). Also to generate the foreign key take the table name followed by the string "_id" by using the XSL stylesheet template (2.2.2.3.2 in Figure 5.7). If the complexType node has parent/child nodes (line 2.2.2.3.3 in Figure 5.7), the XSL stylesheet templates walk through DOM tree nodes, match the child node and map the child node to the corresponding column name in the SQL table (line 2.2.2.3.3.1-2.2.2.3.3.2 in Figure 5.7). Next, the XSL stylesheet iterates through the DOM tree and maps the data types to corresponding types in SQL schema such as VARCHAR, DATE and INTEGER. The XSL stylesheet applies the matched template for the child nodes of the DOM and maps them as column names in the SQL table (line 2.2.2.3.3.3 in Figure 5.7). The result is returned to all templates, generating SQL tables (SQL schema line 2.2.2.4 in Figure 5.7). At the end add SQL CREATE TABLE to the start of the generated SQL Schema.

As a result Figure 5.12 shows the extension of the SQL Schema after we used the generated XSL stylesheet; it also shows the generation of multiple Data Definition Language (DDL) SQL tables related via foreign keys.

The first SQL table names a collection having many column names such as collection_id, num, title, name, publishing_date, remote, features, purpose, description and url. While the exhibition_id, institution_id and staff_id are the foreign keys for the collection table.
The second DDL SQL table similarly defines an object table with column names, data types and foreign keys such as staff_id and institution_id. The third SQL table is objcetInformation, which has many column names and foreign keys such as object_id, institution_id and exhibition_id.
In addition, the SQL schema has SQL tables such as institution table, exhibition table and staff tables, each of which have column names, data types and foreign keys.

Figure 5.13 shows the execution of the Java class with the relational schema and the generated SQL schema as output. The source code for the implemented algorithm for generating an SQL schema can be found in Appendix B.

To conclude, we can generate automatically any SQL schema needed by using the generated XSL stylesheet (generator) shown in Figure 5.10. Although the next example just shows how the generator works, it should be emphasised that the XSL stylesheet is generic, which means the generator is valid for generating any SQL schema from any XML Schema. Furthermore, as we mentioned earlier, this algorithm is the technological solution to the problem of SQL schema generation.

Figure 5.9: SQL Schema for initial database

```
DROP TABLE IF Exist collection;
CREATE TABLE collection (collection_id INTEGER NOT NULL,
        num INTEGER NOT NULL, name VARCHAR (200), title  VARCHAR (200),
        publishing_date DATE, remote VARCHAR (200),
        features VARCHAR (200), purpose  VARCHAR (200),
        description VARCHAR (200), url VARCHAR (200), exhibition_id INTEGER NOT
        NULL, institution_id INTEGER NOT NULL, staff_id INTEGER NOT NULL,
        FOREIGN KEY (exhibition_id) REFERENCES exhibition (exhibition_id),
        FOREIGN KEY (institution_id) REFERENCES institution (institution_id),
        FOREIGN KEY (staff_id) REFERENCES staff (staff_id),
        PRIMARY KEY (collection_id));
```

Figure 5.10: Generic XSL stylesheet (GenericSQL.xsl) to generate SQL Schema

```
<?xml version="1.0" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/
    1999/XSL/Transform">
  <xsl:template match="complexType">
    DROP TABLE IF EXISTS <xsl:value-of select="@name" />;
    CREATE TABLE <xsl:value-of select="@name" />
    (<xsl:value-of select="@name" />_id INTEGER NOT NULL,
    <xsl:apply-templates select="element" mode="@name" />
    PRIMARY KEY (<xsl:value-of select="@name" />_id) )
    (<xsl:apply-templates select="element" mode="@name" />
    FOREIGN KEY (<xsl:value-of select="@name" />_id) REFERENCES
      (<xsl:value-of select="@name))
  </xsl:template>
  <xsl:template match="element" mode="@name">
    <xsl:value-of select="@name" />
    <xsl:apply-templates select="@type" mode="schematype" />
  </xsl:template>
  <xsl:template match="@type" mode="schematype">
    <xsl:variable name="type" select="." />
    <xsl:choose>
      <xsl:when test="$type='String'">VARCHAR ( 200)</xsl:when>
      <xsl:when test="$type='Date'">DATE </xsl:when>
      <xsl:when test="$type='Text'">TEXT</xsl:when>
      <xsl:when test="$type='Integer'">INTEGER NOT
          NULL</xsl:when>
      <xsl:when test="$type='Image'">IMAGE</xsl:when>
      <xsl:when test="$type='Url'">URL</xsl:when>
      <xsl:when test="$type='Char'">CHAR (1)</xsl:when>
      <xsl:when test="$type='Sex'">SEX</xsl:when>
      <xsl:when test="$type='Country'">COUNTRY</xsl:when>
      <xsl:when
          test="$type='ReferenceType'">ReferenceType</xsl:when>
      <xsl:when
          test="$type='CollectionMethod'">CollectionMethod</xsl:when>
    </xsl:choose>,
  </xsl:template>
</xsl:stylesheet>
```

Note: SEX and COUNTRY are user-defined types.

Figure 5.11: A Fragment of our XMLSchema

```
<XMLSchema>
 <element name="collection" type="collection"/>
 <complexType name="collection">
   <element name="collection_id" type="Integer"/>
   <element name="num" type="Integer"/>
   <element name="name" type="String"/>
   <element name="title" type="String"/>
   <element name="publishing_Date" type="Date"/>
   <element name="remote" type="String"/>
   <element name="features" type="String"/>
   <element name="purpose" type="String"/>
   <element name="description" type="String"/>
   <element name="url" type="String"/>
   <element name="exhibition_id" type="Integer"/>
   <element name="institution_id" type="Integer"/>
   <element name="staff_id" type="Integer"/>
   <all>
   <element name="member_of" minOccurs="0">
        <complexType>
          <element ref="object"/>
        </complexType>
   </element>
   <element name="represented_by" minOccurs="0">
        <complexType>
          <element ref="exhibition"/>
        </complexType>
   </element>
   <element name="generated_by" minOccurs="0">
        <complexType>
          <element ref="staff"/>
        </complexType>
   </element>
   <element name="publish_by" minOccurs="1">
        <complexType>
          <element ref="Institution"/>
        </complexType>
   </element>
  </all>
 </complexType>
 <element name="object" type="object"/>
   <complexType name="object">
      <element name="object_id" type="Integer"/>
      <element name="number" type="Integer"/>
      <element name="name" type="String"/>
      <element name="title" type="String"/>
      <element name="registration_date" type="Date"/>
      <element name="description" type="String"/>
      <element name="staff_id" type="Integer"/>
      <element name="institution_id" type="Integer"/>
      <all>
        <complexType>
          <choice minOccurs ="1" maxOccurs="unbounded">
            <element ref="objectInformation"/>
          </choice>
        </complexType>
        <element name="owned_by" minOccurs="1">
           <complexType>
              <element ref="Institution"/>
           </complexType>
        </element>
      </all>
   </complexType>
   <complexType name= "objectInformation">
       <element name="objectInformation_id" type="Integer"/>
       <element name="description_date" type="Date"/>
       <element name="description" type="String"/>
       <element name="object_id" type="Integer"/>
       <element name="Institution_id" type="Integer"/>
       <element name="collection_id" type="Integer"/>
       <element name="staff_id" type="Integer"/>
       <element name="expert_id" type="Integer"/>
     <all>
       <complexType>
         <choice minOccurs ="1" maxOccurs="unbounded">
           <element ref="object"/>
         </choice>
       </complexType>
       <element name="owned_by" minOccurs="1">
         <complexType>
            <element ref="institution"/>
         </complexType>
       </element>
       <element name="approved_by" minOccurs="1">
         <complexType>
           <element ref="staff"/>
         </complexType>
       </element>
       <element name="availabled_in" minOccurs="1">
         <complexType>
           <element ref="address"/>
         </complexType>
       </element>
     </all>
   </complexType>
   <complexType name= "exhibition">
     <element name="exhibition_id" type = "Integer"/>
     <element name="title" type = "String"/>
     <element name="opening_date" type = "Date"/>
     <element name="describtion" type  = "Text"/>
     <element name="opened" type = "String"/>
     <element name="curator_id" type = "Integer"/>
     <element name="institution_id" type = "String"/>
     <all>
      <element name="exhibition.hosted_by" minOccurs="0">
         <complexType>
            <choice minOccurs="0" maxOccurs="unbounded">
               <element ref="institution"/>
            </choice>
          </complexType>
      </element>
      <element name="arrange" maxOccurs="1">
          <complexType>
            <choice minOccurs="1" maxOccurs="unbounded">
                <element ref="curator"/>
             </choice>
          </complexType>
      </element>
     </all>
   </complexType>
   <complexType name="institution">
     <element name="institution_id" type="Integer"/>
     <element name="name" type="String"/>
     <element name="institution_type" type="String"/>
     <element name="country" type="country"/>
     <element name="description" type="String"/>
     <element name="url" type="String"/>
     <all>
       <element name="hosted_by" minOccurs="1">
            <complexType>
               <element ref="exhibition"/>
            </complexType>
       </element>
     </all>
   </complexType>
</XMLSchema>
```

5.2.1.2.1 Output of the algorithm

As we mentioned the output of the algorithm is an SQL schema generated by implementing the algorithms shown in Figure 5.12 and Figure 5.13 respectively.

Figure 5.12: SQL Schema generated after using XSL Stylesheet (generator)

```
DROP TABLE IF Exist collection;
CREATE TABLE collection (collection_id INTEGER NOT NULL,
        num INTEGER NOT NULL, name VARCHAR (200), title VARCHAR (200),
        publishing_date DATE, remote VARCHAR (200),
        features VARCHAR (200), purpose VARCHAR (200),
        description VARCHAR (200), url VARCHAR (200),
        exhibition_id INTEGER NOT NULL,
        institution_id INTEGER NOT NULL,
        staff_id INTEGER NOT NULL,
        FOREIGN KEY (exhibition_id) REFERENCES exhibition (exhibition_id),
        FOREIGN KEY (institution_id) REFERENCES institution (institution_id),
        FOREIGN KEY (staff_id) REFERENCES staff (staff_id),
        PRIMARY KEY (collection_id));

DROP TABLE IF Exist object;
CREATE TABLE object (object_id INTEGER NOT NULL, number INTEGER NOT NULL,
        name VARCHAR (200), title VARCHAR (200),
        registration_date DATE, description VARCHAR (200),
        staff_id INTEGER NOT NULL, institution_id INTEGER NOT NULL,
        FOREIGN KEY (staff_id) REFERENCES staff (staff_id),
        FOREIGN KEY (institution_id) REFERENCES institution (institution_id),
        PRIMARY KEY (object_id));

DROP TABLE IF Exist objectInformation;
CREATE TABLE objectinformation (objectinformation_id INTEGER NOT NULL,
        description_date DATE, description VARCHAR (200), object_id INTEGER NOT
        NULL, institution_id INTEGER NOT NULL, collection_id INTEGER NOT NULL,
        staff_id INTEGER NOT NULL, expert_id INTEGER NOT NULL,
        FOREIGN KEY (object_id) REFERENCES object (object_id),
        FOREIGN KEY (institution_id) REFERENCES institution (institution_id),
        FOREIGN KEY (collection_id) REFERENCES collection (collection_id),
        FOREIGN KEY (staff_id) REFERENCES staff (staff_id),
        FOREIGN KEY (expert_id) REFERENCES expert (expert_id),
        PRIMARY KEY (objectinformation_id));

DROP TABLE IF Exist exhibition;
CREATE TABLE exhibition (exhibition_id INTEGER NOT NULL,
        title VARCHAR (200) NOT NULL, opening_date DATE,
        description VARCHAR (200), opened VARCHAR (200),
        curator_id INTEGER NOT NULL,
        institution_id INTEGER NOT NULL,
        FOREIGN KEY (curator_id) REFERENCES curator (curator_id),
        FOREIGN KEY (institution_id) REFERENCES institution (institution_id),
        PRIMARY KEY (exhibition_id));

DROP TABLE IF Exist institution;
CREATE TABLE institution (institution_id INTEGER NOT NULL,
        name VARCHAR (200), institution_type VARCHAR (200),
        country VARCHAR (200), description VARCHAR (200), url VARCAHR (200),
        PRIMARY KEY (institution_id));
```

Figure 5.13: Execution for Generation of SQL schema for Relational Schema with GenericSQL.xsl

```
Command Prompt
C:\Tomcat\webapps\xml\WEB-INF\classes>c:\j2sdk1.4.0\bin\java -classpath c:\oracle\ora92\jdbc\lib
sses12.jar;. GenSQLSchema GenericSQL.xsl XMLSchema.xsd




DROP TABLE IF EXISTS  collection;
CREATE TABLE  collection(
collection_id INTEGER NOT NULL,
num INTEGER NOT NULL,
name VARCHAR (200),
title VARCHAR (200),
publishing_date DATE ,
remote VARCHAR (200),
features VARCHAR (200),
purpose VARCHAR (200),
description VARCHAR (200),
url URL,
exhibition_id INTEGER NOT NULL,
institution_id INTEGER NOT NULL,
staff_id INTEGER NOT NULL,
FOREIGN KEY (exhibition_id) REFERENCES exhibition (exhibition_id),
FOREIGN KEY (institution_id) REFERENCES institution (institution_id),
FOREIGN KEY (staff_id) REFERENCES staff (staff_id),
PRIMARY KEY( collection_id ));




DROP TABLE IF EXISTS  object;
CREATE TABLE  object(
object_id INTEGER NOT NULL,
number INTEGER NOT NULL,
name VARCHAR (200),
title VARCHAR (200),
registration_date DATE ,
description VARCHAR (200),
staff_id INTEGER NOT NULL,
institution_id INTEGER NOT NULL,
FOREIGN KEY (staff_id) REFERENCES staff(staff_id),
FOREIGN KEY (institution_id) REFERENCES institution (institution_id),
```

Start | Command Prompt | 12:20

5.2.2 XSL stylesheet generation

As we mentioned before, it is possible to code by hand an XSL stylesheet that validates an XML document against some or all constraints of an XML schema. But in this section we introduce an algorithm as shown in Figure 5.14 for generating an XSL stylesheet automatically from an XMLSchema parse file (DOM). Document Object Model (DOM) is an Application-Programming Interface (API) for XML and HTML documents, which defines the logical structure of the document and the way a document is accessed and manipulated. Also, the DOM tree is composed of nodes, each of which represents a parsed document. In other words, this algorithm is the technological solution to the problem of generating an XSL stylesheet automatically by transforming an XMLSchema through an XSL stylesheet. The result is a generic XSL stylesheet providing the mechanism to transform and manipulate the XML data.

In the following we will generate stylesheets for transforming XML data to SQL and HTML data.

- An XSL stylesheet can be used to transform an XML document into other formats such as an SQL statement, which can then be used to store the XML data in a relational database.
- An XSL stylesheet can be used to transform an XML document to an HTML document to present the XML data. The input of this algorithm is also XMLSchema and another XSL stylesheet.

All the XSL stylesheets generated here will be used in the online transformation, as can be seen in the following subsections. In other words the XSL stylesheets will be used to integrate the offline and the online components.

Figure 5.14: An Algorithm for generating XSL from XMLSchema

// Algorithm to generate XSL from XMLSchema
// Input: XMLSchema, XSL stylesheet
// Output: XSL stylesheet

1. start
2. the input arguments (XMLSchema, XSL stylesheet)
3. if the arguments (XMLSchema and XSL stylesheet) exist

    3.1 Build DOM and parse it

    3.2 if parsing XMLSchema is done, then DOM will build dynamically

        3.2.1 perform XSL stylesheet (each XSL stylesheet contains templates and commands to select and manipulate structure of data)

        3.2.2 if transformation is done

            3.2.2.1 invoke the root node of DOM tree

            3.2.2.2 compare the root node with XSL template, if it matches the root node, maps the root node as a new template to the output

            3.2.2.3 If the root node has parent/child nodes

                3.2.2.3.1 the XSL walks through DOM tree and pulls nodes from DOM tree and places them with formatting as a new template to output

                3.2.2.3.2 compare and match complexType nodes of the DOM tree with the XSL template, for each complexType node create a new XSL template

                3.2.2.3.3 map the child nodes to the template as column names, and also the data types of DOM mapped as values to the column names

                3.2.2.3.4 iterate through the DOM tree nodes and set the keyword VALUES to the output as a new template in the XSL stylesheet

                3.2.2.3.5 insert the required statement and then return all templates (new XSL stylesheet generated)

            3.2.2.4 set a message "no parent/child node" and terminate

        3.2.3 report transformation errors and terminate

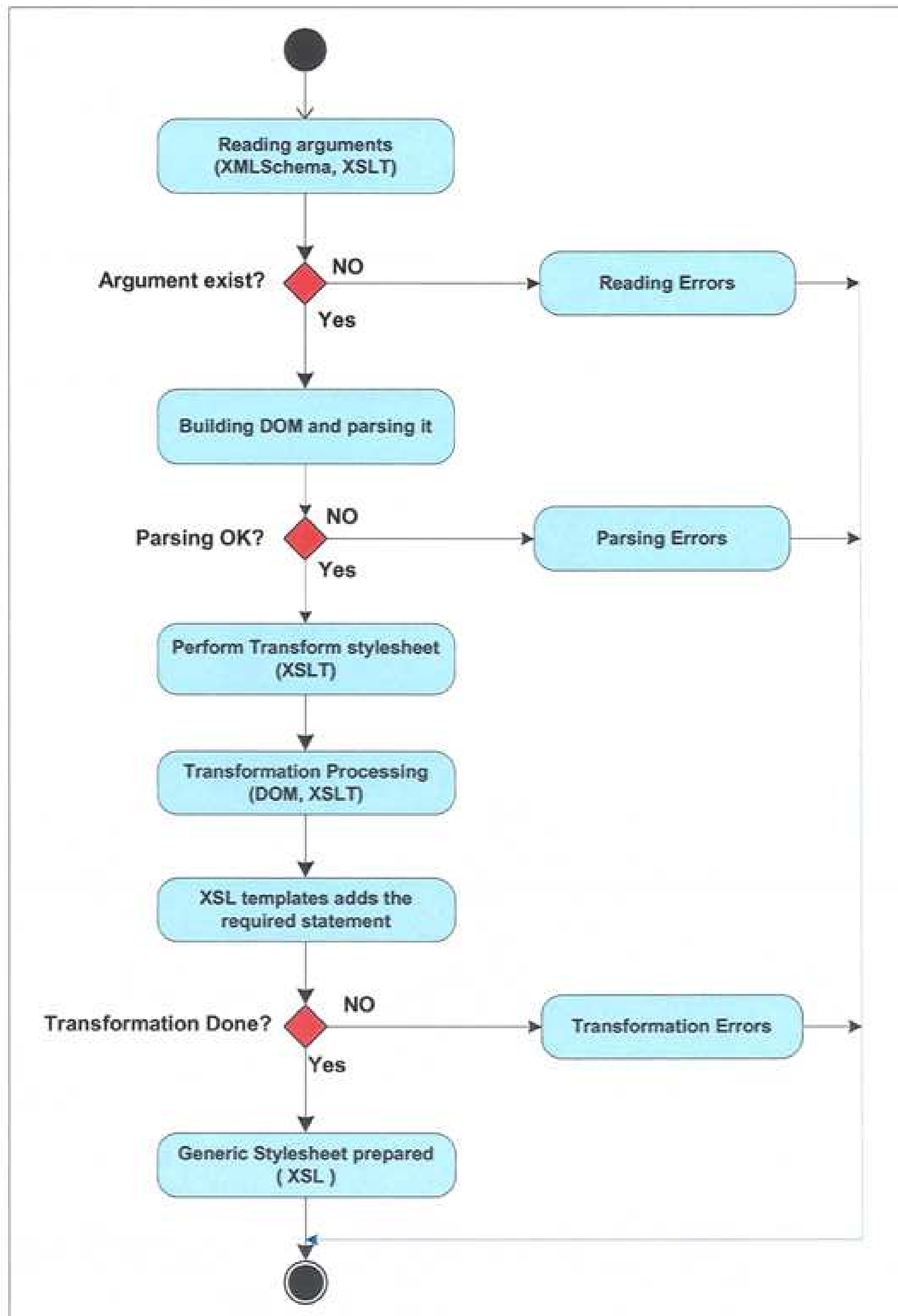
    3.3 report parsing errors and terminate

4. report reading errors and terminate
5. terminate

5.2.2.1 How the algorithm works

The diagram in Figure 5.15 shows how the algorithm works and the result will be an XSL stylesheet generated from the XMLSchema.

Figure 5.15: Diagram for generating XSL Stylesheet from XMLSchema



5.2.2.2 Testing the algorithm

To test the algorithm shown in Figure 5.14 we will use the input parameters of the algorithm (XSL stylesheet and the XMLSchema). Let us walk through the algorithm by using the fragment of XMLSchema shown in Figure 5.11 as input data for testing the algorithm. The algorithm is invoked with its start point by checking that the arguments exist (inputs of the algorithm). If they exist, then we parse and the DOM tree will build dynamically "Building DOM" (line 3.2 in Figure 5.14). Then we perform the XSL stylesheet for each template in the XSL stylesheet, match a node in the DOM tree and invoke the transformation from the root node of the DOM tree (line 3.2.1-3.2.2.1 in Figure 5.14). The XSL stylesheet template walks through the DOM tree, pulls out the child nodes of the DOM tree and places them with formatting to the output as generated templates (lines 3.2.2.3.1 in Figure 5.14). Next, the XSL templates will compare and match the child nodes of DOM tree and generate them as a new template for the output (line 3.2.2.3.2 in Figure 5.14). The XSL templates add the required statement, following them by the complexType name to the new XSL stylesheet. The XSL stylesheet templates iterate through the DOM tree nodes and set the keyword VALUES as new template to the output (line 2.2.2.3.3 in Figure 5.14). The execution returns all new templates generated as the output (line 3.2.2.3.4 in Figure 5.14). The output of the algorithm will be a generic XSL stylesheet generated from the existing XMLSchema parse file automatically (line 3.2.2.3.4 in Figure 5.14) as shown in Figure 5.16.

Furthermore, the resulting XSL stylesheet (generator) can then be used to generate an SQL statement from an XML document or for transforming any XML document to an HTML document. In the following sections we will introduce some examples to show how the generator works.

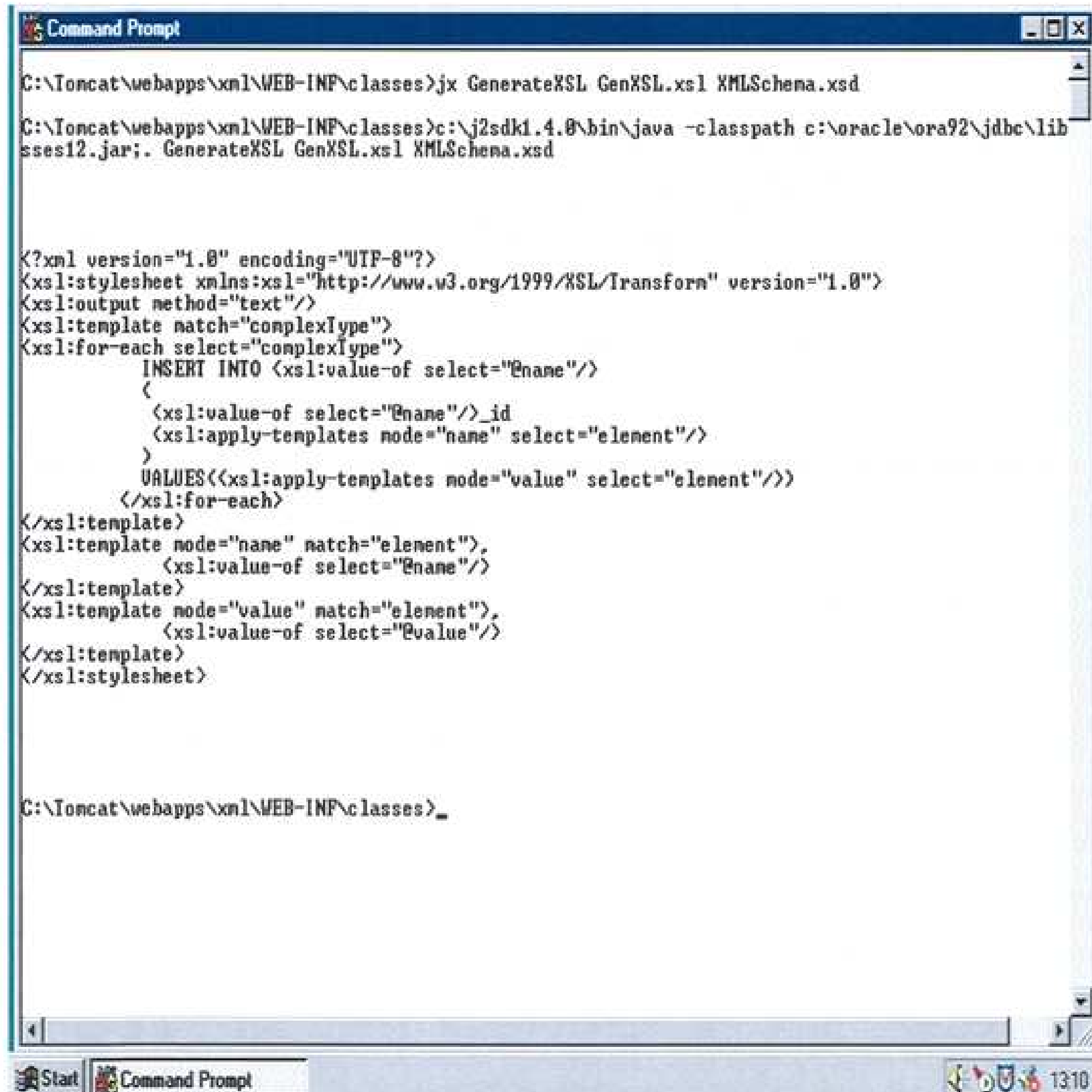
5.2.2.2.1 Output 1 of the algorithm

As already mentioned, the output of the algorithm is a generic XSL stylesheet for transforming an XML to SQL statement and also a generic XSL stylesheet for transforming XML to HTML. Figures 5.16 and 5.18 show respectively the XSL stylesheets that are generated as the output, from running the algorithm with XMLSchema through the XSL stylesheet. Also Figure 5.17 shows the generation of an XSL stylesheet from execution of the Java class with the XMLSchema. The source code for the implemented generating XSL stylesheet algorithm can be found in Appendix B.

Figure 5.16: Generic XSL stylesheet (GenXSL.xsl) generated using the algorithm in Figure 5.14

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      version="1.0">
    <xsl:output method="text" />
    <xsl:template match="complexType">
      <xsl:for-each select="complexType">
        INSERT INTO <xsl:value-of select="@name" /> (
        <xsl:value-of select="@name" />_id
        <xsl:apply-templates mode="name" select="element" />
        ) VALUES (
        <xsl:apply-templates mode="value" select="element" /> )
      </xsl:for-each>
    </xsl:template>
    <xsl:template match="element" mode="name">,
      <xsl:value-of select="@name" />
    </xsl:template>
    <xsl:template match="element" mode="value">,
      <xsl:value-of select="@value" />
    </xsl:template>
  </xsl:stylesheet>
```

Figure 5.17: Execution for generation of XSL stylesheet using XMLSchema and GenXSL.xsl



5.2.2.2.2 Output 2 of the algorithm (generation an XSL for transforming XML to HTML)

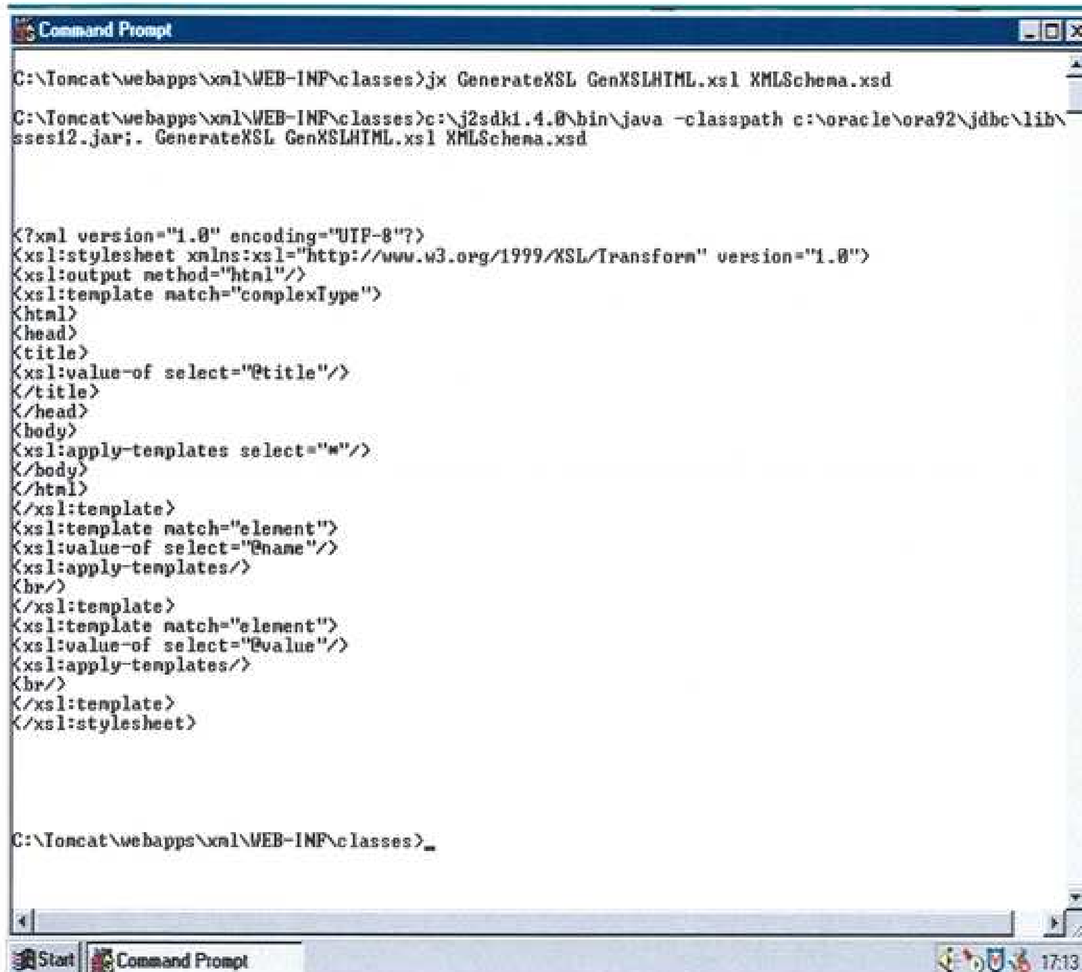
As mentioned in section 5.2.2, this stylesheet is also generated automatically from XMLSchema through an XSL stylesheet by using the Java class shown in Figure 5.19. Figure 5.18 shows an XSL stylesheet for transforming an XML document to an HTML document. The result is a generic XSL stylesheet that can be used for transforming any XML document to HTML format. Here we use the algorithm shown in Figure 5.14 with a different input to generate the XSL stylesheet, as shown in Figure 5.18.

In other words, the inputs of the algorithm are an XML schema and XSL stylesheet to generate an XSL stylesheet for transforming XML to HTML. In the following sections we will test the above generators respectively for transforming an XML document to SQL statement and for XML to an HTML document. The source code for the implemented transforming algorithm can be found in Appendix B.

Figure 5.18: Generic XSL Stylesheet for transforming XML to HTML Document

```
<? xml version="1.0"? >
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="complexType">
     <html>
        <head>
           <title>
              <xsl:value-of select="@title" />
           </title>
        </head>
        <body>
           <xsl:apply-templates select="*" />
        </body>
     </html>
   </xsl:template>
   <xsl:template match="element">
      <xsl:value-of select="@name" />
      <xsl:apply-templates />
      <br />
   </xsl:template>
  <xsl:template match="element">
      <xsl:value-of select="@value" />
      <xsl:apply-templates />
      <br />
   </xsl:template>
</xsl:stylesheet>
```

Figure 5.19: Execution for the Generation of an XSL Stylesheet of XMLSchema for transforming XML to HTML



5.2.2.3 Testing the generator (XSL)

There are several examples in the following sections for testing the generated XSL stylesheets. The examples are given in section 5.2.2.3.1 for transforming an XML data to SQL strings, to be stored in the database [46]. The examples are given in section 5.2.2.3.2 for transforming an XML document into an HTML document.

5.2.2.3.1 Transforming an XML document to SQL statement and storing in the database

To illustrate how the generic XSL stylesheet shown in Figure 5.16 works, consider the following examples shown in Figures 5.20, 5.22 and 5.23 respectively for transforming an XML document to SQL and producing an SQL string. Let us walk through these examples by using the inputs (XSL stylesheet and XML document).

- **Example 1 for transforming XML → SQL**

Example 1 has the XML document and XSL stylesheet as inputs. In the XML document there is a topmost element <collection> as the root element of the XML document that contains the elements (child nodes) such as collection_id, num, title, publishing_date, remote, features, description, url, exhibition_id, institution_id and staff_id; also the XSL stylesheet has templates. The XSL templates will match the root element of the XML document and map this root node to the SQL table name as a table name (collection) and add INSERT INTO to the start of the table. Moreover, the XSL stylesheet templates match the child elements and attribute elements in the XML document and map them as column names to the SQL table such as collection_id, num, title, publishing_date, remote, features, description, url, exhibition_id, institution_id and staff_id as shown in Figure 5.20.

Furthermore, the data values are mapped onto SQL tables as data value to the corresponding column names such as (10,20, 'group1', 'collector', '08.01.03', 'yes', 'exhibition', 'research', 'first exhibition', 'http//:www.virtualexh.com', 100,200,300). When the XSL stylesheet is applied to the XML document, it produces a SQL statement as shown in Figure 5.20.

In addition, Figure 5.21 shows the result of the execution of a Java servlet class for transforming XML to SQL by using the generated XSL stylesheet (GenXSL.xsl). The result will be an SQL string, which can be inserted into a collection table of the database over JDBC [125].

Figure 5.20: Example 1 for transforming an XML document to SQL Statement

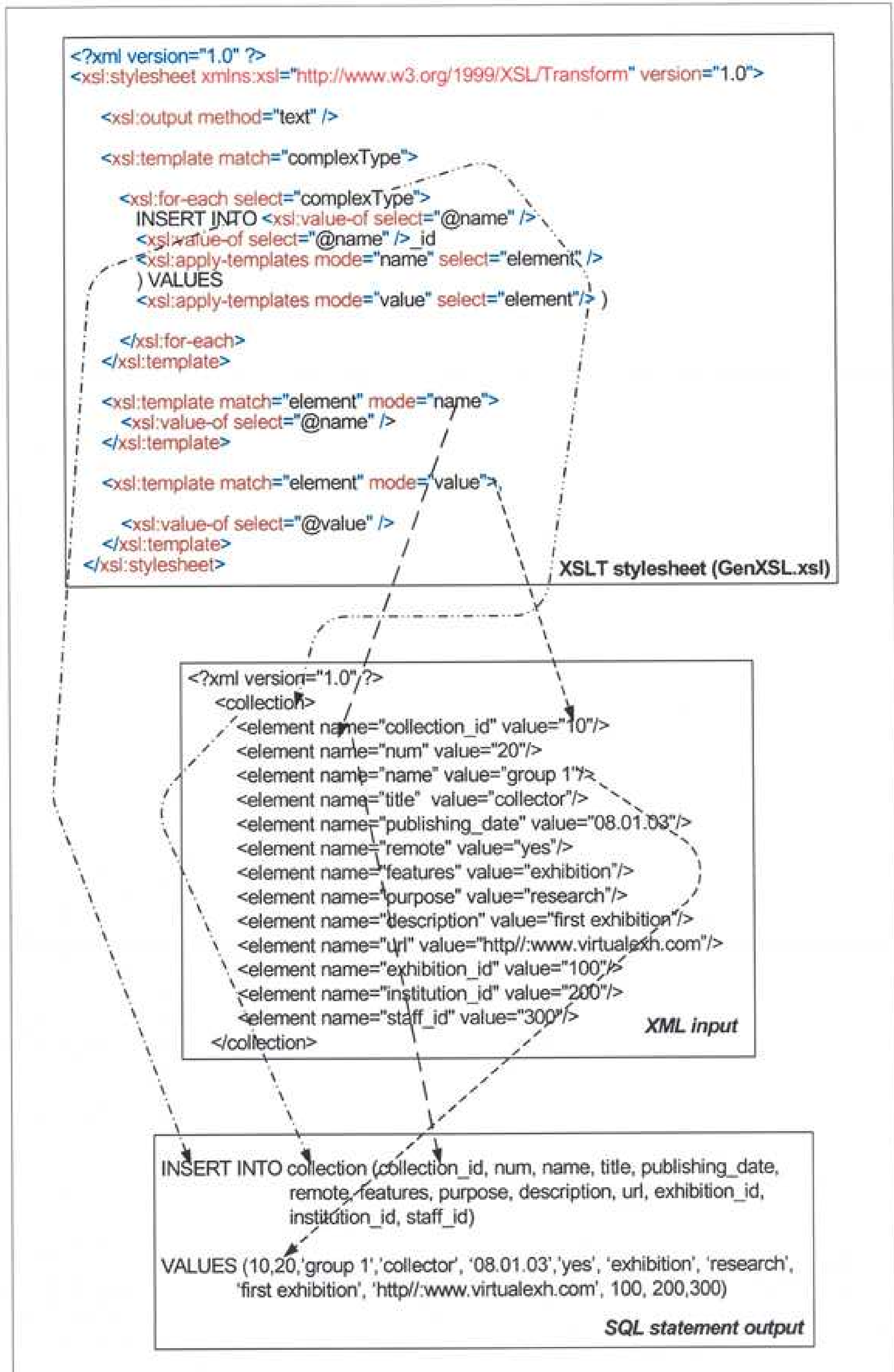
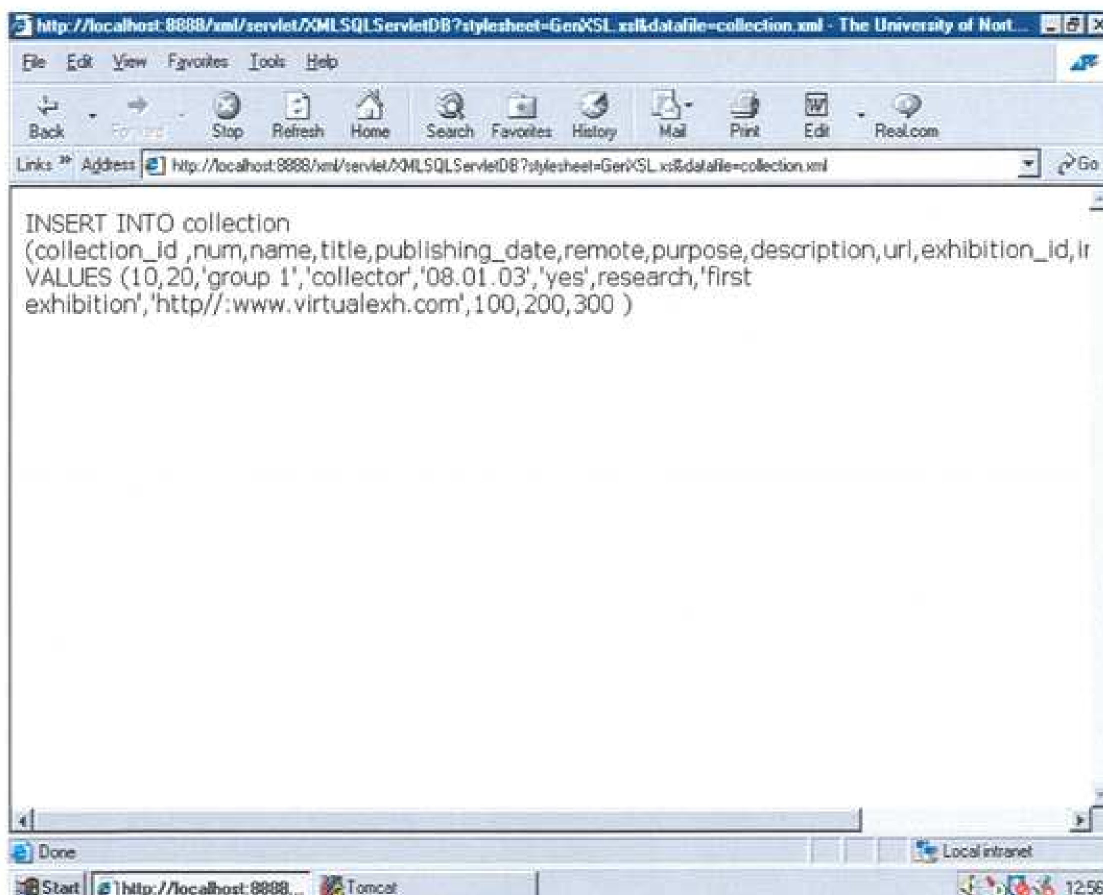


Figure 5.21: The Result of executing the Java Servlet Class (online) Transforming XML to SQL by using GenXSL.xsl



- **Example 2 for transforming XML → SQL**

Example 2 shows the mapping of XML data to SQL strings by using the XSL stylesheet (GenXSL.xsl). The XSL stylesheet templates will match the root element of the XML document and map the root node to the SQL table name, as institution, and the template will add the key word INSERT INTO before the table name. The XSL stylesheet templates match the child elements and attributes in the XML document and map them as column names to the SQL table such as institution_id, name, institution_type, country, description and url, as shown in Figure 5.22.

The XML data is also mapped to values in an SQL statement such as 45, 'British museum', 'private', 'Great Britain', 'yes', 'http//:www.virtualexh.com'. When the XSL stylesheet is applied to the XML document, it produces a SQL statement, as shown in Figure 5.22.

In addition, Figure 5.22a shows the result from execution of a Java servlet class for transforming XML to SQL by the generated XSL stylesheet (GenXSL.xsl) and the result of the transformation is stored into an institution table of database by using Java Database Connectivity (JDBC) [125].

Figure 5.22: Example 2 for transforming an XML document to SQL Statement

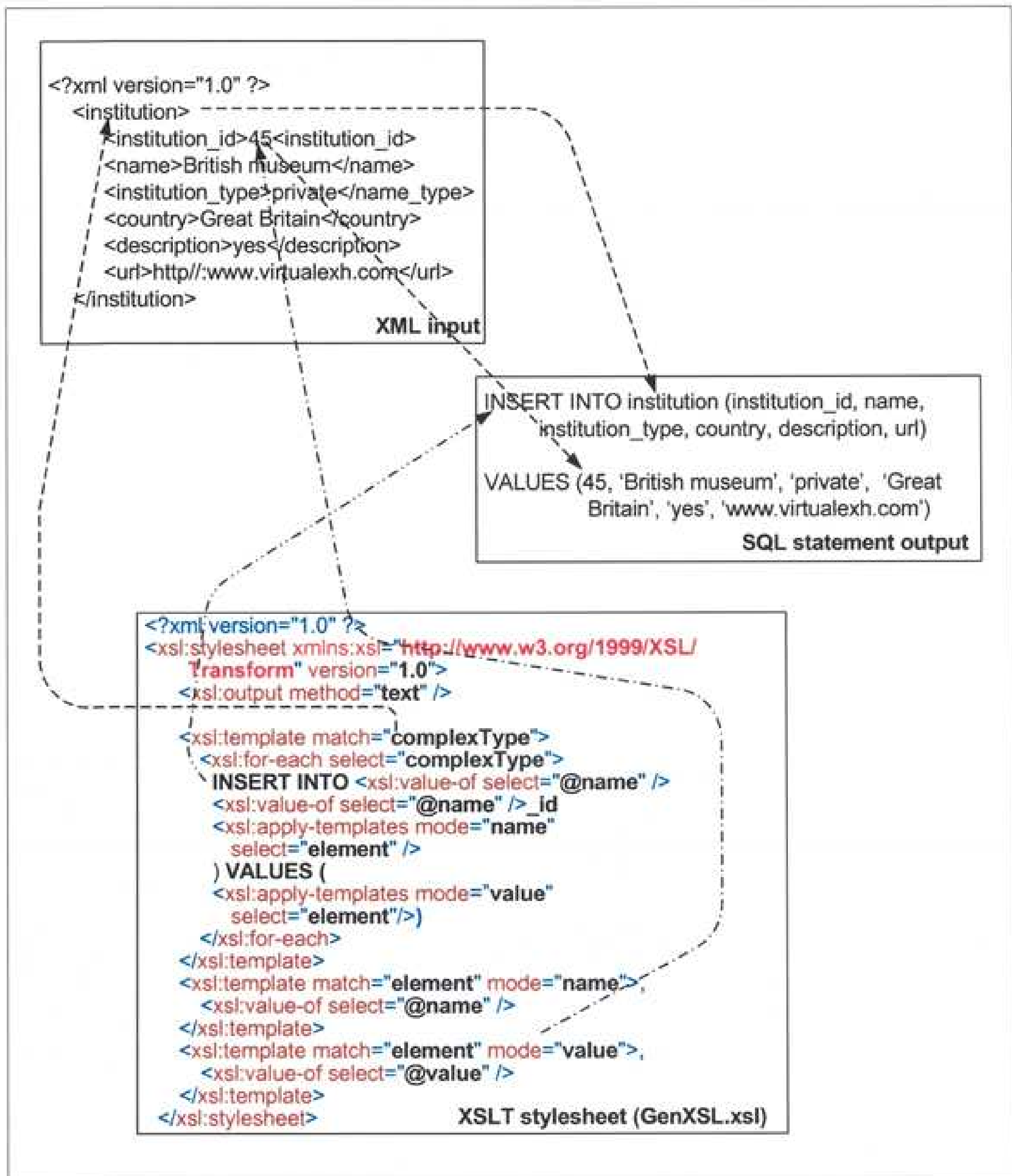
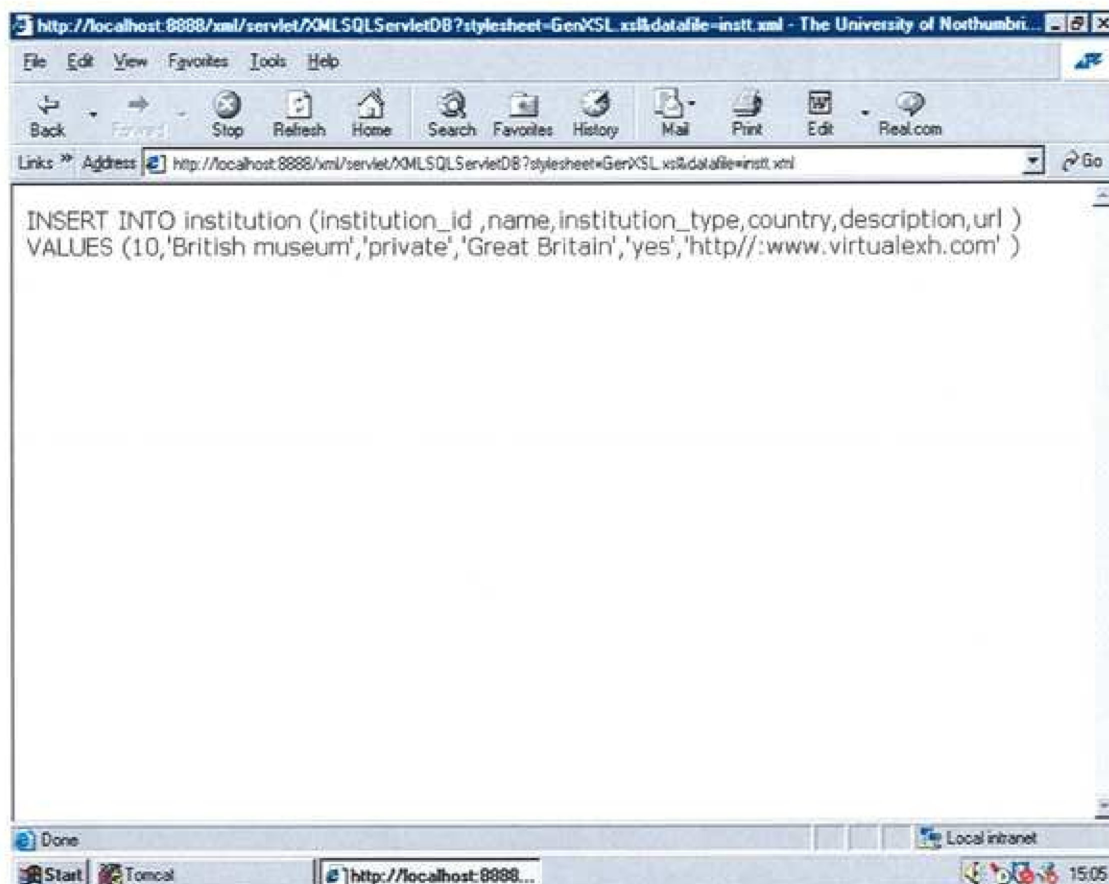


Figure 5.22a: The Result of executing the Java Servlet Class (online) for transforming XML to SQL by using GenXSL.xsl

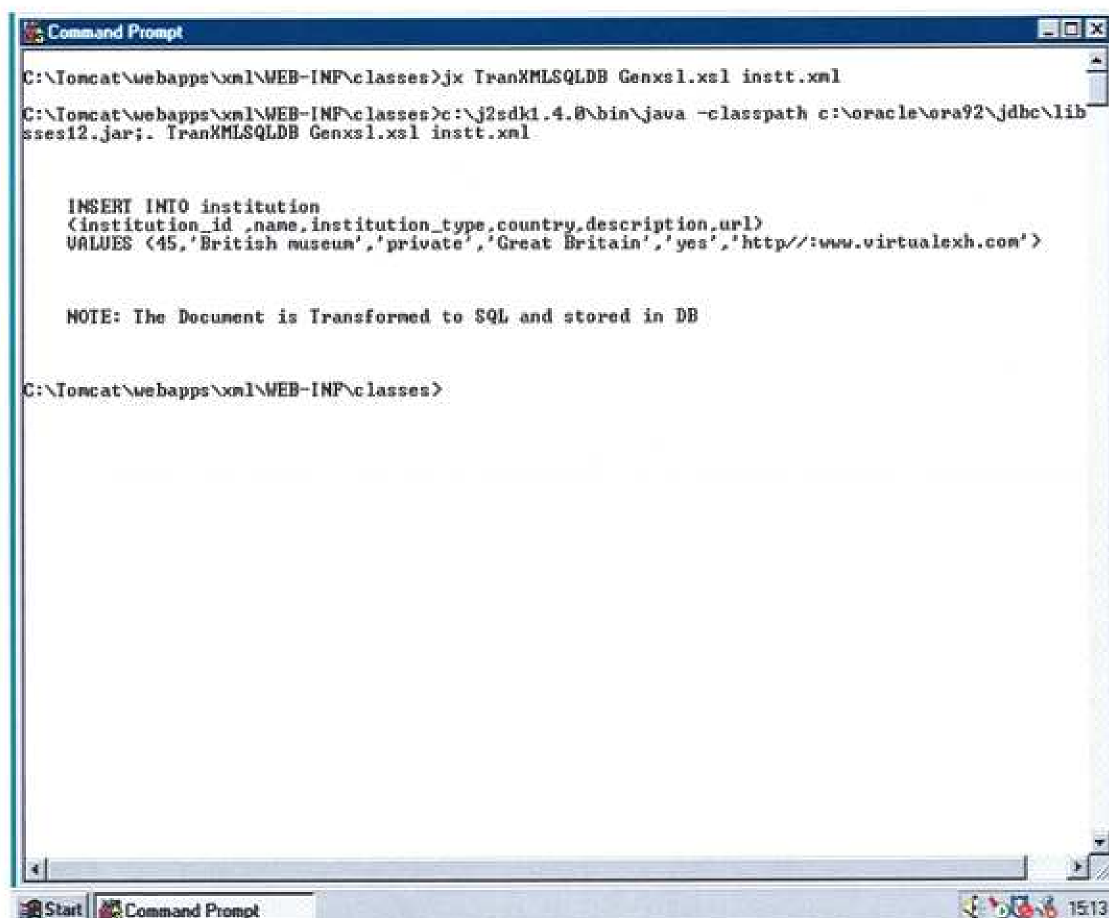


- **Example 3 for transforming XML → SQL**

Example 3 shows the mapping of XML data to SQL strings by using the XSL stylesheet (GenXSL.xsl). The output will be a sequence of SQL strings inserted into relational tables with JDBC and stored in a database.

Finally, based on the previous examples we can conclude that, by using the developed XSL stylesheet shown in Figure 5.19, we can transform the XML data and produce SQL statements as strings programmatically (Java) [99] and then store the result in a relational database with JDBC. The source code for the implemented transformation of XML data to an SQL statement can be found in Appendix B.

Figure 5.23: Example 3 for transforming an XML document to SQL (offline)



5.2.2.3.2 Transforming XML to an HTML document

Basically, HTML combines data and presentation in the data on the web and also HTML is the only language that is understood by all browsers. HTML provides a fixed set of predefined elements (or tags) that tell the browsers how to format and display information (or data) on the Web. Since XML combines only the data, we need to convert it into HTML, in order to display and present it on the Web. The following examples for testing the generated XSL stylesheet are shown in Figure 5.20, but as the stylesheet is a generic one that means we can use this stylesheet for transforming any XML document to an HTML document.

- **Example 1 for transforming XML → HTML**

Figure 5.24 illustrates the transforming of an XML document to HTML document by using the generated XSL stylesheet (GenXSLHTML.xsl). Let us walk through the example and see how the transformation works. The XSL stylesheet contains templates and commands to select and manipulate the structure of the data.

It operates by reading a stylesheet, which consists of one or more templates, then matching the templates as it visits the nodes of the XML document and pulls in data from an XML document and places it, with formatting, into the HTML document. In our example, the <html> tag and the <body> tag is written in the output. Also the complexType in XSL stylesheet matches the collection in the HTML document. The <xsl:apply-templates> templates in the body of HTML match all the child elements of the object1 in XML and transform them to the HTML document such as 100, 1234, 10, 12.12.98, 20, 12.12.98, 301 and 3239. Moreover, the </body> tag and the </html> tag are written to the output. Figure 5.24 shows the transformation of XML to HTML in more detail.

Furthermore, Figure 5.24a shows the result of the execution of a Java servlet class for transforming XML to HTML by using the generated XSL stylesheet (GenXSLHTML.xsl). The result of the transformation is displayed by a browser on the Tomcat server. The source code for the implemented algorithm transforming XML to HTML can be found in Appendix B.

Figure 5.24: Transforming XML to HTML by using the generated XSL stylesheet

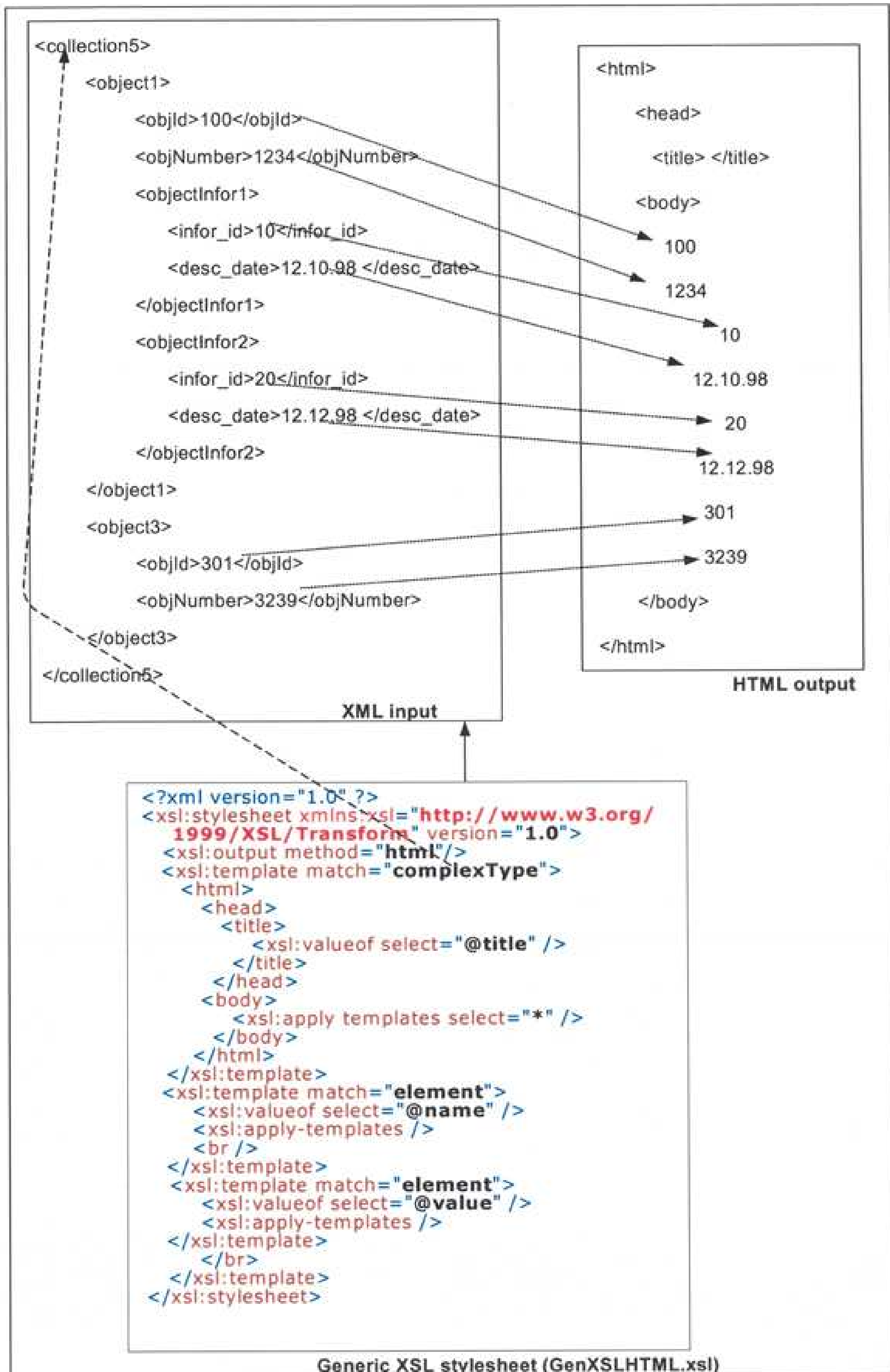
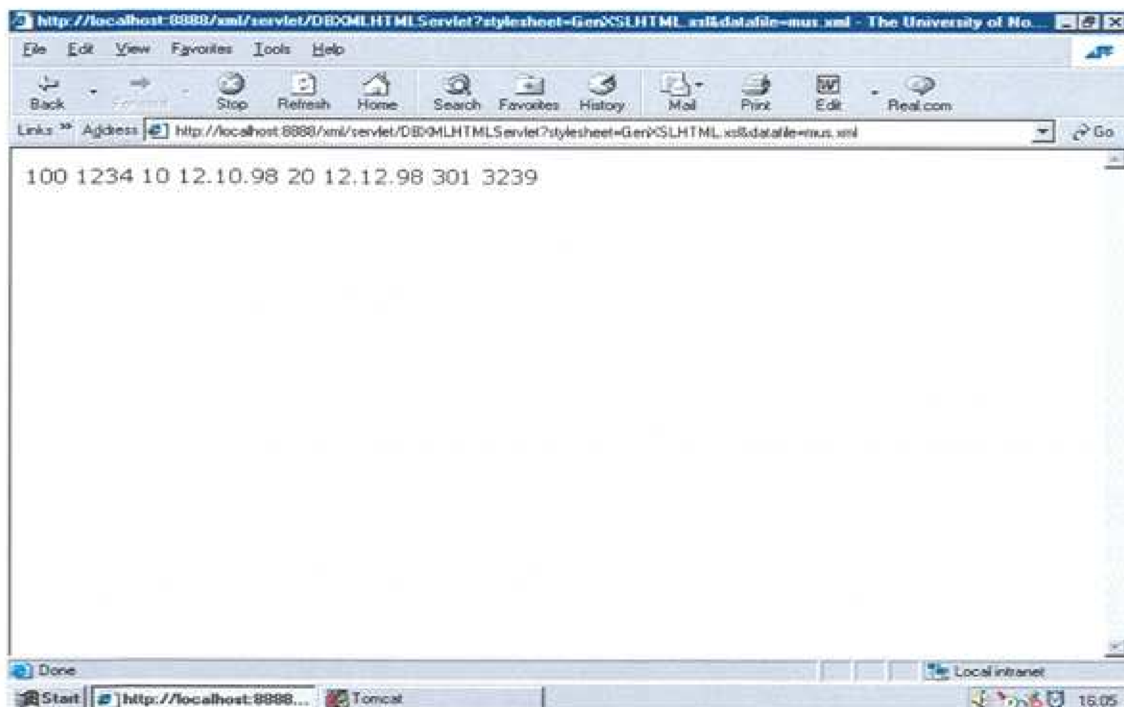


Figure 5.24a: The Result of executing the Java Servlet Class for transforming XML to HTML by using the GenXSLHTML.xsl



- **Example 2 for transforming XML → HTML**

Example 2 is also given for testing the generated XSL stylesheet shown in Figure 5.20. Let us follow the diagram shown in Figure 5.25 by using the XML document and the generated XSL stylesheet as input. The XSL stylesheet contains templates and commands to select and manipulate the structure of the data. It operates by reading a stylesheet, which consists of one or more templates, then matching the templates as it visits the nodes of the XML document and pulls data from an XML document and places it, with formatting, into an HTML document. As in our example, the start <html> and the <body> are written to the new output.

The <xsl:apply-templates> templates in the XSL stylesheet match all the child elements of the object1 in XML and transform all text element to the HTML document as 100, 1234, altarpiece, lasmeninas, 12.01.03, this object of the Archive. Object 2 such as 201, 2234, Rattus rattus, lasmeninas, 12.03.03, displayed in private exhibition as shown in Figure 5.25. What is more the </body> tag and the </html> tag is written out. Figure 5.25 shows the transformation of XML to HTML in more detail.

Figure 5.25a also shows the result from the execution of Java servlet class for transforming XML to HTML by the generated XSL stylesheet (GenXSLHTML.xsl) and the result of the transformation is displayed by a browser on the Tomcat server. The source code for the implementation of transforming XML to HTML can be found in Appendix B.

Figure 5.25: Transforming XML to HTML by using the generated XSL

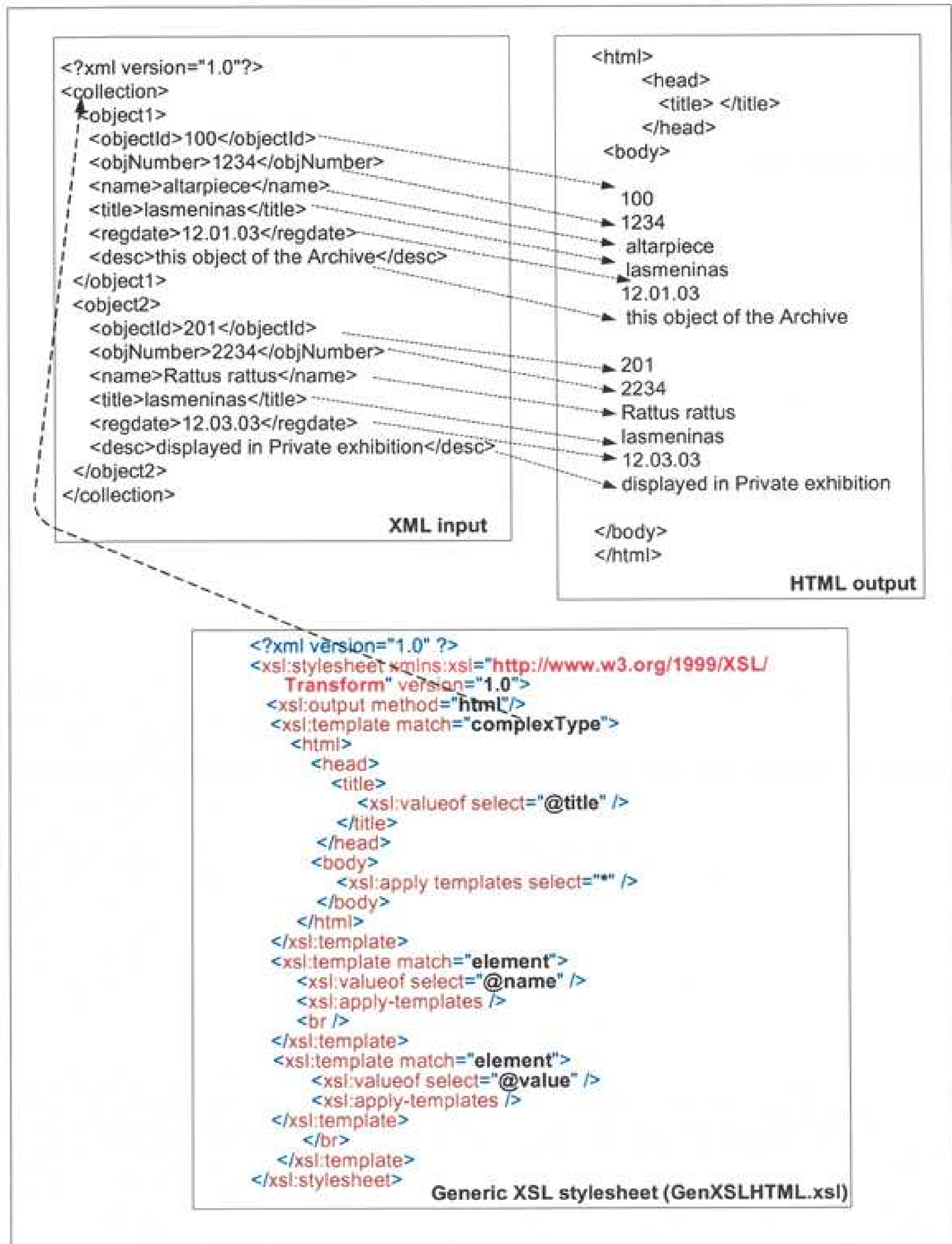
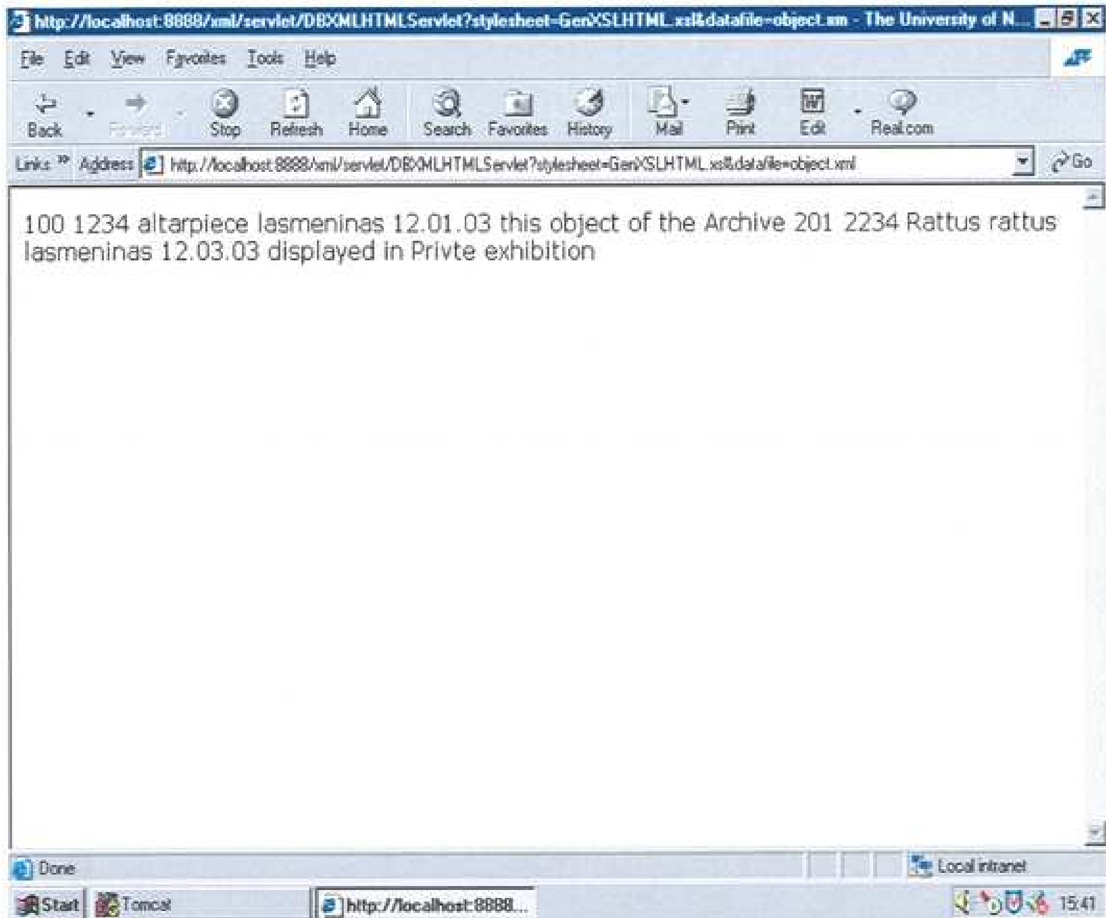


Figure 5.25a: The Result of executing the Java Servlet Class for Transforming XML to HTML by using GenXSLHTML.xsl



- **Example 3 for transforming XML → HTML**

Example 3 is also given for testing the generated XSL stylesheet shown in Figure 5.20. But the result is not displayed by a browser on this Java servlet class which means this example is offline testing. Figure 5.26 shows more details for transforming XML to HTML format. The source code for the implementation for transforming XML to HTML can be found in Appendix B.

Figure 5.26: The Result of executing the Java Class for transforming XML to HTML by using GenXSLHTML.xsl

```
Command Prompt

C:\Tomcat\webapps\xml\WEB-INF\classes\ZTest>jx TranXMLtoHTML GenXSLHTML.xsl object.xml

C:\Tomcat\webapps\xml\WEB-INF\classes\ZTest>c:\j2sdk1.4.0\bin\java -classpath c:\oracle\ora92\jd
ib\classes12.jar;. TranXMLtoHTML GenXSLHTML.xsl object.xml



<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title></title>
</head>
<body>

   100
   1234
   altarpiece
   lasmeninas
   12.01.03
   this object of the Archive


   201
   2234
   Rattus rattus
   lasmeninas
   12.03.03
   displayed in Privte exhibition

</body>
</html>


C:\Tomcat\webapps\xml\WEB-INF\classes\ZTest>

Start | Command Prompt                                                11:25
```

5.2.3 Test plans

In this section we use a test plan to organize the testing of offline components. The test plan describes the way in which we will show that our programs and algorithms work correctly - in other words, that the algorithms are free of faults and perform the function as specified in the requirements. A test plan begins with the test objectives, addressing each type of testing, and shows an overview of testing down into individual components that address a specific item or criteria that will be used in the testing. Table 7 illustrates a test plan's components. The criteria are used to test the algorithms such as test data, expected result, condition test and passed result. According to Table 7, the table contains three algorithms tested for generating different components such as SQL schema and XSL stylesheet for transforming XML to SQL and also XML to HTML. In fact these algorithms are tested with different data as shown in Table 7 i.e. the generation of SQL schema is tested with relational data and the result is as expected.

Also, the generation of XSL stylesheets was tested with relational data and the result was as expected. Table 7 provides a summary of testing the generation components discussed so far.

Table 7: Evolution of the Algorithms

Algorithm Tested	Test data	Expected Result	Condition Test	Passed
1 Generated SQL Schema of XML Schema	Relational data (XMLSchema)	SQL Schema	Offline	√
2 Generated XSL of XML Schema for transforming XML to SQL	Relational data (XMLSchema)	XSL stylesheet	Offline	√
2 Generated XSL of XML Schema for transforming XML to HTML	Relational data (XMLSchema)	XSL stylesheet	Offline	√
3 Generated XSL for transforming of XQuery to SQL	Relational data (XMLSchema)	XSL stylesheet	Offline	√

5.2.3.1 Test Reports

In this section, we analyze the results from testing the algorithms in Table 7 to determine if the function or performance tested meets the requirements. In other words we test the generated offline components presented in Table 7, broken down by functional reports. The testing will be structured to evaluate the reports (generated component). More details on the test report are shown in Table 8, which contains three reports as a result of testing the generated components. Each of these reports is tested with three kinds of examples and the actual result is the expected one.

Moreover, the generated XSL stylesheet is tested with three examples for transforming the XML document to HTML document and the actual result was as expected and shown in section 5.2.2.3.2.

In fact we conclude that these examples not only show how the generator works, but at the same time show the generators can be used for transforming any XML document to SQL strings and any XML document to an HTML document. Table 8 provides a summary of the results for testing the generation components.

Table 8: Evolution of the Reports

Report name	Test data	Expected Result	Actual Result	Condition Test	Passed
Generated XSL to store data in DB	XML data *Example 1*	SQL strings	SQL strings Stored in DB	Online	√
	XML data *Example 2*	SQL strings	SQL strings Stored in DB	Online	√
	XML data *Example 3*	SQL strings	SQL strings Stored in DB	Offline	√
Generated XSL to transform XML to HTML (presentation)	XML data *Example 1*	HTML document	HTML document	Online	√
	XML data *Example 2*	HTML document	HTML document	Online	√
	XML data *Example 3*	HTML document	HTML document	Offline	√
Generated XSL to query XML document	SQL data *Example 1*	XML Document	XML Document querying of DB	Online	√
	SQL data *Example 2*	XML Document	XML Document querying of DB	Online	√
	SQL data *Example 3*	XML Document	XML Document querying of DB	Offline	√

5.2.4 The XQuery interpreter generation

XQuery [83, 87] is a language containing one or more query expressions. XQuery supports conditional expressions, element constructors, FOR, LET, WHERE, RETURN (FLWR) expressions, expressions involving operators and function calls and quantified, type checking, and path expressions. Some XQuery expressions evaluate to simple node values such as elements and attributes, or atomic values such as strings and numbers.

The syntax for retrieving, filtering and transforming records uses FOR, LET, WHERE and RETURN clauses. A FLWR expression creates some bindings, applies a predicate and produces a result set. XQuery does not conform to the same conventions as SQL. But XQuery and SQL share some similar concepts. Both languages provide keywords for projection and transformation operations (SQL SELECT or XQuery RETURN). SQL supports joins between tables, and XQuery supports joins between multiple documents.

Here are two simple examples, one with XPath type of a query and the other with FLWR expression. The single forward slash (/) signifies the parent-child relationship between elements. To trace a path through a tree, the expression starts at the root node.

```
1) X / <collection>/<object>/<objectInfor>

2) For obj in <collection> do
   Return <obj>
   Where obj = <object>
```

The above query1 returns the objectInfor of object in the collection, while query2, a complex query, has the structure for, where and return expression. During execution, the variable obj is bound to collection elements and for each binding check to see that the object exists in the collection and if so returns the object.

Furthermore, we developed a simplified version of XQuery data model based on the existing standard (W3C XML Query) as introduced in Chapter 4.

Now we will introduce an algorithm for generating an XSL stylesheet automatically from the XMLSchema to interpret the XQuery. In other words, this is the technological solution to the problem of the interpreter generating automatically the XQuery.

The algorithm in Figure 5.27 shows how we can generate an XSL stylesheet from an XMLSchema. The result can then be used to transform XQueries to SQL queries as shown in Table 9.

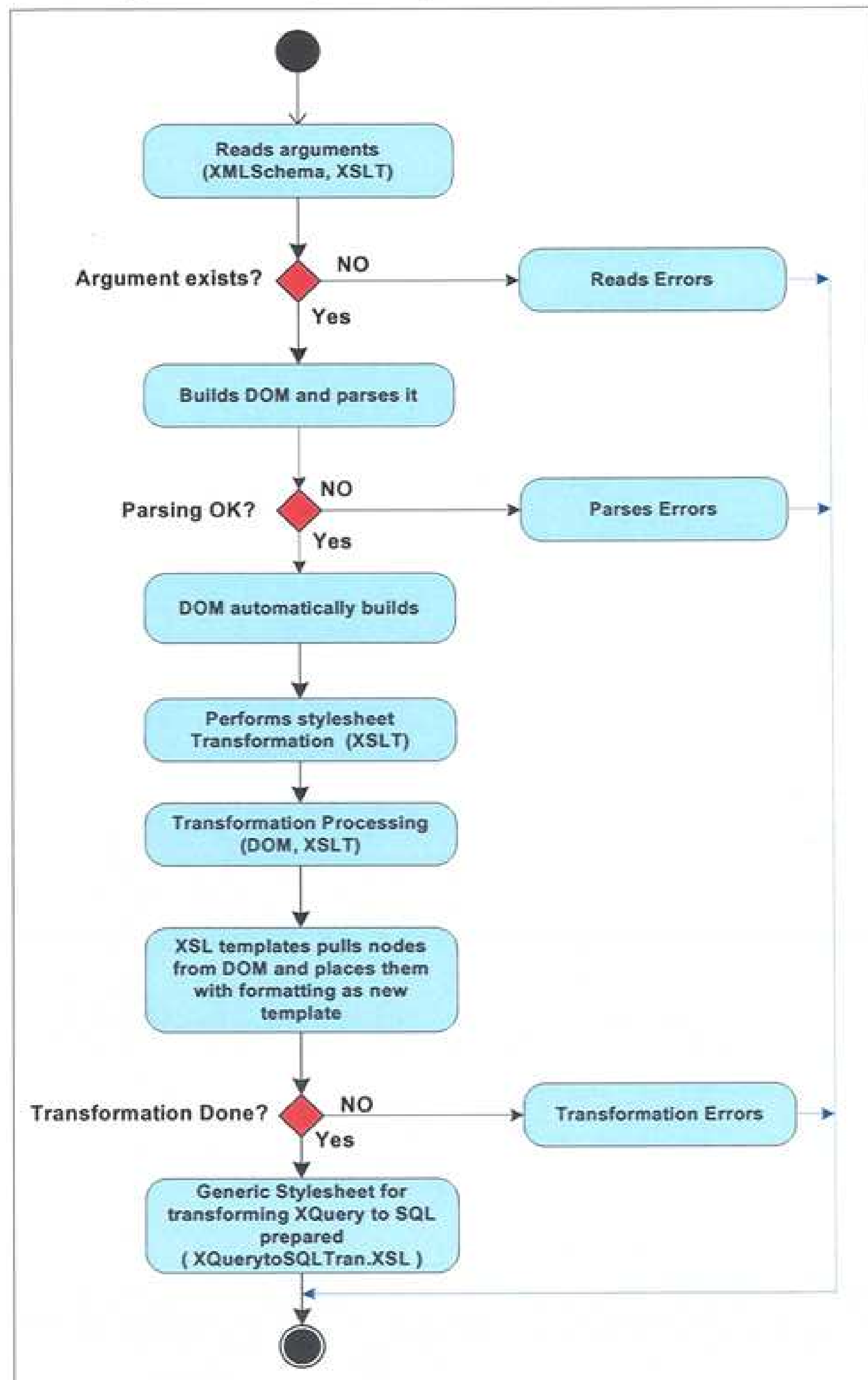
Figure 5.27: An Algorithm for generating XSL to transform XQueries to SQL

```
// Algorithm to generate XSL for transforming XQuery to SQL
// Input: XMLSchema, XSL stylesheet
// Output:  XSL stylesheet  (Generic XSL stylesheet)

 1. start

 2.  if the input arguments (XMLSchema, XSL stylesheet) exist

     2.1 Build DOM and parse it

     2.2 if parsing XMLSchema is done, then DOM will build dynamically

         2.2.1 perform XSL stylesheet transformation

         2.2.2 if transformation is done

               2.2.2.1 the XSL templates start from the root node of DOM tree

               2.2.2.2 If root node has parent/child node

                       2.2.2.2.1 the XSL stylesheet template will walk and match nodes, if
                                 the template matches IN node in DOM tree nodes

                       2.2.2.2.2 the XSL template adds FROM clauses as new template and
                                 maps the root node of DOM tree to the generated template
                                 in the new XSL stylesheet.

                       2.2.2.2.3 the XSL stylesheet template pulls the nodes from DOM tree
                                 and places them with formatting, into a new XSL stylesheet

                       2.2.2.2.4 the XSL stylesheet template walks through the DOM tree
                                 node and when the template matches RETURN clauses

                       2.2.2.2.5 add the SELECT clauses to the template followed by the
                                 name of parent/child of DOM as new template to output

                       2.2.2.2.6 Iterate and walk through the DOM tree nodes

                       2.2.2.2.7 if the XSL stylesheet template matches nodes has WHERE
                                 or IF or ELSE, set the WHERE clauses as new template
                                 into the new XSL stylesheet

                       2.2.2.2.8 When all the templates have been executed and placed in
                                 the output, then the new templates will be a generated
                                 generic XSL stylesheet

               2.2.2.3 report no parent/child node and terminate

         2.2.3 report transformation errors and terminate

     2.3 report parsing errors and terminate

 3.  report arguments error and terminate

 4. terminate
```

5.2.4.1 How the algorithm works

The diagram in Figure 5.28 shows the decisions required when generating the XSL stylesheet for transforming XQueries to SQL queries.

Figure 5.28: Diagram for generating an XSL stylesheet



5.2.4.2 Testing the algorithm

In this section we will use the simplicity test, not a formal test, for testing the algorithm shown in Figure 5.27. The input of the algorithm is XMLSchema. Let us walk through the algorithm by using the XSL stylesheet and XMLSchema as an input data for testing the algorithm.

The algorithm is invoked with its starting point by checking that arguments exist (inputs of the algorithm); if they exist then parse and "Build DOM". If the parsing result is executed (line 2 - 2.2 in Figure 5.27) then the DOM tree will build dynamically. If the transformation is done, (line 2.2.2 in Figure 5.27), the XSL stylesheet templates match the root node in DOM tree and map it as a template to the new XSL stylesheet. If the root node has child nodes, the XSL stylesheet will walk through the DOM tree and if the template matches IN node then the XSL template adds FROM clause as a new template and maps the root node to the generated template (line 2.2.2.2.1 - 2.2.2.2.2 in Figure 5.27). Then the XSL stylesheet pulls nodes from the DOM tree and places them, with formatting, to output as new templates to the new XSL stylesheet (line 2.2.2.2.3 in Figure 5.27). Iterate through the nodes in the DOM tree and if the template matches RETURN then add the SELECT clauses as a new template, followed by the parent/child node of the DOM tree to the new template to output (line 2.2.2.2.4 - 2.2.2.2.5 in Figure 5.27). If the XSL template matches nodes through WHERE or IF or ELSE, then set the WHERE clauses as a new template into the new XSL stylesheet (line 2.2.2.2.6 - 2.2.2.2.7 in Figure 5.27). When the walk in the DOM tree nodes is finished, then the output of the algorithm will be a generic XSL stylesheet for transforming XQueries to SQL queries (line 2.2.2.2.8 in Figure 5.27). Figure 5.29 shows the generated XSL stylesheet as output for executing the algorithm by using the XMLSchema and XSL stylesheet. The source code for implementing the XQuery interpreter-generating algorithm can be found in Appendix B.

5.2.4.2.1 Output of the algorithm

As already mentioned, the output of the algorithm is a generic XSL stylesheet for transforming XQueries to SQL queries. Figure 5.29 shows the XSL stylesheet generated as output from executing the algorithm. Also Figure 5.29a shows the result of the execution of the Java class for generating the XSL stylesheet (XQuerytoSQLTran.xsl).

Figure 5.29: Generic XSL Stylesheet for transforming XQueries to SQL Queries

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text" />
<xsl:template match="return">
    SELECT <xsl:value-of select="@name"/></xsl:template>
<xsl:template match="in">
    FROM <xsl:value-of select="@name"/></xsl:template>
<xsl:template match="Where">
    WHERE <xsl:value-of select="@name"/></xsl:template>
<xsl:template match="equal">
    FROM <xsl:value-of select="@name"/></xsl:template>
<xsl:template match="if">
    WHERE <xsl:value-of select="@name"/></xsl:template>
<xsl:template match="then">
    SELECT <xsl:value-of select="@name"/>
</xsl:template>
<xsl:template match="else">
<xsl:value-of select="@name"/></xsl:template>
<xsl:call-template name="lower">
<xsl:with-param name="value" select="@name"/></xsl:call-template>
</xsl:template>
<xsl:template match="Where" mode="@name">
<xsl:call-template name="lower">
<xsl:with-param name="value" select="@name"/></xsl:call-template>
</xsl:template>
<xsl:template match="equal" mode="@name">
<xsl:call-template name="lower">
<xsl:with-param name="value" select="@name"/></xsl:call-template>
</xsl:template>
<xsl:template match="if" mode="@name">
<xsl:call-template name="lower">
<xsl:with-param name="value" select="@name"/></xsl:call-template>
</xsl:template>
<xsl:template match="then" mode="@name">
<xsl:call-template name="lower">
<xsl:with-param name="value" select="@name"/></xsl:call-template>
</xsl:template>
<xsl:variable name="lowercase" select="'abcdefghijklmnopqrstuvwxyz'"/>
<xsl:variable name="uppercase"
select="'ABCDEFGHIJKLMNOPQRSTUVWXYZ'"/>
<xsl:template name="lower">
<xsl:param name="value" />
<xsl:variable name="complexname"
select="translate($value,$uppercase,$lowercase)"/>
<xsl:value-of select="$complexname"/>
</xsl:template>
</xsl:stylesheet>
```

Figure 5.29a: The Result of executing the Java Class for generating the XSL stylesheet (XQuerytoSQLTran.xsl)

```
Command Prompt

C:\Tomcat\webapps\xml\WEB-INF\classes>jx GenerateXSLXQuerySQL GenXQueryXSL.xsl XMLSchema.xsd

C:\Tomcat\webapps\xml\WEB-INF\classes>c:\j2sdk1.4.0\bin\java -classpath c:\oracle\ora92\jdbc\lib\
sses12.jar;. GenerateXSLXQuerySQL GenXQueryXSL.xsl XMLSchema.xsd

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="text"/>
<xsl:template match="return">
    SELECT <xsl:value-of select="@name"/>
</xsl:template>
<xsl:template match="in">
    FROM <xsl:value-of select="@name"/>
</xsl:template>
<xsl:template match="Where">
    WHERE <xsl:value-of select="@name"/>
</xsl:template>
<xsl:template match="equal">
    FROM <xsl:value-of select="@name"/>
</xsl:template>
<xsl:template match="if">
    WHERE <xsl:value-of select="@name"/>
</xsl:template>
<xsl:template match="then">
    SELECT <xsl:value-of select="@name"/>
</xsl:template>
<xsl:template match="else">
<xsl:value-of select="@name"/>
</xsl:template>
<xsl:template mode="@name" match="in">
<xsl:call-template name="lower">
<xsl:with-param select="@name" name="value"/>
</xsl:call-template>
</xsl:template>
<xsl:template mode="@name" match="Where">
<xsl:call-template name="lower">
<xsl:with-param select="@name" name="value"/>
</xsl:call-template>
</xsl:template>
<xsl:template mode="@name" match="equal">
<xsl:call-template name="lower">
<xsl:with-param select="@name" name="value"/>

Start | Command Prompt                                   11:19
```

5.2.4.2.2 Testing the generator for transforming XQueries to SQL queries

Table 9 contains examples of XQueries transformed to SQL queries by using a generic XSL stylesheet shown in Figure 5.29. The criteria used to test the algorithms are test data XQueries, condition test, expected result and the result status to show the result of using the generator (XSL stylesheet) for transforming XQueries to SQL queries. As can be seen in Table 9 the generator is used for transforming two kinds of queries as follows:

- For complex XQueries containing a loop, use the condition as shown in the queries 1, 2, 3, 4, 5, 6, 7, 9, 11 and 13 of Table 9.
- The generator can be used for transforming the XPath queries as shown in the queries 8, 10 and 12 of Table 9.

The following Figures 5.30, 5.30a and 5.30b show the implementation of some transformed XQueries by using the generated XSL stylesheet (XQuerytoSQLTran.xsl) to SQL queries with the result displayed on the Tomcat server using Java servlet class.

According to Table 9, these examples show how the generator works in a generic manner, which means it can be used for transforming any XQuery to an SQL query. More details on the implementation of the algorithm for transforming XQueries in Table 9 to SQL queries can be found in Appendix C.

Table 9: Transforming XQueries to SQL Queries using the XSL generated

No.	Test data (XQuery)	Condition Test	Expected Result (SQL Query)	Passed
1	**For** coll in <collection> do **Where** coll=coll/<clltn_id> **Return** <clltn_id> <clltn_num> <name> <title>	online	**Select** coll.clltn_id, coll.clltn_num, coll.name, coll.title **From** collection coll **Where** coll=coll.clltn_id	✓

2	**For** obj in <collection>/<object> do
	if obj/<obj_id>=obj/<clltn_id> then
	Return <obj_id> <obj_number>
	<name> <title> **else** ()
		online	**Select** obj.obj_id, obj.obj_number,
			obj.name, obj.title
			From collection obj, object obj
			Where obj.obj_id=obj.clltn_id
				✓

3	**For** inst in <institution> do **Return** <inst>	online	**Select** * **From** institution inst	✓
4	**For** inf in <object>/<information> do **Where** inf/<info_id> = inf/<obj_id> **Return** <info_id> <desc_date> <description>	online	**Select** inf.info_id, inf.desc_date, inf.description **From** object inf, information inf **Where** inf.info_id = inf.obj._id	✓
5	**For** obj in <object> do **Where** obj = obj/<obj_number> **Return** <obj>	online	**Select** obj.obj_id, obj.obj_number, obj.name, obj.title **From** object obj **Where** obj=obj.obj_number	✓
6	**For** obj in <collection>/<object> do **Return** <object>	online	**Select** * **From** collection obj, object obj	✓
7	**For** inf in <object>/<information> do **Return** <info_id> <desc_date> <description>	online	**Select** inf.info_id, inf.desc_date, inf.description **From** object inf, information inf	✓
8	obj/<object>	online	**Select** * **From** object obj	✓
9	**Let** coll <collection>/<object> do **For** obj in <object> **Where** coll/<name> = obj/<name> **Return** <object>	online	**Select** obj.obj_id, obj.obj_number, obj.name, obj.title **From** collection coll, object obj **Where** coll.name = obj.name	✓
10	obj/<collection>/<object>	online	**Select** obj.obj_id, obj.obj_number obj.name, obj.title **From** collection obj, object obj	✓
11	**Let** obj = <object>/<information> do **Where** obj/<obj_id> = obj/<info_id> **Return** <information>	online	**Select** * **From** object obj, information obj **Where** obj.obj_id = obj.info_id	✓
12	inf/<collection>/<object>/<information>	online	**Select** inf.info_id, inf.desc_date, inf.description **From** collection inf, object inf, information inf	✓
13	**For** exh in <exhibition> do **Where** exh/<opened> **Return** <exh>	online	**Select** * **From** exhibition exh **Where** exh.opened = 'yes'	✓

Figure 5.30: Execution of Transformation from XQuery to SQL (xq1.ixq) in Table 9

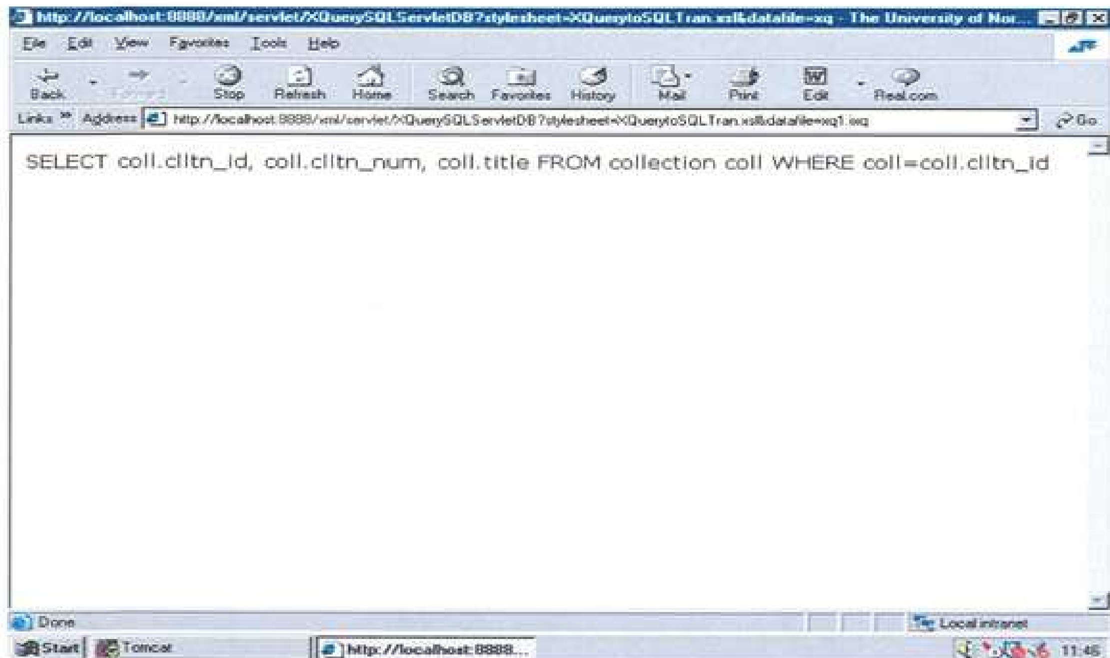


Figure 5.30a: Execution of Transformation from XQuery to SQL (xq2.ixq) in Table 9

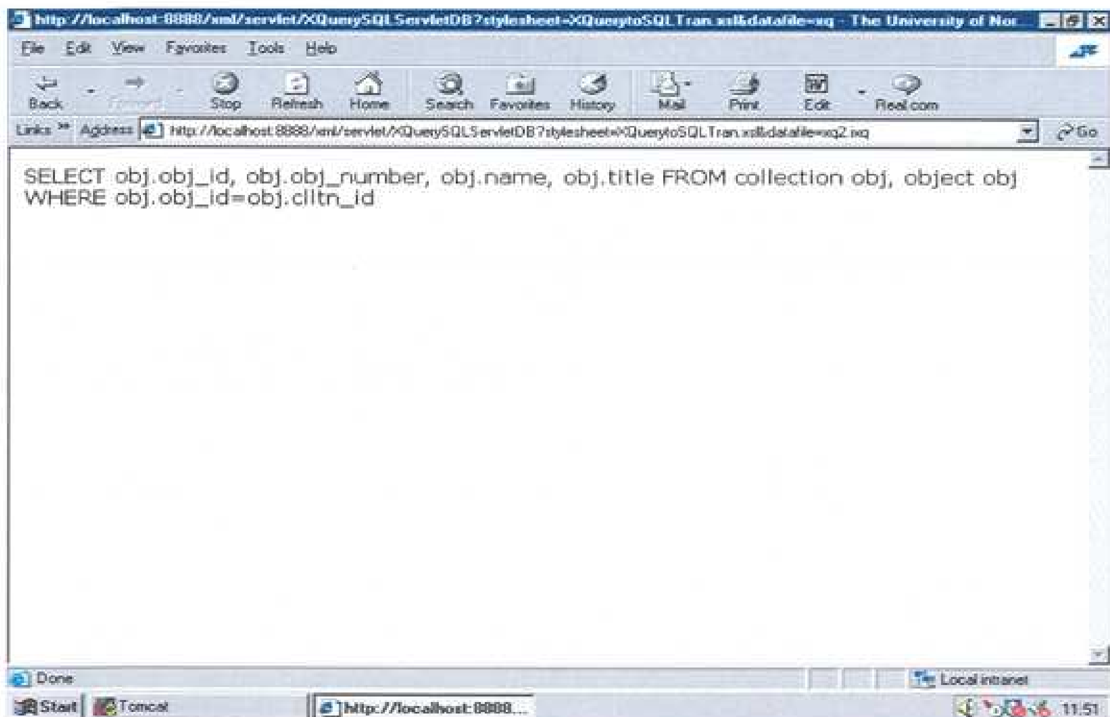
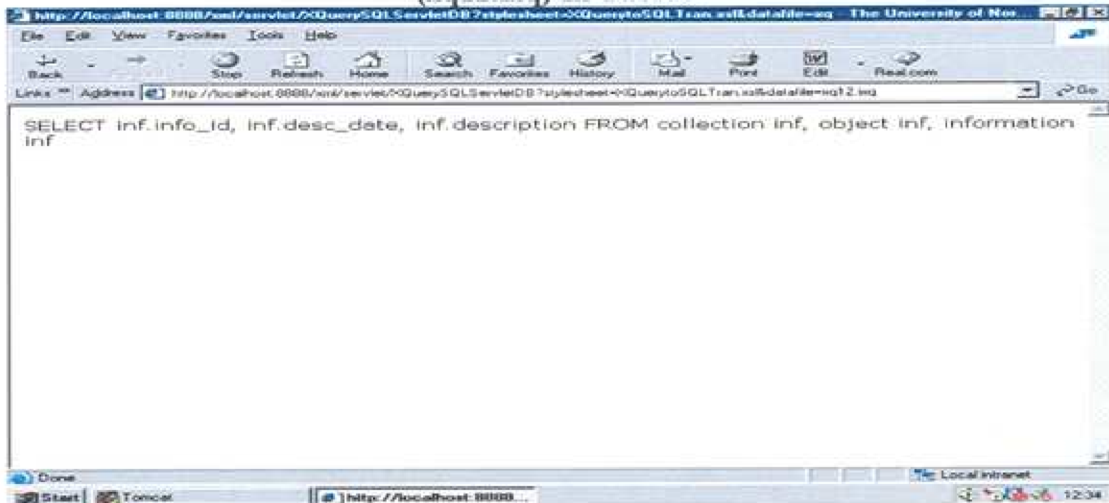


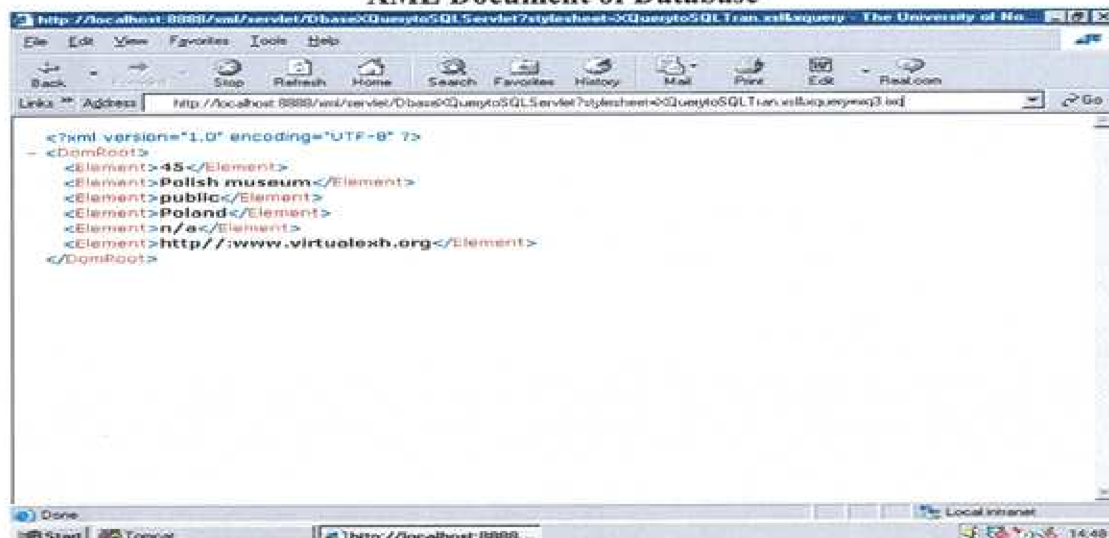
Figure 5.30b: Execution of Transformation from XQuery to SQL (xq12.ixq) In Table 9



5.2.4.2.3 Testing the generator and browsing the result on Tomcat server

In this example the client browser needs to display an institution information on the database. The client browser will send an XQuery such as (For inst in <institution> do Return <inst>). Then the XSL stylesheet (XQuerytoSQLTran.xsl) we generated will transform the XQuery to an SQL query string and the Java servlet connected to the DB over JDBC will execute the query to retrieve data from the database according to the SQL query. Then we will return the result in XML format to the Web browser. The result is shown in Figure 5.31. In this example we execute the XQuery (xq3.ixq) in Table 9.

Figure 5.31: Transforming XQuery to SQL (xq3.ixq) in Table 9 and generating XML Document of Database



5.2.5 Test Reports

Table 9 contains 13 examples tested with the generated XSL stylesheet for transforming XQuery to SQL. Each of these examples was tested and the actual result shown in Table 9 was as expected.

We conclude that these examples show how the generator works in a generic manner, which means it can be used for any XML query to SQL query. Table 10 provides a summary of the results for testing the generation components.

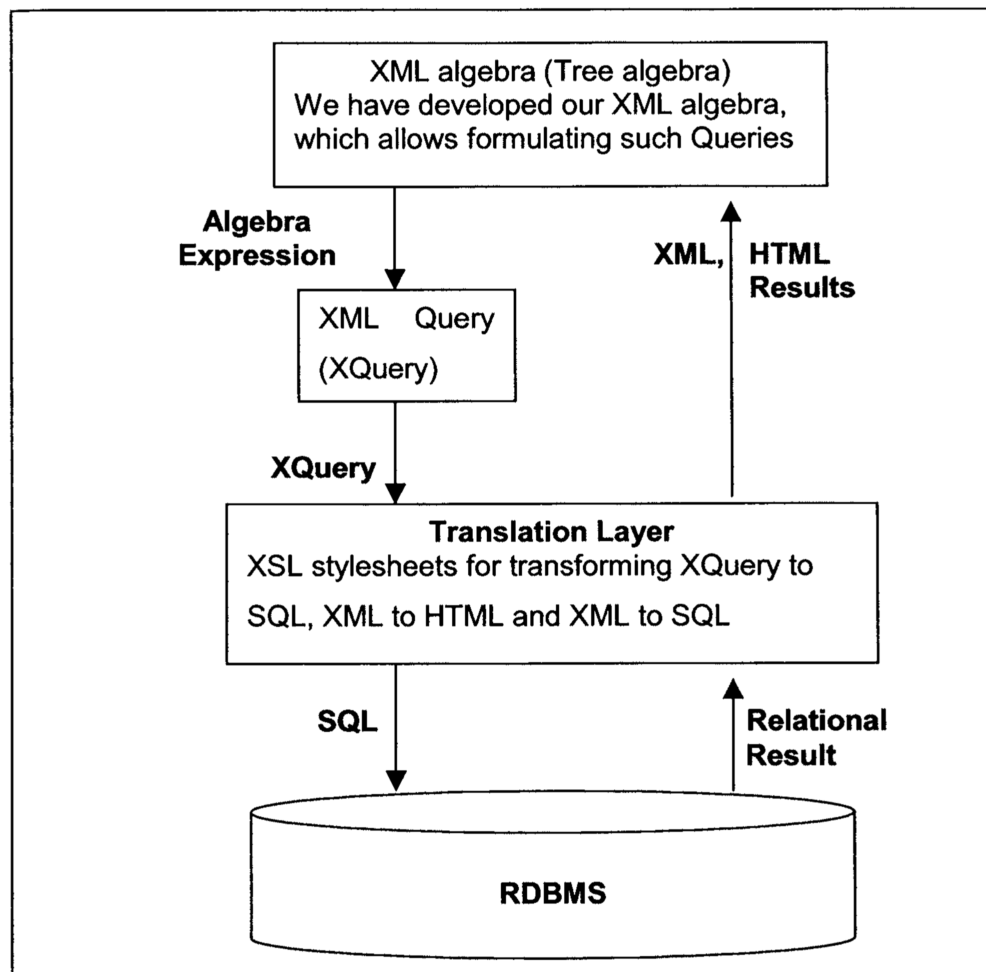
Table 10: Evolution of the Reports

Query Tested	Expected Result	Actual Result	Condition Test	Passed
3 To generate XML of SQL schema	XML document	XML document	Offline	√
4 To generate XML of SQL schema	XML document	XML document	Online	√
7 To generate XML of SQL schema	XML document	XML document	Online	√

5.2.6 The contribution of XML algebra and the XQuery in the implementation

This section introduces the contribution of XML algebra and the XQuery in the implementation. We have developed our XML algebra (Tree algebra) to allow the formulation of such queries, then we built software to interpret this query and the query is executed against the database to retrieve the data. The translation layer contains the XSL stylesheets developed for transforming the XQuery to SQL which is stored in the database. The relational data is then transformed into HTML or XML documents by using the XSL stylesheets on the translation layer, as shown in Figure 5.32. As already mentioned, the algebraic model we have developed for data management, data presentation, data communications and data processing is based on an XML Schema. Figure 5.32 shows in more detail the contribution of XML algebra and the XQuery to the implementation. Chapter 6 deals in more detail with the implementation.

Figure 5.32: Contribution of Algebra and XQuery in the Implementation



5.3 Conclusion

In this chapter we have introduced the mapping of an XML document into a variety of formats such as tree-to-tree, tree to relation, and investigated how the XML document can be stored in different ways. Furthermore, we have introduced different types of methods for transforming XML documents into different formats with XSL stylesheet transformation such as Offline and Online. In other words we have developed the technological solution for the system in generating components and that is how the XMLSchema drives the generation architecture of components.
However, the XSL stylesheet is not just transforming XML to HTML or another format but also can be used to generate components from XMLSchema. The XSL stylesheet transformation provides a generic solution for transforming XMLSchema. We presented techniques for generating an SQL schema, XSL stylesheet and XQuery by using the XMLSchema. These techniques, which (a) enable us to transform XML data to SQL data and store it in a relational database, (b) allow the user to present HTML format, and (c) interpret XQueries, which means transforming XQueries to SQL queries and then we can retrieve and query data from the database.

A potential cause for concern is that our general techniques may be less overlapping. However, based on our prototype implementation in Java, we have found that it is very quick to generate XSL stylesheets as an interpreter for different types of transformations such as SQL Schema and XQueries in to SQL queries. It is now possible to code by hand an XSL stylesheet that validates an XML document against some or all constraints of XMLSchema and to generate an XSL stylesheet, but this takes time and in our system such techniques are easy, quick and overlap less so that through the generation of different generic components they do not occupy as much time.

At the same time this part of the research can be used for evaluating the whole work. Demonstrating acceptable pictures even without very precise measures would be sufficient to prove the value of our research in providing flexible XML mapping. We have been successful in the implementation of the generic approach, which is extremely attractive from a functional point of view.

Chapter 6:

A prototype development of the system

As a test framework of our integrated approach we have implemented a prototype system for exchange of information between several independent museums as an example.

The prototype is an experimental implementation of an integrated system for XML data processing, which covers all the main functions needed for data processing (communication, validation, transformation, presentation, storage and retrieval) and uses all the main components of the online part of the system (web servers, XML parsers, XSL processors, HTML browsers, databases and XQuery interpreters). The domain of virtual museum exhibitions has been selected as a suitable application because of its relative simplicity, good standardization and wider applicability. In the following we will introduce a brief history on the museum systems and the CIDOC documentation standard including the object identification class, use cases, packages, the system design and the implementation of the integrated system.

6.1 Museum information systems

6.1.1 Brief history

As the Web has become increasingly the world's official media, many museums, archives, libraries, and cultural heritage centres throughout the world invest in documenting their collections and in publishing their material via the web, making their information accessible on the Internet.

Generally, museums use many different styles in presenting their collections in their Web pages. In addition, some museums provide sophisticated searching facilities so that the user can retrieve information about required items by subject, date and place. Others classify their material into groups to satisfy the user's requirement. It is fairly certain that with such sophisticated new technologies, the need for standards to manage the information that these collections contain becomes more and more urgent. As a result, several projects have been conducted in the last few years in order to develop a unified standard for organizing managing data and information in such institutions. In the same way, many museum organisations are working together to develop standards and techniques and to make their resources more widely and easily available.

On this basis, the CIDOC[1] and CIDOC documentation standard working group (DSWG) have engaged in the creation of a general data model for museums. The CIDOC has particularly focused on information interchange. As a result, these efforts resulted in 1999 in the first complete package of the "CIDOC Conceptual Reference Model" (CRM). Such an achievement was the result of intensive hard work from many experts both in the field of IT and museum information specialists in order to fully exploit the potential of the CRM as a means to enable information interchange and integration in the museum community and beyond.

Later, the CIDOC group held a meeting in London at 1999, when they decided to submit the CRM proposal to the ISO for standardization. Consequently, CIDOC has been authorized to use the ISO facilities that have resulted in the acceptance of CIDOC CRM as well as a defined global standard under ISO reference number ISO/AWI21127 by ISO/TC46/SC4 [102]. In other words, CIDOC will make use of the services of ISO and collaborate with the respective ISO committees to formally standardize the CRM and to ensure the widest global agreement on it in the whole international community interested in this field. With regard to the CRM standard, the primary role of the CRM is to serve as a basis for mediation of cultural heritage information and thereby provide the semantic 'glue' needed to transform today's disparate, localised information sources into a coherent and valuable global resource [102, 112].

As has been mentioned before, such massive work has been developed and optimized by CIDOC. But what does "CIDOC" mean? The next section gives a brief description of CIDOC and CRM. The "CIDOC object-oriented Conceptual Reference Model" (CRM), was developed by the ICOM/CIDOC Documentation Standards Group [102,121,122]. Since September 2000, the CRM has been developed into an ISO standard in a joint effort of the CIDOC CRM SIG and ISO/TC46/SC4 (ISO/AWI 21127). It represents an "ontology" for cultural heritage information i.e., it describes in a formal language the explicit and implicit concepts and relationships relevant to the documentation of cultural heritage.

[1] (CIDOC) Committee International for DOCumentation

6.1.2 Object-Oriented Conceptual Reference Model (OOCRM)

The object-oriented [104] CIDOC Conceptual Reference Model (referred to as "CRM") is the result of work done by the CIDOC Documentation Standards Group, from 1994-2000, and the CIDOC CRM Special Interest Group from 2000-2002. CIDOC came from an initiative to define the underlying semantics of database schemata and document structures needed in museum documentation for the support of good practice in conceptual modelling, data transformation, data exchange, information integration and mediation of heterogeneous sources [107].
Basically, the scope of the CRM is the formal knowledge of any particular museums that is information required solely for the administration and management of cultural heritage institutions. That implies that any other information relating to a museum such as personnel, accounting, and visitor statistics, is not covered by CRM scope. For it is specifically intended to cover contextual information: the historical, geographical and theoretical background in which individual items are placed and which gives them their significance and value.

With regard to heritage collections, the term *cultural heritage collections* covers all types of material collected and displayed by museums and related institutions, as defined by ICOM [112, 123]. This includes collections, sites and monuments relating to natural history, ethnography, archaeology, historic monuments, as well as collections of fine and applied arts. The exchange of relevant information with libraries and archives, and the harmonisation of CRM with their models, falls within CRM's intended scope.

6.1.3 Primary objectives of CRM

The CIDOC group had specific objectives for establishing the CRM standard. These objectives can be summarized as follows:

- CRM standard as a primary tool for the museum community.
- Enabling effective communication between the museum community and different heritage preserving institutions.
- Supporting data interchange within the museum community.
- Providing a core language that allows for integrating the semantics of heterogeneous data structures.

- To be more specific, such a standard should provide some key features for its users in order to make it easy, effective and more productive. These key features are:

  - The model is self-explanatory and extensible.
  - The semantic deviations between various CRM models are minimal.
  - The CRM model has been formulated to be an object oriented semantic model.
  - The system is natural and expressive for domain experts.
  - It is easy to convert to other machine-readable formats such as RDF and XML.

Now throughout the analysis and design stages in this research, a Unified Modelling Language (UML) [110,111] is used to present the concepts and techniques necessary to effectively use system requirements captured with diagrams of use case to drive the development of a robust design model.

The main aim of using UML is to be able to produce detailed object models and designs from system requirements, in addition to identifying the use case diagrams for the proposed system and expanding them into full behavioural designs.

6.2 Object identification and class deriving

Basically, the first step in constructing a class diagram is to identify objects in the problem domain [106, 123]. These may be physical objects, such as people or documents of organizational entities. All the objects identified at this stage relate to the problem domain. In the next paragraph all the nouns underlined are identified as potential objects.

The proposed museum system aims to provide different kind of services to different kinds of clients. These clients are different kinds of users; these users are recognized as persons. In addition every user has limited access to the system (rights). They are categorized as follows:

Internally, staff are responsible for publishing the museum objects and collections (set of museum objects), in addition to their main task in handling different kinds of information, such as data related to objects, institutions and staff. Next, there is the curator, the person who manages the exhibition.

Then, there is the collector, the person who collects information related to museum objects (collection information- CIDOC).

The information related to museum objects is classified into six categories: image information, documentation, acquisition information, location information, reference information and collection information.

Externally, the proposed system has three different kinds of persons. Firstly, the Visiting groups, those who visit the Public displays (different exhibitions). Also, the public display can be visited by individuals (visitors). Secondly, the External person (who is a member of the Board), this board belonging to an institution; the institution could be a public, international, private or state institution. In addition, the museum archive will be considered as a type of exhibition in the proposed system.

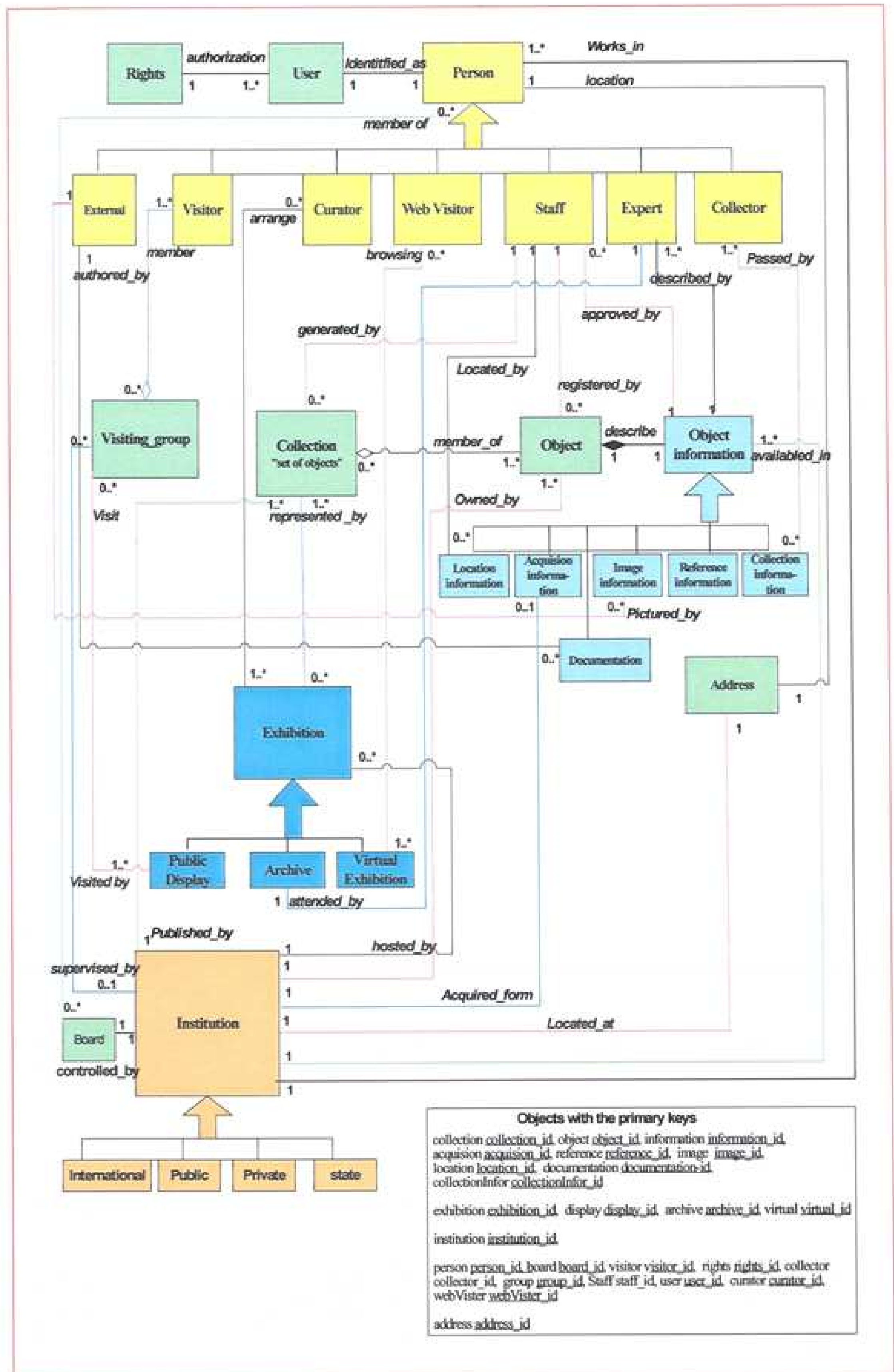
Finally, the proposed system will be visited via the web by different kinds of visitors (web visitors). These visitors will visit the virtual exhibition, which is a kind of exhibition. In the proposed system, the address might be needed for the institutions and persons.

The purpose of identifying objects in the problem domain is to derive useful classes. The object classes that can be picked out from the museum system problem brief are shown in Table 11 while Figure 6.1 shows the museum system's association diagram.

Table 11: Objects in the museum system

Person	Collection	Exhibition	Institution
Staff	Objects	Virtual exhibition	Public
Curator	Object Information	Public display	State
Collector	Image information	Archive	Private
Web visitor	Acquisition information		International
Visitor	Location information		Address
Visiting group	Reference information		
External	Collection information		
Board	Documentation		
User			
Rights			

Figure 6.1: Association diagram



6.3 Use Case Diagrams

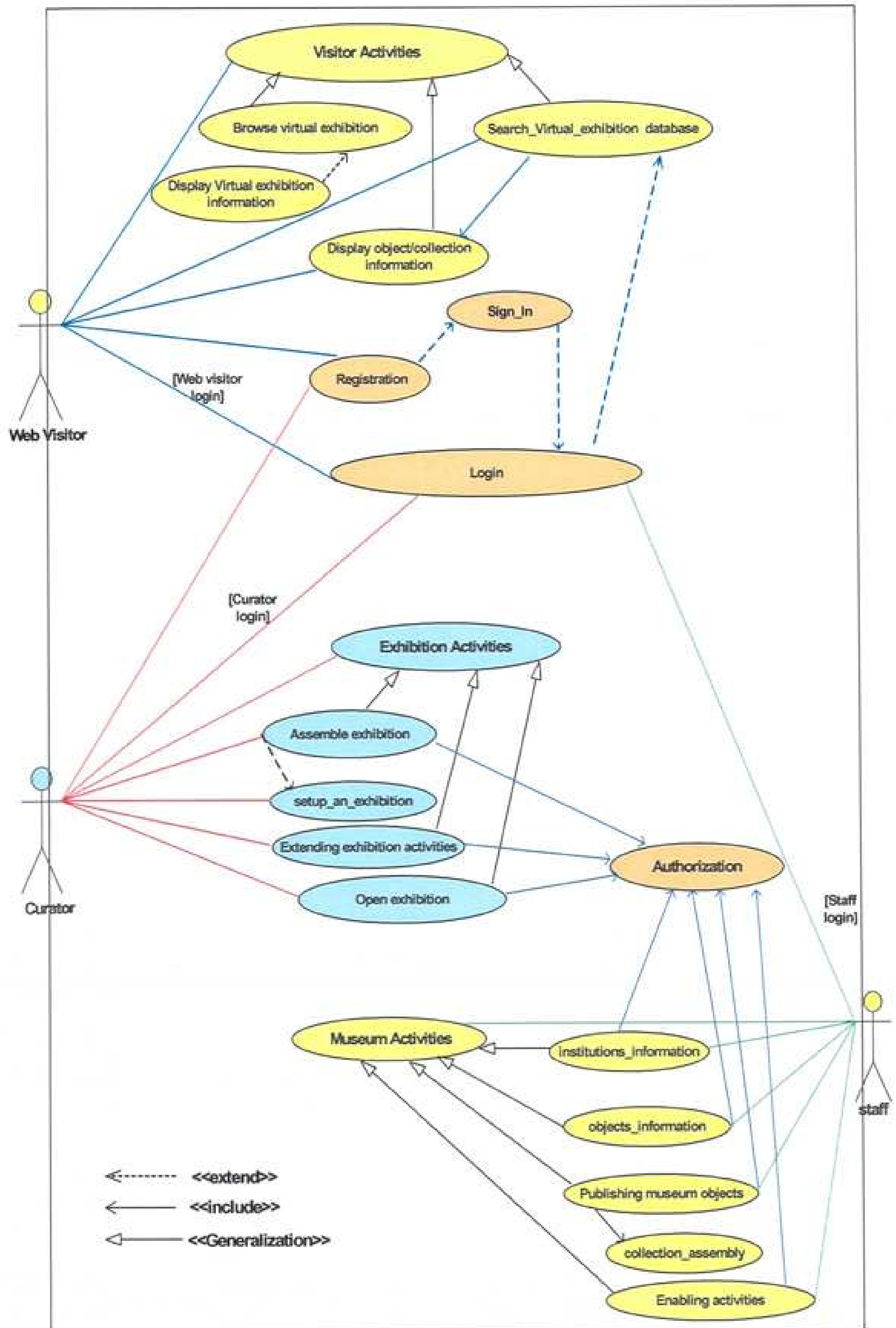
A use case is a sequence of actions that an actor performs within a system to achieve a particular goal. Also, the use case diagram shows the interactions between the roles of the system users, called actors, and subsets of system functionality [103,105]. Analysis starts with the search for the actors (categories of users) of this access museum system. An actor represents a generic role played by someone or something interacting with the system. A use case diagram captures a model of several use cases, which depend on one another and how one or more actors interact with those use cases. Figure 6.2 shows a use case diagram for the museum system and the relationships between the actors and the use cases. It is not always easy to determine the boundaries of the system but by definition, the actors are always outside it.

The actors are 'recruited' from the system's users and the people responsible for its configuration and maintenance. Figure 6.2 shows three different actors in the museums system. They are divided up into the following categories:

- Web visitor can have access to the system for browsing or for search and display activities.
- Curator has the highest privilege: (Curator) can have an access as Staff but Staff cannot have access as Curator.
- Staff can access the system as Web visitor, but Web visitor cannot access the system as staff.

Furthermore each use case, rendered as an oval in the diagram, accompanied by structured document information such as a goal statement, priority, assumptions and list of activities, describes how the actors fulfill the identified goal. As shown in Figure 6.2, a dependency between use cases may be labelled as <<extend>> or <<include>> or <<generalization>>. Assemble exhibition includes Authorization, which means that the first use case (referred to as the base use case) depends on the results or outcome of the included use case. The use case extends Assemble exhibition in this model. The extensions specify behaviour that is optional or exceptional in the description. This use case may be the extension used by another use case.

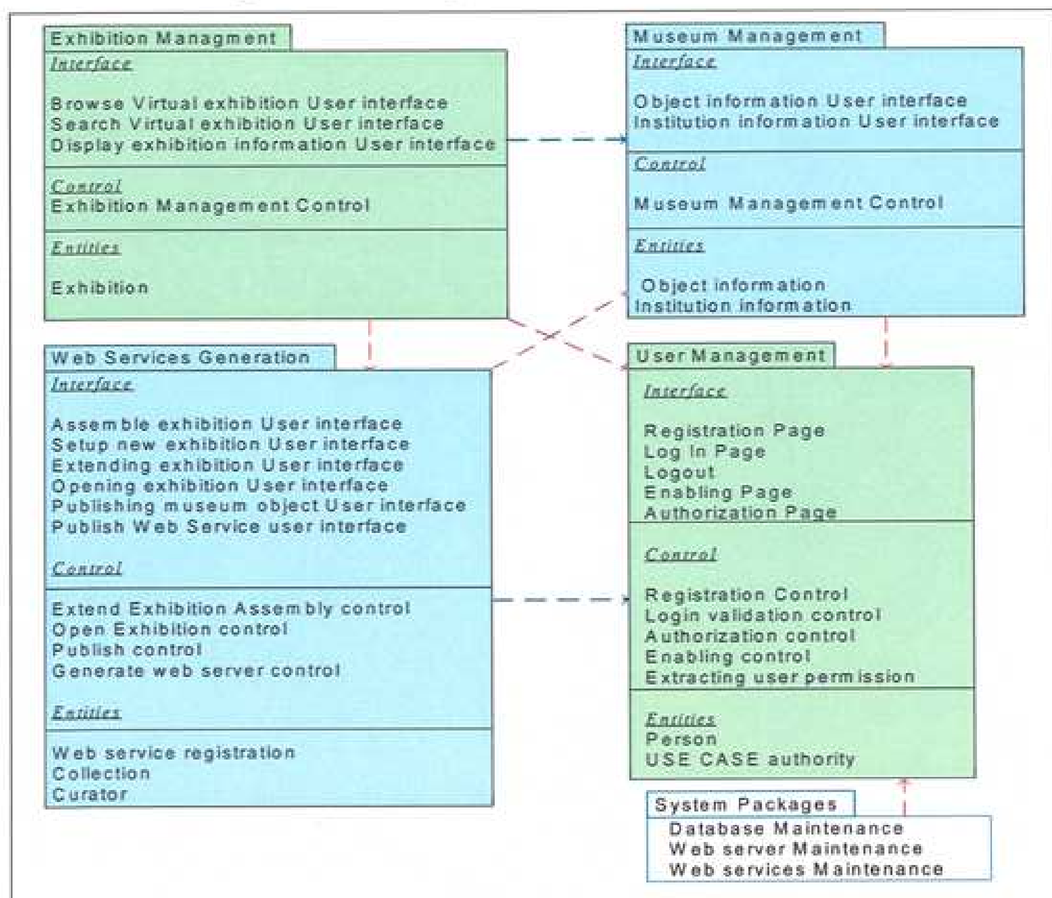
Figure 6.2: Use case diagram



6.4 Packages

A package is a grouping of pieces of a model. Functionally, the packages are very useful in managing models. The packages we have identified represent different views of the proposed system. These views are organized and described in packages as follows: exhibition management, museum management, web services generation and user management. For each package in Figure 6.3 a subset of the classes that it contains is listed. The dependency arrows in the diagram indicate that the *Exhibition Management* subsystem depends on three packages such as *Museum Management, Web Services Generation* and *User Management*. The *Museum Management* depends on the *User Management* package and also the *Web Services Generation* subsystem depends on the *Museum Management* and *User Management* subsystems. Moreover, the System Package depends on the *User Management subsystem*. However the *User Management* subsystem is specified independently of the others. Each package contains interfaces to user pages, control class and one or more entities, as shown in Figure 6.3.

Figure 6.3: Packages for the Museum System



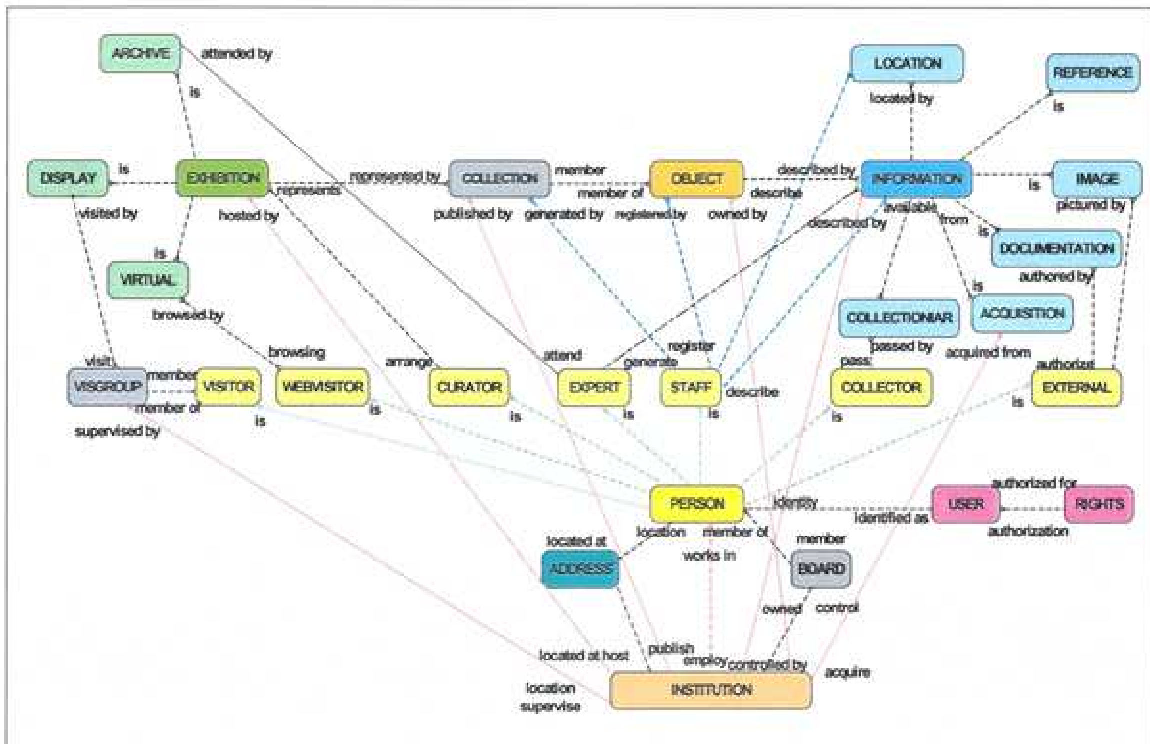
6.5 Design stage

This part gives a description of the proposed mode. It covers in detail the design aspects of the system.

6.5.1 Design of database for the proposed system

The Entity Relationship Diagrams (ERDs) [127,128] illustrate the logical structure of databases for our proposed system. Figure 6.4 shows the cardinality constraints between the objects. As can be seen from the diagram each Collection may relate to a numbers of Objects. Also the Object may contain much Information. Information has Location, Reference, Image, Documentation, Acquisition and Collection information. The Exhibition has many Collections, and the Exhibition can be of different kinds such as a public display, virtual exhibition or archive (type of exhibition). An Institution has many Visitor groups, Exhibitions, Collections, Objects, Information and one Address. Note that the Person could be Staff, Expert, Curator, Web visitor, a Visitor group, Collector or External this is also one to many cardinality constraints. More details on the entity relationship between the objects classes for our proposed system are shown in Figure 6.4.

Figure 6.4: Entity relationship diagram



6.5.2 Design of XMLSchema specification

The XMLSchema [101,108,143] is used to specify the structure and constraints on the XML documents. Moreover the XMLSchema language can be used to define, describe and catalogue XML vocabularies for classes of XML documents. In our proposed system we design a XMLSchema that defines the elements, attributes, whether an element is a child, whether an element is empty or includes text, data type for elements and attributes and fixed values for elements and attributes in our system. Figure 5.15 in Chapter 5 shows a fragment of our XML Schema.

Our XMLSchema has exactly one root element as <XMLSchema>. There are many <complexType> elements in our schema, each of which defines an element type in the schema. Any element that contains attributes or child elements is defined by using a complexType. Furthermore the XMLSchema specification makes a clear distinction between *definitions* and element *declarations*. <element> is a declaration of an element that may appear in a valid document instance, but it does not define that element type. It declares that an element named "collection" has type "collection" that is elsewhere defined in the schema as <element name="collection" type="collection">. In fact, it declares the root element for vaild document instances. Each of the <complexType> definitions includes child elements that define the content model and/or attributes for this element type. Both the child element definitions and the attribute definitions specify the type of their content. Each complexType must define the content model for its child elements. The content model for collection, object, objectInformation, exhibition, institution and person is a sequence which has identical properties to the sequence in the schema. The XMLSchema collection definitions may use element type inheritance. As the documention definition is a subtype of objectInformation, and is derived by *extension*. This is specified as follows:

```
<complexType name="documentation">

        <complexContent>
           <extension base="objectInformation">
           </extension>
        </complexContent>
 </complexType>
```

As shown in the above example, <documentation> extends the content model of objectInformation by combining all element and attribute definitions. Because both documention and its parent type objectInformation are defined with a sequence content model, the new elements defined for documention will be appended to the end of the sequence defined for objectInformation. This capability of XMLSchema enables schemas to be written in a much more object-oriented style.

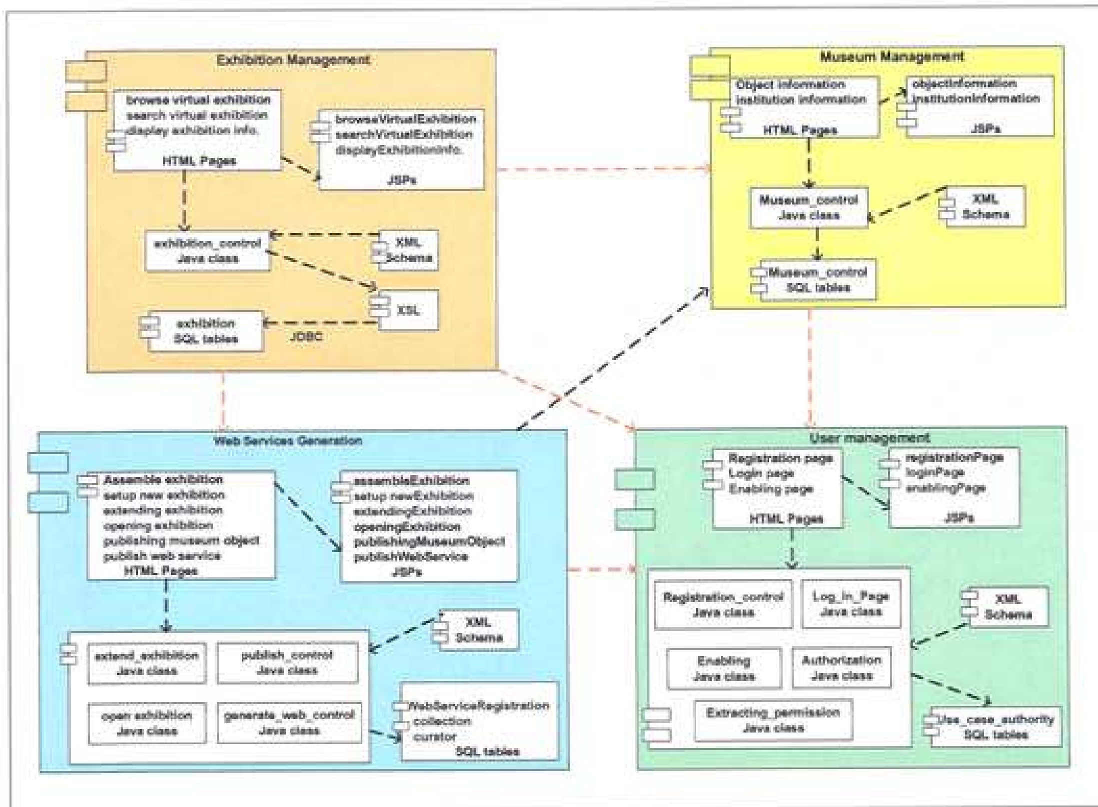
In addition, each element may be optional. The property is implemented in XML by adding a minOccurs="0" attribute to each element declaration. More details on our XMLSchema can be found in Appendix A.

6.5.3 Design of the component diagram

There are four main packages in the component diagram such as exhibition management, museum management, web services generation and user management. First, the *Exhibition Management* package, which contains packages like HTML pages for the interface, JSPs for processing the HTML pages, exhibition control (Java class), XMLSchema, XSL stylesheet and exhibition SQL tables. Second, the *Museum Management* package, which contains components like HTML pages, JSPs, XMLSchema, Museum control (Java classes, SQL tables) as sub-packages. Third, there is the *Web Services Generation*, which contains components like HTML pages, JSPs, XMLSchema, user management control (Java classes, SQL tables) as sub-packages. Finally, there is the *User Management package*, which contains components like HTML pages, JSPs, XMLSchema and web services generation control (Java classes, SQL tables).

The diagram shown in Figure 6.5 shows the usage dependencies as a relationship between packages and which package requires another. The dependency is shown as a dashed arrow and the arrowhead points from the component to the one on which it is dependent. Figure 6.5 shows the component diagram for our proposed work in more detail.

Figure 6.5: Component diagram



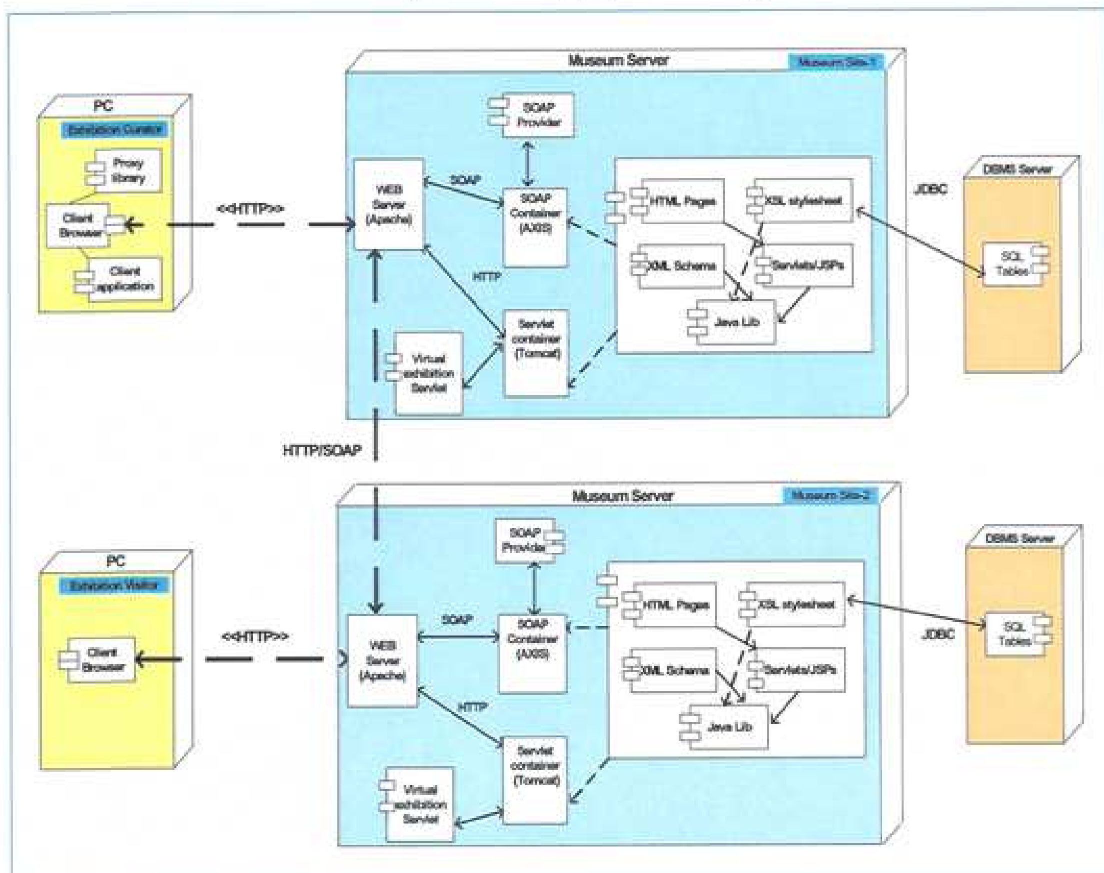
6.5.4 Design of the Deployment diagram

The deployment diagram is used to show the configuration of run-time processing elements and the software components and processes that are located on them. Figure 6.6 shows the deployment diagram for our proposed system. The deployment diagram shows the three nodes such as PC (Exhibition Curator, Exhibition Visitor), Museum Server and DBMS Server that represent the nodes for the client, application server and database server respectively. Furthermore the Museum Server (Application server) represents the node that will process user requests from the Web server and send application responses back to the Web server. The application server node will host the different kinds of the system components such as Servlet container (Tomcat), SOAP container, XSL stylesheets, Java Lib and Web Server. The Web Server node will receive the user requests and send responses from the application to the Client over HTTP protocol. Each of these components deploys some task, for example the XSL stylesheet is used to translate from the XQuery as a user request to an SQL string, from SQL to XML and from XML into an HTML document that can be read by the web browser. The museum server was implemented using the Tomcat Java application server [22].

Tomcat is the reference implementation for the Java servlet technology [57]. Moreover, the Database Server node that will host the database server is used by the components in the application server node to store and retrieve the data used by the System. To be more specific the deployment diagram for the museum system shown in Figure 6.6 is composed of three components:

- Web browser (Client) that can connect to the museum server, i.e., to access the Java servlets, the client can use PCs to run a Java servlet.
- Museum server a set of servers and internal network connecting them. This provides a web server capable of accessing data from DBMS and making it available to the client. Technology choices for the middle-tier include a Web server, Web server with servlets (Tomcat), SOAP container (AXIS), a Virtual exhibition servlet, Java Server Pages, HTML Pages and XSL stylesheets. The communication protocol between the database and the museum server could be JDBC.
- DBMS server with SQL Tables provides database storage.

Figure 6.6: Deployment diagram



6.6 Prototype Implementation of the Integrated System

This part describes the implementation of the integrated system. In the following sections the processes performed by the off-line component presented in Chapter 5 and based on functions of prototype are illustrated by various screens.

6.6.1 Client Browser display of institution's information

The client browser is able to display institutional information. The HTTP client sends an HTTP GET parameters request such as a XQuery For inst in <institution> Return <inst> for display institution information to a Web server, then sends a request to the servlet container tomcat (Step1). The Java servlet will use the XSL stylesheet (XQuerytoSQLTran.xsl) on the server to transform the XQuery user request to SQL query string (Step 2). Once the transformation is done, SQL query string is generated, then the Java servlet connects to the DB and passes the SQL query string to the database server over JDBC (Step 3). The Java servlet class will retrieve the information according to the SQL query string. Then the XSL stylesheet will transform the retrieved data to XML and send the output back to the client (Step 4). The result is shown in Figure 6.7 and again on Tomcat server in Figure 6.8. The source code for the Java servlet can be found in Appendix B.

Figure 6.7: Institutional information displayed

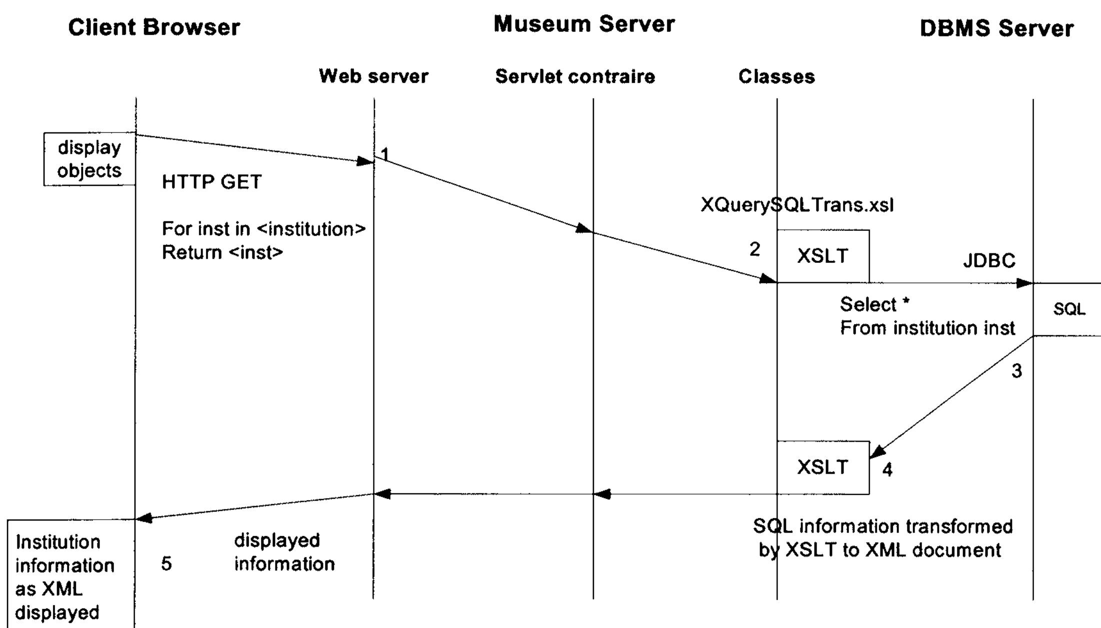
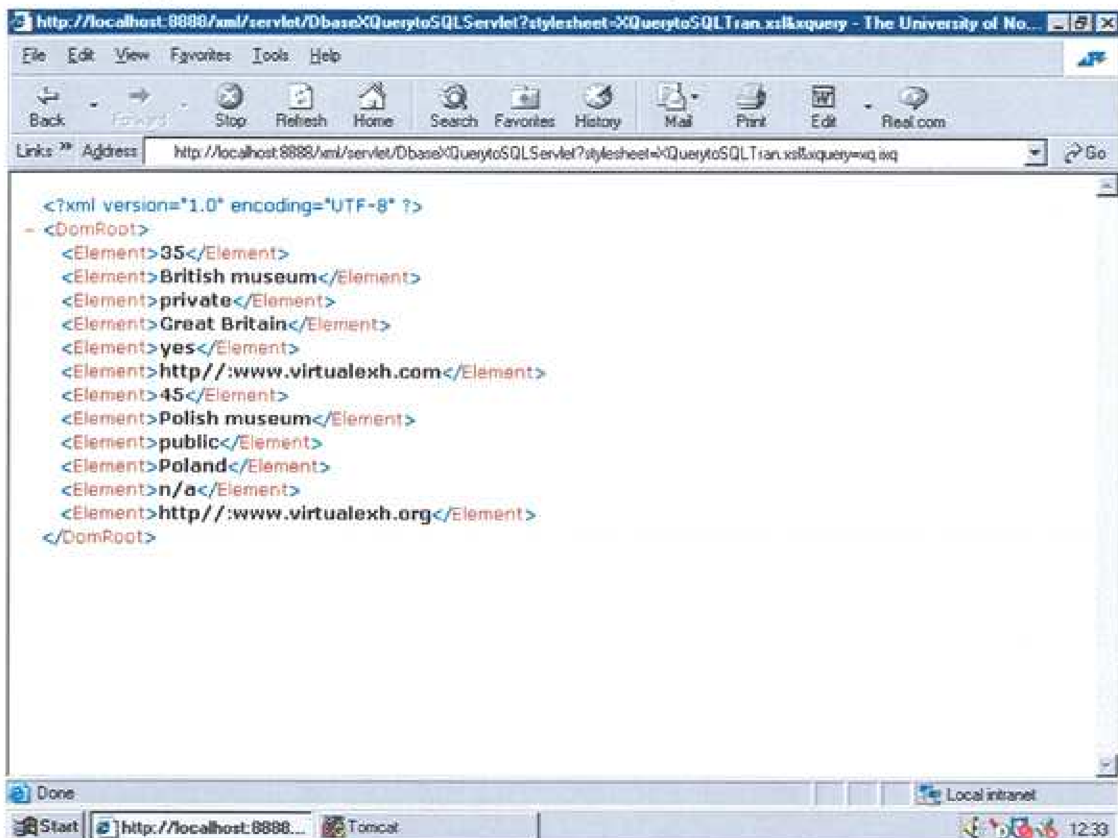


Figure 6.8: Institutional Information displayed on Tomcat server



6.6.2 Client search and display of museum objects

Here the client will search and display object information. The client specifies and sends an XQuery such as For obj in <collection> Return <obj> Where obj/<object_id> > 200 to the Web server as a URL by using HTTP (Step1). The museum server parses and transforms the request by the XSL stylesheet (XQuerytoSQLTran.xsl) on the server and creates a SQL query string (Step 2). Once the transformation is done and the SQL query string generated, then the Java servlet is connected to the DB server and the SQL query passed to the database server over JDBC (Step 3). The query is executed and then the database server returns the report to the museum server. The XSL stylesheet will transform the report to HTML by using the XSL stylesheet (GenXSLHTML.xsl) and sending the output back to the client (Step 4). This result is shown in Figure 6.9 and also on Tomcat server Figure 6.10.

Figure 6.9: Search and display museum objects

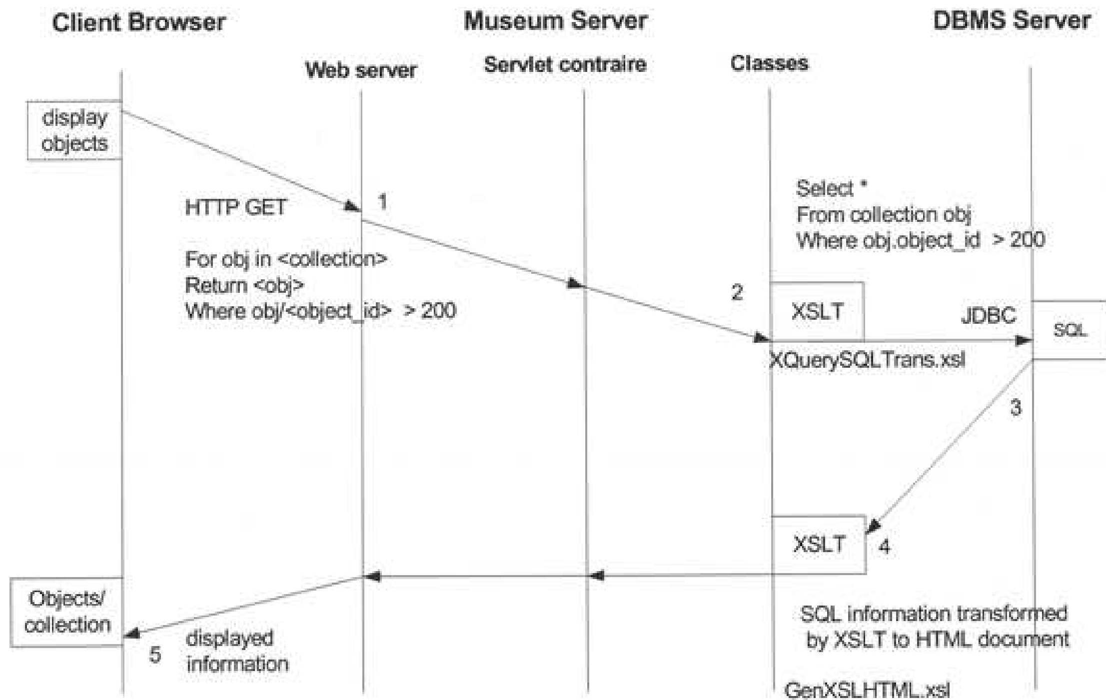
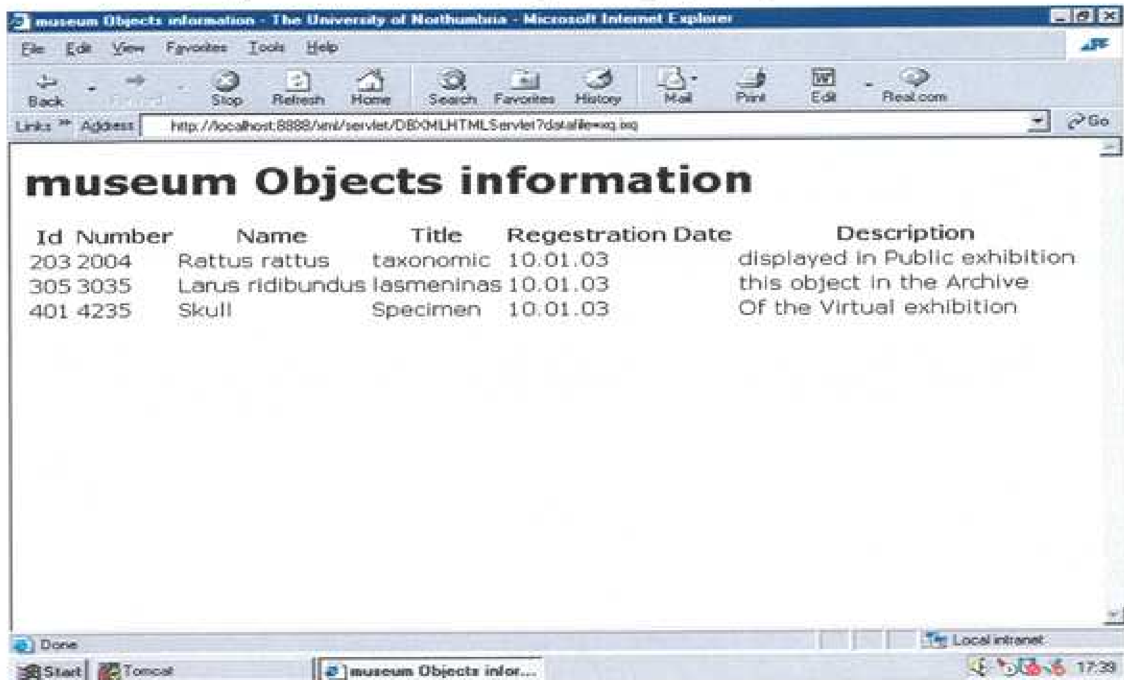


Figure 6.10: Museum objects displayed by the Client



6.6.3 Client browser Search and display of objects information

The client sends an HTTP GET request with the XQuery such as For obj in <collection> Return <object> Where obj/<regis_date> = 10.01.03 to museum server to display the objects have registrations date is equal 10.01.03 (Step1). The XSL stylesheet (XQuerytoSQLTran.xsl) generated in Chapter 5 will transform the XQuery to the SQL query string (Step 2). Once the transformation is done and the SQL query string generated, then the Java servlet is connected to the DB server and the SQL query passed to the database server with JDBC (Step 3). Now the Java servlet retrieves the information according to the SQL query string. Again the XSLT will transform the client request to HTML by using the XSL stylesheet (GenXSLHTML.xsl) and sends the output back to the client (Step 4). This result is shown in Figure 6.11 and again on Tomcat server Figure 6.12. The source codes for Java servlet can be found in Appendix B.

Figure 6.11: Search and display museum object

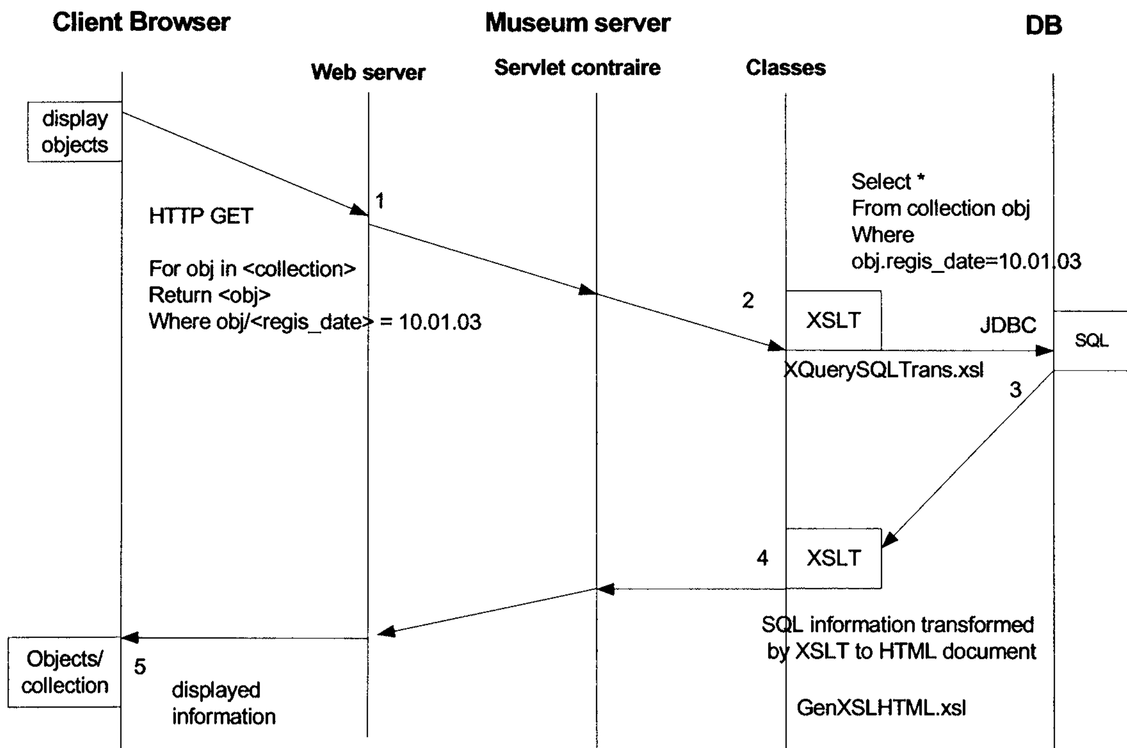
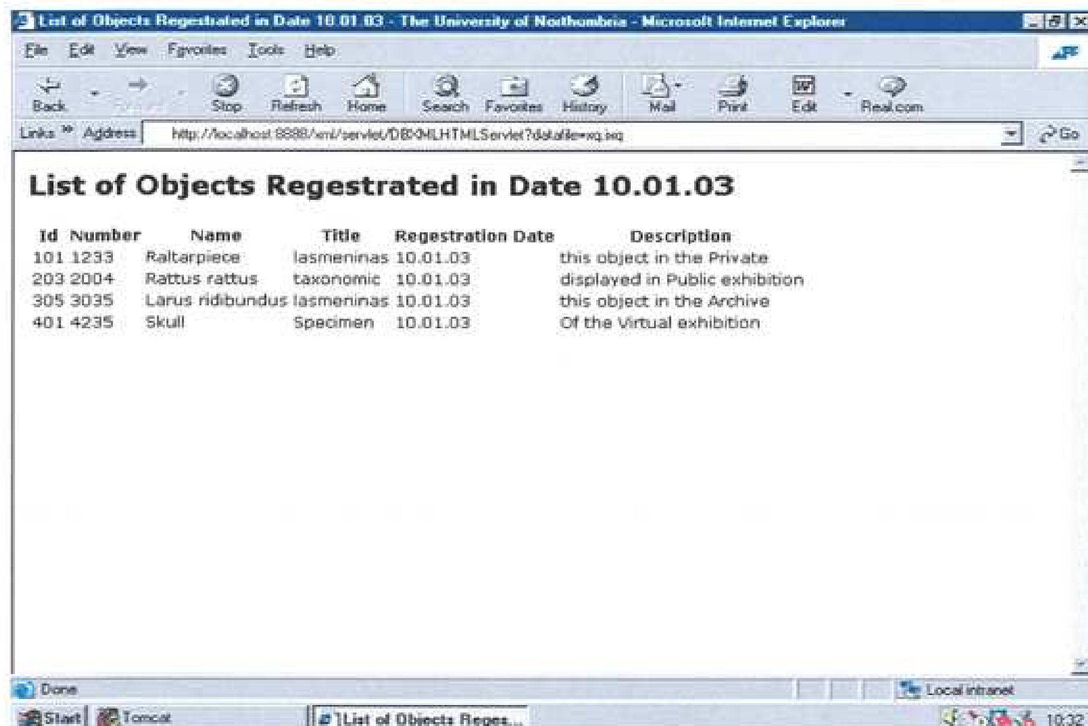


Figure 6.12: Objects with registration date is 10.01.03



6.6.4 Client Search and display of exhibition information

After a successful login the client is able to search and display exhibition information. The client specifies a XQuery like (For exh in <exhibition> Return <exh>) Where exh/<opened> = "yes" to Web server as a URL by HTTP request (Step 1). The Web browser sends the URL to the data server by Java servlet (Step 2). Then the museum server parses the URL request and transforms the XQuery to an SQL query string by using the XSL stylesheet (XQuerytoSQLTran.xsl) (Step 3). The museum server passes the SQL query to the database server over JDBC (Step 4). Then the SQL query is executed and the result produced (Step 5). Again the XSLT will transform the result to HTML by using the XSL stylesheet (GenXSLHTML.xsl) and sends it to the Web server (Step 6), then the Web server sends the output back to the client (Step 7). This is shown in Figure 6.13 and on Tomcat server in Figure 6.14. The source codes for Java servlet can be found in Appendix B.

Figure 6.13: Search and display the opened exhibition

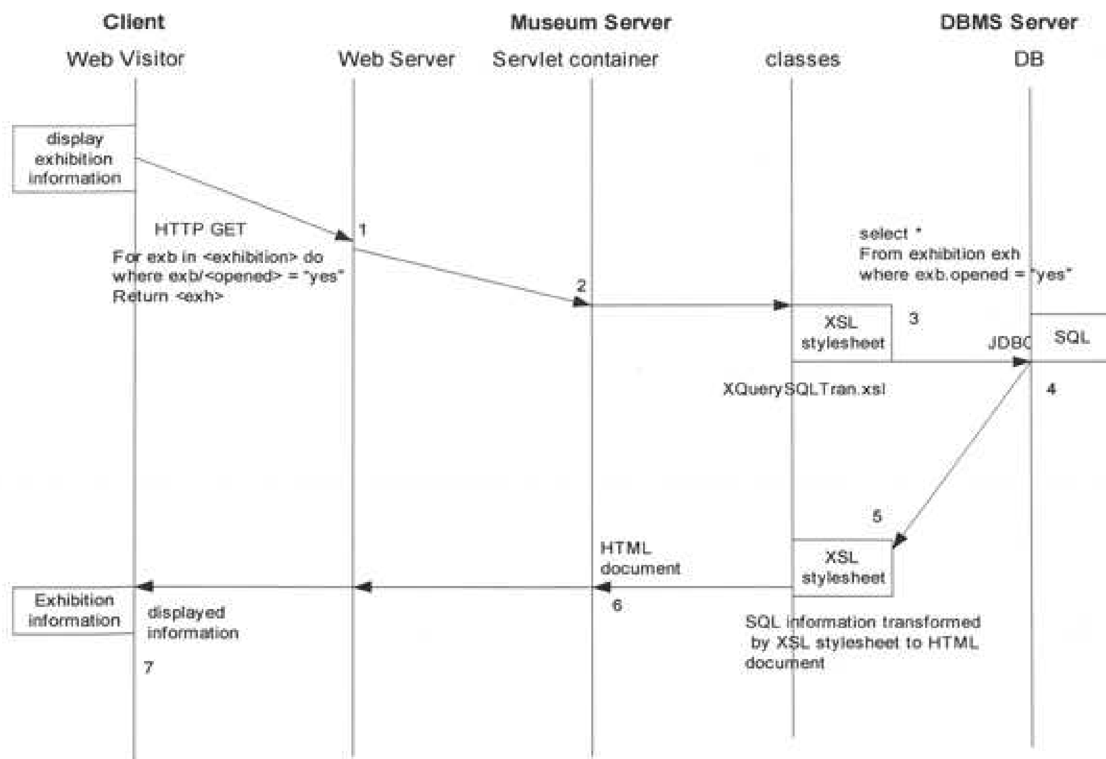
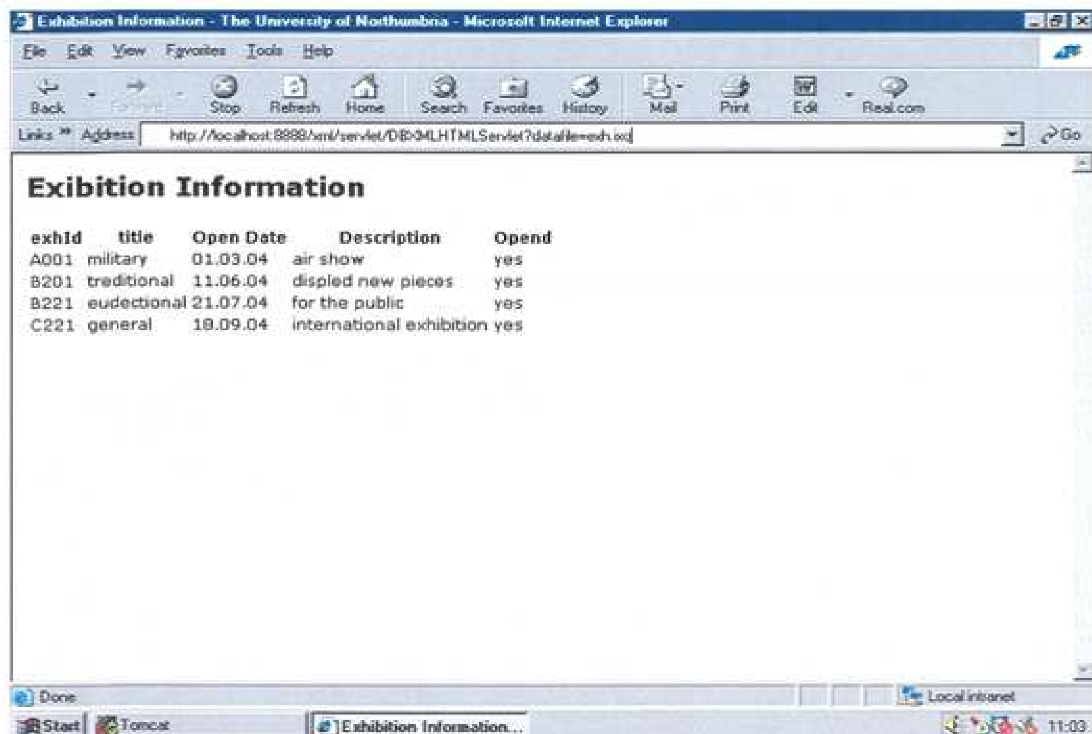


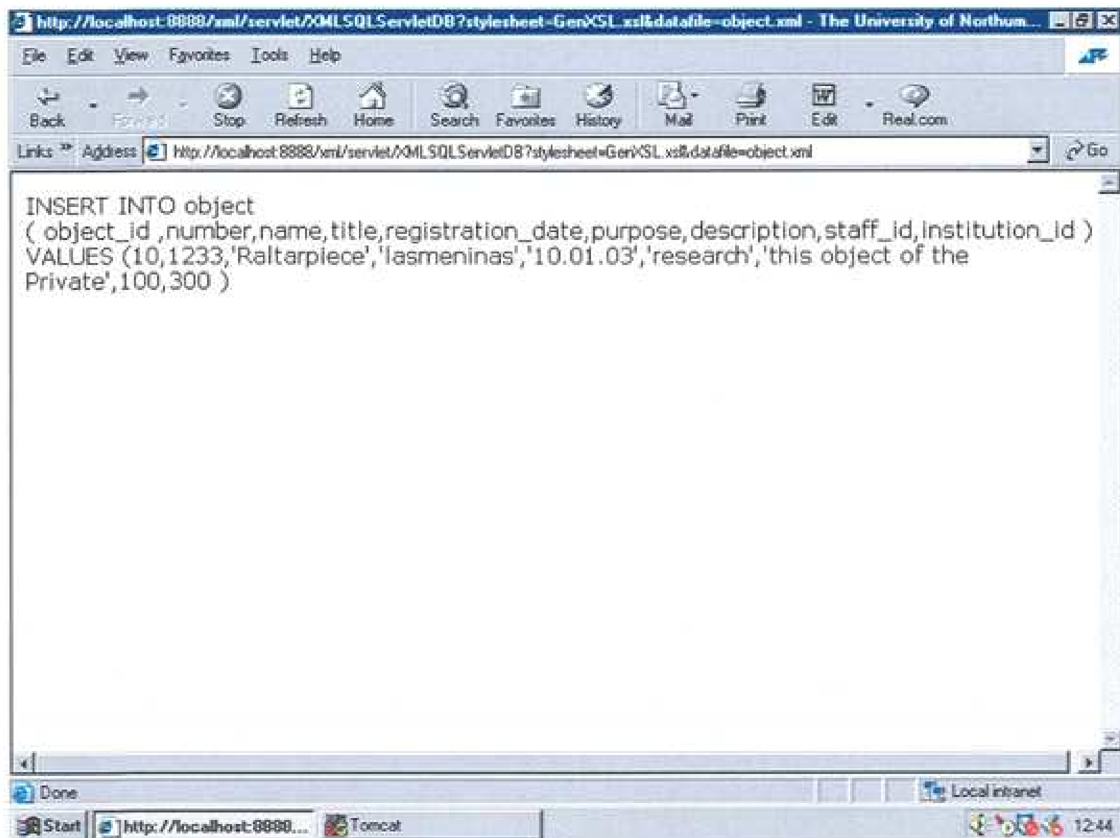
Figure 6.14: Opening exhibition information



6.6.5 Staff stored an objects into database

The staff specifies an XML document and sends it to Web browser as a URL by HTTP request. The Web browser sends the URL to the data server by the Java servlet. Then the museum server parses the URL request and transforms it to an SQL query string by using the XSL stylesheet (GenXSL.xsl). The application server passes the SQL statement to the database server over JDBC to SQL tables. The result is shown in Figure 6.15 on the Tomcat server. The source codes for Java servlet can be found in Appendix B.

Figure 6.15: Storage of objects in database



6.7 Conclusion

As a test framework of our integrated approach we have implemented a prototype system for exchange of information between several independent museums as an example.

The prototype system presented is built entirely using the public domain stack of technologies for processing XML data in Java (J2SE, J2EE and additional XML and Web Services packages). It functions as an entirely server-side Web application executed by Tomcat server connected to a backend database (one for each participating museum plus one for the exhibition itself).

Furthermore the role of offline transformation is to generate all the generic feature of the solution to be developed outside of the integrated system and we use these components to generate specific components of the system. Moreover, we tested the offline components with the online transformation as presented in Chapter 5. Finally, we tested the integrated system and the actual result was as expected. The transformation process for the integrated system is presented in this chapter.

Chapter 7:

Conclusion and further research possibilities

Of course, there are many indications, which could be concluded from this study. The first section of this chapter examines the original contribution made by this dissertation; the second section introduces possibilities for future work.

The results from our literature review gave us a choice of an existing solution for applications in our context of either (1) Lore algebra, adopting an existing standard for building an original solution, (2) the standard XML algebra of W3C, or (3) developing a more targeted solution, that is a domain-specific algebra, to handle our requirements. All the considerations, given earlier in the review, have already been taken into account by the team working on the standard XML algebra (W3C February 2001) and XQuery (W3C 2004, W3C 2005). In fact, the standard algebra is largely based on the AT&T model with some additional features and revisions in the spirit of Niagara. So, the standard is a satisfactory starting point for us in our efforts to develop a domain-specific algebra and was therefore adopted as the basis for our way forward. However, it does not make sense to implement the full algebra defined by the standard for one single specific task or even for a class of similar tasks. Because of this, we have developed a formal data model as a restricted version of the universal algebra suitable for a representative class of problems. Of course, such an approach may lead to a non-universal model, but at the same time, it is feasible and does produce a more effective solution for particular classes of problems. On this basis, we started with the XML Schema (W3C September 2001) of the data to develop a domain-specific XML algebra more suitable for data processing of the specific data and then we used it for implementing the main off-line components of the system. A contribution of this work is that it introduces an algebra that operates on a new data model, because our algebra employs XML trees as data sources and targets. Our algebra framework can be used in integrated architectures for distributed information processing and its components will be XMLSchema-driven. To evaluate the algebra operators we present algorithms for many of the algebra operators and these algorithms have been faithfully implemented in Java.

The result of the algorithms shows how the algebra operators work on nodes of the XML tree or element and attributes of an XML document. At the same time we have introduced many examples to show how the algebraic operators work with results as expected.

General techniques have been introduced for generating SQL schema, XSL stylesheet and XQuery using XMLSchema and XSL stylesheet. A potential cause for concern is that our general techniques may be less overlapping and thus less efficient. This is discussed later. As we know it is possible to code by hand an XSL stylesheet that validates an XML document against some or all constraints of an XMLSchema and to generate an XSL stylesheet. The algorithms introduced have shown how we can generate these components automatically. We also introduce many examples to evaluate the generated components with results as expected. At the same time this part of the research can be used for evaluating the whole work. Demonstrating acceptable pictures even without very precise measures would be sufficient to prove the value of this research. We have been successful in the implementation of the generic approach, which is extremely attractive from a functional point of view. This was presented in Chapter 5.

As a test framework for our integrated approach we have implemented a prototype system for exchange of information between several independent museums for organizing virtual exhibitions over the Web. This was presented in Chapter 6.

The hypothesis in Chapter 1 was that an integrated approach to processing XML data incorporating several principally different functions such as communication, validation, transformation, presentation, storage and retrieval will lead to an increased effectiveness and flexibility in information management. The effective virtual exhibition, built on a framework of an integrated approach, confirms that the hypothesis is correct.

7.1 Originality

The originality can be seen on different levels, such as in the theoretical approach (algebra model) and the technical solution. Furthermore the central idea behind the research is to use a single theoretical model of the data to be processed by the system (XML algebra) and controlled by one standard metalanguage specification (XML Schema) for solving a class of problems (generic architecture). The work has combined theoretical novelty and technological advances in the field of Internet computing. Furthermore, the practicality of the approach was shown by building a prototype for exchange of information between several independent museums as an example. In the following paragraphs more details are given on the contribution.

7.1.1 The first contribution of this thesis: a new domain-specific XML algebra for generic distributed problem solving

We have presented a framework for representing XML algebra and XQueries over XML documents. The algebra model provides formal data semantics for the constructed XMLSchema. The algebraic operators and the algebraic relational operators we have defined explore XML data as well as constructing new XML documents. In other words the algebraic operator works on one tree if it is a unary operator and on two trees if it is a binary operator producing one XML tree as output. The contribution of this chapter is that it introduces an algebra that operates on a new specific data model because the input and output of our algebra works on XML trees. The relational algebraic operators work on XML documents and the XML document defined as a tree.

Furthermore, our XML algebra is generic, since our XML algebra model is XMLSchema-driven. Also our algebra has a clear data structure and a simple presentation of the data. There are two types of operators in our algebra: firstly, the algebraic operators are *join, union, complement, project, select, expose* and *vertex.* Every operator has as output, a tree, which makes it distinct from other algebraic operators. Secondly, the algebra relational operators are *universal, subsuming, equivalence* and *similarity.*

Our algebraic framework can be used in an integrated architecture for distributed information processing and its components will be XMLSchema-driven.

In Chapter 5 we used our XML algebra for implementing the main offline components of the system for data processing (XMLSchema → SQL schema generation, XML Schema→ HTML presentation, XML Schema → XML query interpreter). More details on this contribution were presented in Chapter 4.

7.1.2 The second contribution of this thesis: XML Schema-driven software architecture

The novelty of our approach is that both the integrated architecture for distributed information processing and its components are XMLSchema-driven. In addition, the components generated automatically before the integration are based on the generic domain specific XMLSchema of the algebraic model. In the following sections more details are given on this contribution.

- XML Schema-driven integrated architecture for distributed information processing over the Web. The proposed solution uses a single XMLSchema for all kinds of transformation (online and offline). Details on the online transformation were presented in Chapter 5. The online components transformation is as follows:
  - XML Schema-driven storing (XML → SQL transformation)
    The XML Schema prepares for the storage of XML data to relational databases.
  - XML Schema-driven querying (XQUERY→ SQL transformation)
    In this component all specific queries using SQL (standard query language) are executed against the database, which contains the transformed data.
  - XML Schema-driven generation (SQL → XML transformation)
    XML schema will drive the generation of XML data from the data stored in SQL format. For the purposes of the integrated architecture we will process the XML data, and this is why we retrieve and transform the data in XML format.
  - XML Schema-driven presentation (XML → HTML transformation)
    This is an example of presentation of the XML data as HTML formats.

- XMLSchema-driven generation of architecture components. The single XML Schema will be used in all kinds of off-line transformation. Why do we need this three-generation component? Because this component is related to querying the data and generating the components that we need. Details on the offline techniques can be found in Chapter 5. The generation of the offline components includes the following:
  - SQL schema generation
    This component is related to the generation of the SQL schema of XMLSchema.
  - XSL Stylesheet generation
    This component is related to generation of all online stylesheets such as transforming XML data to SQL, XML data to HTML and so on.
  - The XQuery interpreter generation
    This component is related to querying the data and answering queries.

Further aspects of this research's second contribution are techniques for generating SQL schema, XSL and XQuery, using XMLSchema and XSL stylesheet, which (a) enables us to use these techniques for transforming XML data to SQL data and storing it in the relational database, (b) allows the user to present data in HTML format, and (c) provides interpretation of XQueries, which means transforming XQueries to SQL ones so that we can retrieve and query data from the database.

A potential cause for concern is that our general techniques may be less overlapping. However, based on our prototype implementation in Java, we have found that generating XSL stylesheets as an interpreter for different types of transformation such as XMLSchema to SQL schema and XQueries to SQL queries can be done efficiently and quickly. Whereas it is possible to code by hand an XSL stylesheet that validates an XML document against some or all constraints of XML schema and to generate an XSL stylesheet, using the techniques here it is automatic, easy, quick and there is less overlapping. Details on this contribution were presented in Chapter 5.

7.1.3 The third contribution of this thesis: algorithmic analysis of the algebraic transformations

We have introduced a generic algorithm for relational mapping from a class of XML schemes into relational databases. One consequence of the fact that we use a single unified model (XML Algebra) which is present across the whole technological stack using a single specification (XMLSchema), is that we are allowed to organize all possible transformations and component generations needed to implement the generic architecture around the very simple but extremely powerful idea of using XSL stylesheets for all kinds of algorithmic transformations. Details on this contribution were presented in Chapter 5.

7.1.4 The fourth contribution of this thesis: introducing a test framework of our integrated approach

We have presented the implementation of a prototype system for exchange of information between several independent museums. The prototype demonstrates the practicality of the approach through the development of a framework for exchange of the information.

Moreover, we have tested the offline components with the online transformation as presented in Chapter 5. Finally, we installed the offline component (XSL stylesheets) in the museum server and we used them during the online transformation with the integrated system, as presented in Chapter 6.

7.1.5 The final contribution of this thesis

The final contribution of this work is the publication of the results.
Chapter 4 was published as Paper 1 in the USA in 2005 [155]: A Tree Based Algebra Framework for XML Data Systems, in which we introduced an algebra that operates on a new data model, because our algebra employs XML trees as data sources and targets.
This work has also been published as a technical report, XML Data Systems: A Tree Based Algebra Framework with Worked Examples [156]. This introduces the algebra in more detail and gives worked examples.

Chapter 5 was published as two papers. Paper 2 in the USA in 2005 [153]: XML Schema-driven generation of architecture components, which introduces general techniques as a technological solution for different problems such as (a) generation of SQL schema from XML Schema, (b) generating XSL stylesheets from XML Schema, and (c) XQuery interpreter generating. This work was also published as a technical report [154]: XML Schema driven generation of architecture components and integration of online and offline Systems. This introduces the novelty of our approach in that both the integrated architecture for distributed information processing (online) and generation of architecture components (offline) are XML Schema driven. The online components are XML Schema driven storage (XML → SQL transformation), XML Schema driven querying (XQUERY→ SQL transformation), XML Schema driven generation (SQL → XML transformation) and XML Schema-driven presentation (XML → HTML transformation). The offline components are SQL schema generation, XSL Stylesheet generation and the XQuery interpreter generation. Also, Paper 4 was published in the UK in 2005: Mapping XML document in a variety of formats. This introduces different ways and examples for mapping and storing the XML document in database.

Chapter 6 was published as paper 3 in Croatia in 2005 [157], Virtual Exhibitions Framework: Utilisation of XML Data Processing for Sharing Museum Content over the Web. This presented a prototype of a framework for organising virtual exhibitions, using information provided by collaborating museums in the form of Web services.
In all papers the designs, implementations and papers were written by this author, with Nick Rossiter collaborating in preparing the papers for publication.

7.2 Future Research Possibilities

This research has yielded some interesting and useful results about how XML Schema can be used for the generation of architecture components and also how it can be used for solving a class of problems with a generic architecture. There are many interesting avenues for further work.

7.2.1 Regarding our XML algebra

Although we used a single XML Schema, our experience suggests that implementing this algebra by using more than one XML Schema can lead to significant benefits such as more flexibility and accuracy. In addition there is still further work to be done in order to make the algebra more efficient and accurate.

7.2.2 Regarding the technology solutions

It is still open on how to make the technological solutions more efficient and accurate, particularly if more than one XML Schema is used. The HTTP client comes equipped with a browser that sends requests to an HTTP server and elicits a response in return. Also, in the implementation of the integrated system, we used the client communication protocol and XML, employing HTTP, to access the data. As the time has been limited, this was done with the SOAP protocol as shown in the design in Chapter 6.

7.2.3 Regarding the implementation of the virtual museum

The implementation of the virtual museum joint exhibition on the Web was limited by the time available. We have implemented sufficient screens to show how the client (Web browser) can display or search objects or institutions in the virtual museum. We also show how the staff can store objects in the database server. It would be interesting to complete the implementation of the entire museum system.

Overall, such enhancements will help in improving both sides of the work, the theoretical part (XML algebra) and the technological solutions, thus achieving a more accurate and efficient evaluation.

7.3 The overall achievement of the thesis

The overall achievement of the thesis can be summarized as follows:

- The first achievement of this thesis is that it introduces an algebra that operates on a new data model, as our algebra employs XML trees as data sources and targets. Our algebra framework can be used in integrated architectures for distributed information processing and its components will be XML Schema-driven.
- The second achievement of this thesis is the XML Schema-driven software architecture.

The novelty of our approach is that both the integrated architecture for distributed information processing (online) and generation of architecture components (offline) are XML Schema-driven.

- The third achievement of this thesis is that it introduces general techniques as a technological solution for different problems such as (a) generating SQL schema from XML Schema, (b) generating XSL stylesheet from XML Schema, and (c) XQuery interpreter generating.
- The fourth achievement of this thesis is that it introduces a variety of formats for mapping XML documents such as mapping from tree-to-tree, which means XML to XML and XML to XHTML, as well as mapping XML documents (tree) to relations.
- This thesis also demonstrates the practicality of the approach through the development of a framework for exchange of the information between several independent museums.

Overall the author feels that an improved framework for handling XML data has been designed and the research experience has been very stimulating.

REFERENCES:

[1] W3C, "Extensible Markup Language (XML) 1.0 (Third Edition)", W3C Recommendation, February 04, 2004.
http://www.w3c.org/TR/REC-xml.

[2] Ronald Bourret, "XML and Database", published online at URL http://www.rpbourret.com/xml/XMLAndDatabases.htm, last updated December 2004.

[3] Apache group, Apache Xerces-J (http://xml.apache.org/xerces-j/index.html). The latest version: Xerces 2.0 (beta 3) (October 2001).

[4] Brett McLaughlin and Jason Hunter, "JDOM (http://www.jdom.org) the latest version: JDOM Beta 7," July 7, 2001.

[5] W3C, DOM Working Group, "Document Object Model", published online at URL http://www.w3.org/DOM/, (2004).

[6] Khun Yee Fung, "XSLT Working with XML and HTML", Addison-Wesley, ISBN: 0-201-71103-6, December 2000.

[7] Michael Claßen, "XML Parser Comparison", October 20, 2000. http://www.webreference.com/xml/column22/index.html.

[8] SourceForge project, "SAX 2.0 the Simple API for XML", http://sax.sourceforge.net, April 27 2004.

[9] David Fallside and Priscilla Walmsley, "XMLSchema Part0:" Primer second Edition, World Wide Web Consortium, Proposed Edited Recommendation xmlschema-0-20040318, March 2004.

[10] W3C, "XSL Working Group and XML Linking Working Group, W3C Recommendation on XML Path Language (XPath) Version 1.0", November 16, 1999. http://www.w3.org/TR/xpath.

[11] W3C, "XSL Working Group", W3C Recommendation on XSL Transformations (XSLT) version 1.0, November 1999. http://www.w3.org/TR/xslt.

[12] W3C, "Extensible Stylesheet Language (XSL) Version 1.0", W3C Recommendation, October 15, 2001. http://www.w3.org/TR/2001/REC-xsl-20011015/

[13] SourceForge: www.cs.rpi.edu/~puninj/XMLJ/classes/class5/all.html, 2000.

[14] F. Anklesaria, M. McCahill, P. Lindner, D. Johnson, D. Torrey, and B. Alberti, "The Internet Gopher Protocol: A Distributed Document Search and Retrieval Protocol", RFC 1436, University of Minnesota, March 1993.

[15] T. Berners-Lee, R. Fielding and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax and Semantics", RFC 2396, August 1998.

[16] T. Berners-Lee, and D. Connolly, "Hypertext Markup Language - 2.0", RFC 1866, MIT/W3C, November 1995.

[17] T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform Resource Locators (URL)", RFC 1738, CERN, Xerox PARC, University of Minnesota, December 1994, Berners-Lee, et al., Informational RFC 1945 HTTP/1.0, pp52, May 1996.

[18] K. Moore, "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, November 1996.

[19] J. Robie, J. Lapp, D. Schach, “XML Query Language (XQL)”, See http://www.w3.org/TandS/QL/QL98/pp/xql.html, 2001.

[20] D. Crocker, "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, UDEL, August 1982.

[21] F. Davis, B. Kahle, H. Morris, J. Salem, T. Shen, R. Wang, J. Sui, and M. Grinbaum, "WAIS Interface Protocol Prototype Functional Specification." (V1.5), Thinking Machines Corporation, April 1990.

[22] The Jakarta Project, “Tomcat version 5.5. Reference Implementation for the Java servlet 2.2 and Java Server Pages 1.1”, http://jakarta.apache.org/tomcat/, 2005.

[23] B. Kantor, and P. Lapsley, "Network News Transfer Protocol: A Proposed Standard for the Stream-Based Transmission of News", RFC 977, UC San Diego, UC Berkeley, February 1986.

[24] J. Postel, "Simple Mail Transfer Protocol," STD 10, RFC 821, USC/ISI, August 1982.

[25] Alin Deutch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu, “A Query Language for XML”, See http://www.research.att.com/mff/files/final.html, 2003.

[26] J. Postel, and J. Reynolds, "File Transfer Protocol (FTP)", STD 9, RFC 959, USC/ISI, October 1985.
http://www.w3.org/Protocols/rfc959/.

[27] Steven Pemberton, “XHTML 1.0: The eXtensible HyperText Markup Language (Second Edition)”, World Wide Web consortium, Recommendation REC-xhtml-20020801, August 2002.

[28] K. Sollins, and L. Masinter, "Functional Requirements for Uniform Resource Names", RFC 1737, MIT/LCS, Xerox Corporation, December 1994.

[29] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener, "The Lorel Query Language for semistructured data", International Journal on Digital Libraries, **1**(1): 68-88, April 1997.

[30] W3C, http://www.w3.org/TR/SOAP/: W3C Recommendation June 24, 2003.

[31] W3C, "HTTP - Hypertext Transfer Protocol", http://www.w3.org/Protocols/, last updated 2003.

[32] W3C, http://www.zope.org/Members/Amos/XML-RPC, last modified August 2003.

[33] W3C, www.parish-supply.com/how_soap_works.htm, last updated 2004.

[34] W3C, www.zsplat.freeserve.co.uk/soap/doc/TclSOAP.html, last modified Jun 2001.

[35] W3C, "XQuery 1.0 and XPath 2.0 Data Model", W3C Working Drafts February 11, 2005. www.w3.org/TR/query-datamodel/.

[36] W3C, "XQuery 1.0 and XPath 2.0 Functions and Operators Version 1.0", W3C Working Draft 23 July 2004. www.w3.org/TR/xquery-operators/.

[37] W3C, "XML Information set", www.w3.org/TR/xml-infoset/: last version February 2004.

[38] W3C, "Quilt - XML Query language specification at Don Chamberlain's home page", March 2000.
www.almaden.ibm.com/cs/people/chamberlin/quilt_euro.html

[39] W3C, "XQuery 1.0 Formal Semantics", W3C Working Draft 20 February 2004. www.w3.org/TR/query-algebra/.

[40] W3C, "XQuery 1.0: An XML Query Language", W3C Working Draft July 23, 2004. www.w3.org/TR/xquery/.

[41] W3C, "XML Schema - Part 2 Datatypes - XML Schema language specification describing the Datatypes definition in XML document schema definitions", www.w3.org/TR/xmlschema-2/, October 2004.

[42] J. Shanmugasundarm, et al., "Relational Database for Querying XML Documents: Limitations and opportunities," In Proceeding of the 25th International Conference On Very Large Data Base (VLBD), Edinburgh, Scotland, UK, September 1999.

[43] Feng Tian, David J. DeWitt, Jianjun Chen, and Chun Zhang, "The Design and Performance Evaluations of Various XML Storage Strategies", SIGMOD Record, Volume 31(1), March 2002.

[44] W3C, "The XML Query Algebra", W3C Working Draft February 2001.

[45] S. Cluet, S. Jacqmin, and J. Simeon, "The New YATL: Design and Specifications", Technical Report, INRAI, 1999.

[46] Ronald Bourret, "XML-DBMS - Middleware for Transferring Data between XML Documents and Relational Databases", last updated January 2003. www.rpbourret.com/xmldbms.

[47] Kajal T. Claypool and Elke A. Rundensteiner," Sangam: A Framework for Modeling Heterogeneous Database Transformations", Fifth International Conference on Enterprise Information Systems, Angers, France, April 2003.

[48] M. Klettke, H. Meyer, "Managing XML documents in object-relational databases", Database Research Group, Computer Science Department, University of Rostock, 18051, Rostock, Germany, 1999.

[49] Dongwon Lee, Wesley W. Chu, "CPI: Algorithm for Mapping XML DTD to Relational Schema", University of California Los Angeles USA, July 2001.

[50] Rob Allan, Xiao Dong Wang and Daniel Hanlon, "An Introduction to Web services and related Technology for building an e-Science Grid", Version 0.2, September 22, 2004.

[51] W3C, Technical report and publications, "Web Services Description Language (WSDL) Version 1.2, March 2003, "Document Object Model Level 3 Core, February 2003 and XML Protocol Abstract Model", February 2003.

[52] S. Chawathe, "Describing and Manipulating XML Data", Invited paper, Bulletin of the IEEE Technical Committee on Data Engineering, **22**(3): 3-9, 1999.

[53] R. Goldman, J. McHugh and J. Widom, "From Semistructured Data to XML: Migrating the Lore data model and Query Language," Proceedings of the 2nd International workshop on the Web and Database, Philadelphia, Pennsylvania, pp. 25-30, June 1999.

[54] A. Deutsch, M. Fernandez, A. Levy and D. Suciu, "XML-QL: A Query Language for XML," Proceedings of the International World Wide Web (WWW) Conference, Toronto, May 1999.

[55] W3C, "XML Query Language for XML", W3C Working Draft, February 2001. http://www.w3c.org/TR/Xquery.

[56] J. Shanmugasundaram, E. Shekita, J. Kiernan, R. Krishnamurthy, E. Viglas, J. Naughton and Igor Tatarinov, "A General Technique for Querying XML Documents using a Relational Database System", ACM SIGMOD Record, **30**(3), 2001.

[57] Sun Microsystems, "The Java Servlet specification and Documentation": http://java.sun.com/products/servlet, June 2004.

[58] W3C, "Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language", August 2004.
http://www.w3.org/TR/2004/WD-wsdl20-20040803.

[59] Jams Clark, "Comparison of SGML and XML", W3C website, December 1997.
http://www.w3.org/TR/NOTE-sgml-xml.

[60] Chris Bates, "Web programming: building Internet applications", 2nd edition, Chichester, Wiley, ISBN: 0470843713, 2002.

[61] S. Christ, M. Mulchandani, B. Murphy, E. Rundensteiner, and X. Zhang, "Rainbow: Mapping-Driven XQuery Processing System", International Conference on Management of Data, ACM SIGMOD, pp. 616-614, June 2002.

[62] D. Beech, A. Malhotra, M. Rys, "A formal data model and algebra for XML", W3C XML Query Working Group Note, September 1999.

[63] M. G. Chinwala, R. Malhotra, J. Miller, "Progress Towards Standards for XML Databases," In Proceedings of the 39th Annual ACM Southeast Conference, pp. 227-284, 2001.

[64] E. F. Codd, "A relational model of data for large shared data banks", Communications of the ACM, **13**(6): 377-387, June 1970.

[65] V. Christophides, S. Cluet, J. Simeon, "On wrapping query languages and efficient XML integration," Proceedings of the ACM SIGMOD Conference on Management of Data, pp. 141-152, Dallas, May 2000.

[66] M. Fernanadez, J. Simeon, P. Wadler, "A Semi-Monad for Semi-structured data", International Conference on Database Theory, pp. 263-300, London, Jan 2001.

[67] L. Galanis, et al., "Following the Paths of XML Data: An Algebraic Framework for XML Query Evaluation", Technical Report, University of Wisconsin, Madison, 2001.

[68] J. McHugh and J. Widom, "Query Optimization for XML", In Proceedings of International Conference on Very Large Databases (VLDB), Edinburgh, August 1999.

[69] W3C, "The XML Query Algebra", Working Draft, December 2000. http://www.w3.org/TR/2000/WD-query-algebra-20001204.

[70] W3C, "XML Query Data Model", Working Draft, May 2000. http://www.w3.org/TR/query-datamodel.

[71] W3C, "XQuery 1.0 Formal Semantics", W3C Working Draft February 20, 2004. http://www.w3.org/TR/2001/WD-query-semantics-20010607.

[72] D. Florescu, D. Koaamann, "Storing and Querying XML Data using an RDBMS", IEEE Data Engineering Bulletin, **22**(3):27-34, September 1999.

[73] J. McHugh and J. Widom, "Query optimization for semi-structured data", Technical report, Stanford University, Database Group, August 1998. http://www-db.standford.edu/pub/papers/qo.ps.

[74] T. Lahiri, S. Abiteboul, and J. Widom, "Ozone: Integration structured and Semi-structured data," Proceedings of the Seventh International Conference on Database Programming Languages, Kinloch Rannoch, Scotland, September 1999.

[75] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy and Dan Suciu, "A Query Language for XML", Computer Networks, Vol. 31, pp. 1155-1169, 1999.

[76] E. F. Codd, "Relational completeness of database sublanguages", In Courant Computer Science Symposium **6** on Database Systems, R. Rustin, Ed., pp. 65-98, 1971.

[77] W3C, the XML Query Algebra, Working Draft, February 2001.
http://www.w3.org/TR/2001/WD-query-algebra-20010215

[78] J. McHugh, S. Abiteboul, R. Goldman, D. Quass and J. Widom, "Lore: A Database Management System for Semistructured Data", SIGMOD Record, **26**(3):54-66, September 1997.

[79] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and Janet L. Wiener, "The Lorel Query Language for Semistructured Data", Int. J. on Digital Libraries **1**(1):68-88, 1997.

[80] Mark A. Roth, Herry F. Korth, Abraham Silberschatz, "Extended Algebra and Calculus for Nested Relational Databases," ACM Transactions on Database Systems (TODS), **13**(4): 389-417, December 1988.

[81] J. Bosak, T. Bray, D. Connolly, E. Malor, G. Nicol, C.M. Sperberg-McQueen, "W3C XML Specification DTD," www.w3.org/XML/1998/06/xmlspec-report.htm, 1998.

[82] C. Beeri, Y. Kornatzky, "Algebraic optimization of object-oriented query languages", Theoretical Computer Science **116**(1&2): 59-94, August 1993.

[83] W3C, "XQuery 1.0: An XML Query Language", W3C Working Draft November 12, 2003.
http://www.w3.org/TR/2003/WD-xquery-20031112/.

[84] X. Zhang and E. A. Rundensteiner, "XML Algebra for the Rainbow System", Technical Report WPI-CS-TR-02-24, Worcester Polytechnic Institute, July 2002.

[85] Peyton Jones, "The Implementation of Functional Programming Languages", Prentice Hall, International Series in Computer Science, Prentice-Hall, New York, 1987.

[86] C. Beeri and Y. Tzaban, "SAL: An algebra for Semi-structures Data and XML", ACM SIGMOD workshop on the Web and Database, pp. 37-42 Philadelphia, PA, June 1999.

[87] W3C, "XQuery 1.0 and XPath 2.0 Full-Text", W3C Working Draft July 09, 2004. http://www.w3.org/TR/2004/WD-xquery-full-text-20040709/.

[88] J. Doner, "Tree acceptors and some of their applications", Journal of Computer and System Sciences Volume **4,** pp. 406-451, 1971.

[89] M. H. Scholl, "Theoretical Foundations of Algebraic Optimization Utilization unnormalized relation", in: G. Ausiello, P. Atzeni (Eds.), ICDT'86, Volume **234** of Lecture Notes in Computer Science, Springer Verlag, pp. 409-420, 1986.

[90] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and S. Tommasi, "Tree Automata Techniques and Applications", Available on: http://www.grappa.univ-lille3.fr/tata, 1997.

[91] D. Chamberlin, J. Clark, D. Florescu, J. Robie, J. Simeon and M. Stefanescu, "XQuery 1.0: An XML Query language", W3C working Draft 07 June 2001.

[92] Sun Microsystems: Enterprise JavaBeans Specification, Version 3.0, Sun Microsystems, Inc., June 28, 2004.

[93] M. Fenkhauser, J. Simeon, and P. Wadler, "An algebra for XML Query", In Foundations of Software Technology and Theoretical Computer Science, New Delhi, December 2000.

[94] C. Zhang, J. Naughton, D. DeWitt, Q. Luo and G. M. Lohman, "On Supporting Containment Queries in Relational Database Management Systems", In Proceedings SIGMOD, 2001.

[95] Kurt Cagle, Mark Fussell, Nalleli Lopez, Dan Maharry and Rogerio Saran, "Early Adopter XQuery", ISBN: 1-861006-95-0, January 2002.

[96] Sushant Jain, Ratul Mahajan, and Dan Suciu, "Translating XSLT programs to Efficient SQL queries", In Proceedings of the eleventh international conference on World Wide Web, pages 616-626, 2002.

[97] Ronald Bourret, "Mapping W3C Schemas to Object Schemas to Relational Schemas", March 2001.
www.rpbourret.com/xml/SchemaMap.htm.

[98] H. L. Lau and W. Ng, "Storing XML Documents in Nested Relational Sequence Relations", Research Note, The Hong Kong University of Science and Technology, 2002.

[99] Brett McLaughlin, "Java & XML, 2nd Edition: Solutions to Real-World Problems", O'Reilly & Associates, ISBN: 0596001975, September 2001.

[100] Manolescu Ioana, Florescu Daniela and Kossmann Donald, "Pushing XML Queries inside Relational Databases", Donald Kossmann, INRIA, Technical Rapport No. 4112, 41 pages, January 2001.

[101] W3C, "XMLSchema": At http://www.w3.org/xml/schema, June 24, 2004.

[102] Crofts, Nicholas, "Combining data sources – prototype applications developed for Geneva's department of historical sites and monuments based on the CIDOC CRM", 2003.

[103] David Carlson, Foreword by Jeffrey Hammond, "Modelling XML Applications With UML Practical e-Business Applications", Rational Software Corporation, The Addison-Wesley object technology series, ISBN: 0-201-70915-5, First printing April 2001.

[104] Carol Britton, "Object-Oriented Systems Development: a gentle introduction", University of Hertfordshire and Jill Doake Anglia Polytechnic University, ISBN: 0077095448, 2000.

[105] Ian Sommerville, "Software Engineering", Sixth Edition, Addison-Wesley publishing Ltd. ISBN: 020139815X, August 11, 2000.

[106] J. Hou, Y. Zhang and Y. Kambayashi, "Object-Oriented Representation for XML data," The Third International Symposium on Cooperative Database Systems for Advanced Application, Beijing, China, April 2001.

[107] Vassil Vassilev, Ivan Stoev, Bisserk Gaydarska, Stefan Alexandrov, Georgi Nehrizov and Mihail Vaklion, "Museum Information System: CIDOC data model implementation in the Arch Terra Project", Boland, CILEA No. 69, Sept 1999.

[108] Trace Galloway, "Principles of XML Schema Design", USA, May 2001, Inc Beverly, XML conference & Exposition, December 8-13, 2002, Baltimore convention centre, Baltimore, MD. USA, World Wird Web Consortium, "XMLSchema", at: http://www.w3.org/xml/schema, as accessed 24 June 2004.

[109] Yuan Wang, David J. DeWitt and Jin-Yi Cai, "X-Diff: A fast change detection algorithm for XML documents", University of Wisconsin-Madison, Computer Sciences Department, USA, 2002.

[110] Klaus Bergner, Andreas Rausch and Marc Sihling, "Using UML for Modelling a Distributed Java Application," Institute for Informatics, July 1997. http://www4.informatik.tu-muenchende.

[111] J. Conallen, "Modelling Web Applications Architectures with UML", Comm. ACM, Vol. **42**, No. 10, pp. 63-70, 1999.

[112] ICOM-CIDOC, "Introduction to the International Committee for Documentation of the International Council of Museums (ICOM-CIDOC)", 1996-2003. http://www.willpowerinfo.myby.co.uk/cidoc/cidoc0.htm#English.

[113] Kajal T. Claypool and Elke A. Rundensteiner, "Sangam: A Transformation Modeling Framework", Technical Report TR02-8, Computer Science Department, UMass-Lowell, Kyoto, Japan, March 2003.

[114] Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt, "A language for Bi-Directional Tree Transformations", Technical Report MS-CIS-03-08, Department of Computer and Information Science, University of Pennsylvania, August 2003.

[115] Elisa Bertino, Barbara Catania, "Integrating XML and Databases", IEEE Internet Computing, V. 5, No. 4, pp. 84-88, July 2001.

[116] W3C, "XML Schema Requirements", W3C Note, Feb 15, 1999. http://www.w3.org/TR/1999/NOTE-xml-schema-req-19990215.

[117] Peter Abiteboul, "Semi-structured data," In Proceeding of the AXM SIGACT SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), pages 117-121, Tucson, Arizona, 1997.

[118] Yannis Papakonstantantinou, Hector Garcia-Molina and Jennifer Widom, "Object Exchange Across Heterogeneous", Information Sources, IEEE International Conference on Data Engineering (ICDE), pages 251-260, Taipei, Taiwan, 1995.

[119] Hiroshi Maruyama, Kent Tamura and Naohiko Uramoto, "XML and Java, Developing Web application", 1st Edition, Addison-Wesley, ISBN: 0201485435, May 1999.

[120] N. Klarlund, A. Moller, M.I. Schwatzbach, "DSD: A Schema Language for XML", In ACM SIGSOFT Workshop on Formal Methods in software Practice, Portland, pp. 101-111, Aug 2000.

[121] CIDOC, International Council of Museums, International Committee for Documentation (CIDOC) (1995a), "CIDOC Relational Data Model", CIDOC Data Model Working Group: Washington, D.C, 1995.

[122] CIDOC, International Council of Museums, International Committee for Documentation (CIDOC) (1995c), Draft International Core Data Standard For Archaeological Sites and Monuments, English and French versions, Archaeological Sites Working Group, International Council of Museums, 1995.

[123] D. A. Roberts, "Terminology for museums," Proceedings of an International Conference, held in Cambridge, England, 1988, Edited by D. A. Roberts, Cambridge: Museum Documentation Association, 1990.

[124] Sangho Ha and Kyoungera Kim, "Mapping XML documents to the object-relational form," In Proceedings of the International Symposium on Industrial Electronics, June 2001.

[125] SUN Microsystems, INC., "Java Language Specification", 2004. http://Java.sun.com.docs/books/jis/second-edition/html/i.title.doc.html.

[126] A. Banifati and S. Ceri, "Comparative analysis of five XML queries Language", ACM SIGMOD Record, **29**(1):68-97, March 2000.

[127] P. P. Chen, "The entity-relationship model: Toward a unified view of data", ACM Transactions on Database Systems, **1**(1): 9-39, 1976.

[128] D. Florescu and D. Kossmann, "A performance evaluation of alternative mapping schemes for storing XML data in a relational database", Technical report No. 3684, INRIA, France, May 1999.

[129] Steven DeRose, W3C QL98, "XQuery: A unified syntax for linking and querying general XML documents", October 1998.

[130] P. Bohannon, J. Freire, P. Roy, J. Simeon, "From XML Schema to Relations: A Cost-Based Approach to XML Storage", In ICDE, 2002.

[131] D. Florescu, D. Kossmann, "XML Query Processing", Tutorial for the IEEE Int. Conference on Data Engineering, Boston, USA, March 2004.

[132] H. V. Jagadish, Laks V. S. Lakshmanan, D. Srivastava and K. Thompson, "TAX: A Tree Algebra for XML", DBPL Conference, pp. 149-164, 2001.

[133] M. Ramanath, J. Freire, J. Haritsa and P. Roy, "Searching for efficient XML-to-relational mappings," In Proceedings of the International XML Database Symposium (XSym), 2003.

[134] R. Krishnamurthy, Venkatesan T. Chakaravarthy, R. Kaushik and F. Jeffrey, "Naughton: Recursive XML Schemas, Recursive XML Queries, and Relational Storage: XML-to-SQL Query Translation", ICDE, 2004.

[135] Francis Norton, "Generating XSL for Schema validation", Communication W3C, May 20, 1999.

[136] BRAGANHOLO Vanessa, DAVIDSON Susan, HEUSER Carlos, "From XML View Updates to Relational View Updates: old solutions to a new problem," In Proceedings of International Conference on Very Large Data Bases (VLDB), Pages 276-287, Toronto, Canada, 2004.

[137] D. DeHaan, D. Toman, M. Consens, and M. T. Ozsu, "A Comprehensive XQuery to SQL Translation using Dynamic Interval Encoding," In Proceedings of ACM International Conference on Management of Data (SIGMOD'03), pages 623-634, San Diego, June 2003.

[138] David Peterson, Paul V. Biron, and Ashok Malhotra, "XML Schema 1.1 Part 2: Datatypes", World Wide Web Consortium, Working Draft WD-xmlschema11-2-20040716, July 2004.

[139] Johnny Stenback and Andy Heninger, "Document Object Model (DOM) Level 3 Load and Save Specification", World Wide Web Consortium, Recommendation REC-DOM-Level-3-LS-20040407, April 2004.

[140] Anders Berglund, "Extensible Stylesheet Language (XSL) Version 1.1", World Wide Web consortium, Working Draft WD-xsl 11-20031217, December 2003.

[141] Zohra Bellahsene, Tova Milo, Michael Rys, Dan Suciu and Rainer Unland, "Database and XML Technologies", Second International XML Database Symposium, XSym 2004, Toronto, Canada, August 29-30, 2004.

[142] D. Barbosa, J. Freire, and A. Mendelzon, "Information preservation in XML-to relational mappings," In Proceedings of the International XML Database Symposium (XSym), pages 66–81, 2004.

[143] Mary Holstege and Asir, Vedamuthu, "XMLSchema: Component Designators", World Wide Web Consortium, Working Draft WD-xmlschema-ref-20040716, July 2004.

[144] K. Runapongsa and J. M. Patel, "Storing and querying XML data in object relational DBMS," In Proceedings of the International Conference on Extending Database Technology (EDBT), 2002.

[145] Y. Chen, S. B. Davidson, and Y. Zheng, "Constraints preserving schema mapping from XML to relations," In Proceedings of the Workshop on Web and Databases (WebDB), pages 7–12, 2002.

[146] S. Davidson, W. Fan, C. Hara, and J. Qin, "Propagating XML constraints to relations," In Proceedings of the International Conference on Data Engineering (ICDE), 2003.

[147] Catriel Beeri and Tova Milo, "Schemas for integration and translation of structured and semi-structured data", ICDT 99, January 1999.

[148] Stelios Paparizos, Yuqing Wu, Laks V. S. Lakshmanan and H. V. Jagadish, "Tree logical classes for efficient evaluation of XQuery", Proceedings of the 2004 ACM SIGMOD international conferences on Management of data, Paris, France, June 13-18, 2004.

[149] Ronald Bourret, "XML Database Products":
http://www.rpbourret.com/xml/XMLDatabaseProds.htm, last updated January 1, 2005.

[150] W3C, "XML Query Use Cases", W3C Working Draft February 11, 2005.
http://www.w3.org/TR/2005/WD-xquery-use-cases-20050211/.

[151] Fraenkel J. R. and Wallen N. E.(1993), "How to design and evalute research in eudcation", New York, McGraw-Hill Inc, 1993.

[152] Mckoy F., Vangas-Hernandez N., Summers J. D. and Shah J. J., "Experimental evaluation of engineering design representations for idea generation", in Design Engineering Technical Conferences, 2001, DETC-2001, pp. DTM-21685.

[153] Elbekai Ali and Rossiter Nick, "XML Schema-Driven Generation of Architecture Components", Web Services: Modelling, Architecture and Infrastructure - WSMAI 2005, 7th ICEIS, Florida, USA, 159-164 (2005); republished as p.137-142 in selected papers from Workshop.

[154] Elbekai Ali and Rossiter Nick, "XML Schema-Driven Generation of Architecture Components and Integration of online and offline Systems", Technical Report, School of Informatics, Northumbria University 12pp (2005).

[155] Elbekai Ali and Rossiter Nick, "A Tree Based Algebra Framework for XML Data Systems", 7th ICEIS, Florida, USA, 25-28 May, 305-312 (2005).

[156] Elbekai Ali and Rossiter Nick, "XML Data Systems: A Tree Based Algebra Framework with Worked Examples", Technical Report, School of Informatics, Northumbria University 11pp (2005).

[157] Elbekai Ali, Rossiter Nick, and Vassilev Vassil, "Virtual Exhibitions Framework: Utilisation of XML Data Processing for Sharing Museum Content over the Web", Proc CIDOC Annual Conference 2005, Zagreb, Croatia, 24-27 May, CD-ROM, paper 7, 16pp (2005).

APPENDICES:

APPENDIX A:

OUR XMLSCHEMA FOR THE MUSEUM SYSTEM

Our XMLSchema for the museum system

```
<?xml version="1.0"?>
<XMLSchema>
<!-- ************************************************************ -->
<!-- **  S T A R T   O F  C O L L E C T I O N   C L A S S   ** -->
<!-- ************************************************************ -->
  <element name="collection" type="collection"/>
  <complexType name ="collection">
     <element name="collection_id" type="Integer"/>
     <element name="num" type="Integer"/>
     <element name="name" type="String"/>
     <element name="title" type="String"/>
     <element name="publishing_Date" type="Date"/>
     <element name="remote" type="String"/>
     <element name="features" type="String"/>
     <element name="purpose" type="String"/>
     <element name="description" type="String"/>
     <element name="url" type="String"/>
     <element name="exhibition_id" type="Integer"/>
     <element name="institution_id" type="Integer"/>
     <element name="staff_id" type="Integer"/>
     <all>
     <element name="collection.member_of" minOccurs="0">
          <complexType>
             <element ref="object"/>
          </complexType>
     </element>
     <element name="collection.represented_by" minOccurs="0">
          <complexType>
             <element ref="exhibition"/>
          </complexType>
     </element>
     <element name="collection.generated_by" minOccurs="0">
          <complexType>
             <element ref="staff"/>
          </complexType>
     </element>
     <element name="collection.Publish_by" minOccurs="1">
          <complexType>
             <element ref="institution"/>
          </complexType>
     </element>
    </all>
  </complexType>

  <!-- ////E N D   O F   C O L L E C T I O N   C L A S S///// -->

  <!-- **  S T A R T   O F   O B J E C T     C L A S S   ** -->
    <element name="object" type="object"/>
    <complexType name= "object">
        <element name="object_id" type = "Integer"/>
        <element name="number" type = "Integer"/>
        <element name="name" type = "String"/>
        <element name="title" type = "String"/>
        <element name="registration_date" type = "Date"/>
        <element name="description" type = "String"/>
        <element name="staff_id" type = "Integer"/>
        <element name="institution_id" type = "Integer"/>
```

```
<all>
  <complexType>
    <choice minOccurs ="1" maxOccurs="unbounded">
      <element ref="objectInformation"/>
    </choice>
  </complexType>
  <complexType>
    <choice minOccurs ="0" maxOccurs="unbounded">
      <element ref="collection"/>
    </choice>
  </complexType>
  <element name="Object.registered_by" minOccurs="1">
    <complexType>
      <element ref="staff"/>
    </complexType>
  </element>
  <element name="Object.owned_by" minOccurs="1">
    <complexType>
      <element ref="institution"/>
    </complexType>
  </element>
</all>
</complexType>

<!-- /////// E N D   O F   O B J E C T   C L A S S ///////////--------------->

<!-- ** S T A R T   O F   O B J E C T I N F O R M A T I O N   C L A S S -->
<complexType name= "ObjectInformation">
    <element name="objectInformation_id" type="Integer"/>
    <element name="description_date" type="Date"/>
    <element name="description" type="String"/>
    <element name="object_id" type="Integer"/>
    <element name="Institution_id" type="Integer"/>
    <element name="collection_id" type="integer"/>
    <element name="staff_id" type="Integer"/>
    <element name="expert_id" type="Integer"/>
    <all>
      <complexType>
        <choice minOccurs ="1" maxOccurs="unbounded">
          <element ref="object"/>
        </choice>
      </complexType>
      <element name="owned_by" minOccurs="1">
        <complexType>
          <element ref="institution"/>
        </complexType>
      </element>
      <element name="objectInformation.described_by" minOccurs="1">
        <complexType>
          <element ref="expert"/>
        </complexType>
      </element>
      <element name="objectInformation.approved_by" minOccurs="1">
        <complexType>
          <element ref="staff"/>
        </complexType>
      </element>
```

```
    <element name="objectInformation.availabled_in" minOccurs="1">
      <complexType>
        <element ref="address"/>
      </complexType>
    </element>
  </all>
</complexType>

<!-- ************************************************************ -->
<!-- *            Ref : imageInformation                        * -->
<!-- ************************************************************ -->

<complexType name= "imageInformation">
  <complexContent>
    <extension base="objectInformation">
        <element name="img_id" type="String"/>
        <element name="fileFormat" type="String"/>
        <element name="fileName" type="image"/>
        <element name="fileDirectory" type="String"/>
        <element name="fileObject" type="BLOB"/>
        <element name="url" type="URL"/>
        <element name="objectInformation_id" type="Image"/>
    </extension>
  </complexContent>
    <element name="imageInformation.pictured_by" minOccurs="0">
        <complexType>
          <choice minOccurs="1" maxOccurs="unbounded">
            <element ref="external"/>
          </choice>
        </complexType>
    </element>
</complexType>

<!-- ************************************************************ -->
<!-- *            Ref : Documentation                           * -->
<!-- ************************************************************ -->
<complexType name= "doumentation">
  <complexContent>
    <extension base="objectInformation">
        <element name="docmentation_id" type="Integer"/>
        <element name="fileFormat" type="String"/>
        <element name="fileName" type="String"/>
        <element name="fileDirectory" type="String"/>
        <element name="fileObject" type="BLOB"/>
        <element name="url" type="URL"/>
        <element name="objectInformation_id" type="Integer"/>
    </extension>
  </complexContent>
  <element name="Documentation.authored_by" minOccurs="1">
        <complexType>
          <choice minOccurs="0" maxOccurs="unbounded">
            <element ref="external"/>
          </choice>
        </complexType>
  </element>
</complexType>
```

```
<!-- ****************************************************** -->
<!-- *     Ref : referenceInformation                   * -->
<!-- ****************************************************** -->
<complexType name= "referenceInformation">
  <complexContent>
    <extension base="objectInformation">
        <element name="ref_id" type="Integer"/>
        <element name="title" type="referenceType"/>
        <element name="author" type="String"/>
        <element name="reference_Type" type="String"/>
        <element name="publishing_status" type="String"/>
        <element name="publication_date" type="Date"/>
        <element name="publisher" type="String"/>
        <element name="url" type="String"/>
        <element name="objectInformation_id=" type="Integer"/>
    </extension>
  </complexContent>
</complexType>

<!-- ****************************************************** -->
<!-- *     Ref : collectionInformation                  * -->
<!-- ****************************************************** -->
<complexType name= "collectionInformation">
  <complexContent>
    <extension base="objectInformation">
        <element name="collectionInformation_id" type="Integer"/>
        <element name="collectionDate"  type="referenceType"/>
        <element name="collectionMethod"  type="collectionMethod"/>
        <element name="objectInformation_id" type="Integer"/>
    </extension>
  </complexContent>
 <element name="collectionInformation.passed_by" minOccurs="0">
      <complexType>
        <choice minOccurs="1" maxOccurs="unbounded">
          <element ref="Collector"/>
        </choice>
      </complexType>
  </element>
</complexType>

<!-- ******************************************************************************** -->
<!-- ********** S T A R T   O F  E X H I B I T I O N   C L A S S ****** -->
<!-- ******************************************************************************** -->
<complexType name= "exhibition">
    <element name="exhibition_id" type = "Integer"/>
    <element name="title" type = "String"/>
    <element name="Opning_date" type = "Date"/>
    <element name="describtion" type  = "Text"/>
    <element name="opened" type = "String"/>
    <element name="curator_id" type = "Integer"/>
    <element name="institution_id" type = "Integer"/>
  <all>
    <element name="exhibition.hosted_by" minOccurs="0">
      <complexType>
        <choice minOccurs="0" maxOccurs="unbounded">
          <element ref="institution"/>
        </choice>
      </complexType>
    </element>
```

```
      <element name="represented_by" minOccurs="0">
        <complexType>
          <choice minOccurs="1" maxOccurs="unbounded">
            <element ref="collection"/>
          </choice>
        </complexType>
      </element>
      <element name="arrange" maxOccurs="1">
        <complexType>
          <choice minOccurs="1" maxOccurs="unbounded">
            <element ref="Curator"/>
          </choice>
        </complexType>
      </element>
    </all>
  </complexType>
  <!-- ***************************************************  -->
  <!-- *   P U B L I C   D I S P L A Y   C L A S S    * -->
  <!-- ***************************************************  -->
  <complexType name= "publicDisplay">
    <complexContent>
      <extension base="exhibition">
         <element name="dsp_id" type ="String"/>
         <element name="closing_date" type ="Date"/>
         <element name="exhibition_id" type ="String"/>
      </extension>
      <all>
       <element name="visited_by" minOccurs="0">
         <complexType>
           <choice minOccurs="0" maxOccurs="unbounded">
             <element ref="Visiting_group"/>
           </choice>
         </complexType>
       </element>
      </all>
    </complexContent>
  </complexType>
  <!-- *************************************************** -->
  <!-- *      A R C H I V E   C L A S S           * -->
  <!-- *************************************************** -->
  <complexType name= "Archive">
    <complexContent>
      <extension base="exhibition">
         <element name="arch_id" type ="Integer"/>
         <element name="purpose" type ="String"/>
         <element name="accessibility" type ="String"/>
         <element name="exhibition_id" type ="Integer"/>
      </extension>
     <all>
        <element name="Archive.attended_by" minOccurs="1">
          <complexType>
            <choice minOccurs="1" maxOccurs="unbounded">
              <element ref="expert"/>
            </choice>
          </complexType>
        </element>
     </all>
    </complexContent>
  </complexType>
```

```
<!-- ************************************************************ -->
<!-- *    V I R T U A L   E X H I B I T I O N          * -->
<!-- ************************************************************ -->
<element name="virtualExhibition" type ="virtualExhibition"/>
<complexType name= "virtualExhibition">
  <complexContent>
    <extension base="exhibition">
        <element name="virt_id" type ="Integer"/>
        <element name="url" type ="URL"/>
        <element name="exhibition_id" type ="Integer"/>
    </extension>
  </complexContent>
</complexType>
<!-- /////////////// E N D   O F   E X H I B I T I O N   C L A S S  //////////// -->

<!-- ***** S T A R T   O F  I N S T I T U T I O N   C L A S S ***** ---> 
 <complexType name= "Institution">
     <element name="institution_id" type="Integer"/>
     <element name="name" type="String"/>
     <element name="institution_Type" type="String"/>
     <element name="country" type="country"/>
     <element name="description" type="String"/>
     <element name="url" type="String"/>
   <all>
     <element name="institution.located_at_host" maxOccurs="1">
        <complexType>
             <element ref="address"/>
        </complexType>
     </element>
     <element name="institution.belongsTo" minOccurs="1">
        <complexType>
             <element ref="person"/>
        </complexType>
     </element>
     <element name="institution.acquired_from" minOccurs="1">
        <complexType>
             <element ref="acquision"/>
        </complexType>
     </element>
     <element name="institution.owned_by" minOccurs="1">
        <complexType>
             <element ref="object"/>
        </complexType>
     </element>
     <element name="institution.availabled_in" minOccurs="1">
        <complexType>
             <element ref="objectinformation"/>
        </complexType>
     </element>
     <element name="institution.published_by" minOccurs="1">
        <complexType>
             <element ref="collection"/>
        </complexType>
     </element>
     <element name="institution.controlled_by" minOccurs="1">
        <complexType>
             <element ref="board"/>
        </complexType>
     </element>
```

```
        <element name="institution.supervised_by" minOccurs="1">
          <complexType>
               <element ref="visiting_group"/>
          </complexType>
        </element>
        <element name="institution.hosted_by" minOccurs="1">
          <complexType>
               <element ref="exhibition"/>
          </complexType>
        </element>
      </all>
    </complexType>

    <!-- ************************************************************ -->
    <!--* P U B L I C   I N S T I T U T I O N   C L A S S *-->
    <!-- ************************************************************  -->
    <element name="public" type ="public"/>
    <complexType name= "public">
      <complexContent>
        <extension base="institution">
        </extension>
      </complexContent>
    </complexType>

    <!-- ************************************************************* -->
    <!--* P R I V A T E   I N S T I T U T I O N  C L A S S*-->
    <!-- *************************************************************-->
    <element name="private" type ="private"/>
    <complexType name= "private">
      <complexContent>
        <extension base="institution">
        </extension>
      </complexContent>
    </complexType>

<!-- ****************************************************************** -->
<!-- ********** S T A R T   A D D R E S S   C L A S S    **** -->
<!-- ****************************************************************** -->

    <element name="address" type ="address"/>
    <complexType name= "address">
      <element name="aaddrss_id" type="Integer"/>
      <element name="addr1" type="String"/>
      <element name="addr2" type="String"/>
      <element name="town" type="String"/>
      <element name="county" type="String"/>
      <element name="country" type="String"/>
      <element name="fax" type="String"/>
      <element name="zipCode" type="String"/>
      <element name="institution_id" type="Integer"/>
      <element name="person_id" type="Integer"/>
    </complexType>
```

```
<!-- ************************************************************** -->
<!-- *** START  OF PERSON  CLASS *** -->
<!-- ************************************************************** -->

<element name= "person" type = "person"/>
<complexType name= "person">
     <element name="person_id" type="Integer"/>
     <element name="f_Name" type="String"/>
     <element name="l_Name" type="String"/>
     <element name="registered" type="String"/>
     <element name="birth_Date" type="Date"/>
     <element name="sex" type="Sex"/>
     <element name="nationality" type="String"/>
     <element name="job" type="String"/>
     <element name="phone" type="String"/>
     <element name="mobile" type="String"/>
     <element name="email" type="String"/>
     <element name="last_Login" type="Date"/>
     <element name="institution_id" type="Integer"/>
     <element name="broad_id" type="Integer"/>
</complexType>

<!-- ************************************************************** -->
<!-- *     COLLECTOR  CLASS          * -->
<!-- ************************************************************** -->
<element name="collector" type ="collector"/>
<complexType name= "collector">
  <complexContent>
     <extension base="person">
         <element name="collector_id" type="Integer"/>
         <element name="activity" type="String"/>
         <element name="description" type="String"/>
         <element name="person_id" type="Integer"/>
     </extension>
  </complexContent>
</complexType>

<!-- ********************************************************** -->
<!-- *         STAFF CLASS           * -->
<!-- ********************************************************** -->
<element name="staff" type ="staff"/>
<complexType name= "staff">
  <complexContent>
     <extension base="person">
         <element name="staff_id" type="Integer"/>
         <element name="enrollment_Date" type  ="Date"/>
         <element name="position" type="String"/>
         <element name="dept" type ="String"/>
         <element name="authorized" type="String"/>
         <element name="person_id" type="Integer"/>
     </extension>
  </complexContent>
</complexType>
```

```
<!-- ********************************************************** -->
<!-- *    C U R A T O R     C L A S S              * -->
<!-- ********************************************************** -->
<element name="curator" type ="curator"/>
<complexType name= "curator">
  <complexContent>
     <extension base="person">
         <element name="curator_id" type="Integer"/>
         <element name="enrollment_date" type  ="Date"/>
         <element name="activity" type="String"/>
         <element name="description" type="String"/>
         <element name="authorized" type="String"/>
         <element name="person_id" type="Integer"/>
     </extension>
  </complexContent>
</complexType>

<!-- ************************************************************ -->
<!-- *        V I S I T O R    C L A S S              * -->
<!-- ************************************************************ -->
<element name="visitor" type ="visitor"/>
<complexType name= "visitor">
  <complexContent>
     <extension base="Person">
        <all>
         <element name="visitor_id" type="Integer"/>
         <element name="grp_id" type="Integer"/>
         <element name="person_id" type="Integer"/>
        </all>
     </extension>
  </complexContent>
</complexType>

<!-- ************************************************************** -->
<!-- *        W E B  V I S I T O R    C L A S S         * -->
<!-- ************************************************************** -->
<element name="webVisitor_Virtual" type ="webVisitor_Virtual"/>
<complexType name= "WebVisitor_Virtual">
  <complexContent>
     <extension base="person">
        <all>
         <element name="virt_id" type ="Integer"/>
         <element name="web_id" type ="Integer"/>
         <element name="LastLogInDate" type ="Date"/>
        </all>
     </extension>
  </complexContent>
</complexType>
```

```
<!-- ************************************************************ -->
<!-- *       E X T E R N A L   C L A S S                        * -->
<!-- ************************************************************ -->
<element name="external" type ="external"/>
<complexType name= "external">
  <complexContent>
     <extension base="person">
        <all>
          <element name="ext_id" type ="Integer"/>
          <element name="activity" type ="String"/>
          <element name="description" type ="String"/>
          <element name="person_id" type ="Integer"/>
        </all>
     </extension>
  </complexContent>
</complexType>

<!-- ************************************************************ -->
<!-- *       E X P E R T     C L A S S                          * -->
<!-- ************************************************************ -->
<element name="Expert" type ="Expert"/>
<complexType name= "expert">
  <complexContent>
     <extension base="Person">
        <all>
          <element name="expert_id" type ="Integer"/>
          <element name="activity" type ="String"/>
          <element name="expertise" type ="String"/>
          <element name="description" type ="String"/>
          <element name="person_id" type ="Integer"/>
          <element name="arch_id" type ="Integer"/>
        </all>
     </extension>
  </complexContent>
</complexType>

<!-- ************************************************************ -->
<!-- ****        U S E R       C L A S S                     **** -->
<!-- ************************************************************ -->

<element name= "user" type = "user"/>
<complexType name= "user">
  <all>
     <element name="user_id" type = "Integer"/>
     <element name="usrName" type  = "String"/>
     <element name="usrPasswd" type = "String"/>
     <element name="registration_Date" type  = "country"/>
     <element name="expire_Date" type = "Integer"/>
     <element name="righits_Rights_id" type  = "Integer"/>
     <element name="user.identitfied_as" maxOccurs="1">
        <complexType>
            <element ref="person"/>
        </complexType>
     </element>
  </all>
</complexType>
```

```
<!-- ******************************************************** -->
<!-- *      RIGHTS     CLASS          * -->
<!-- ******************************************************** -->

<element name= "rights" type ="rights"/>
<complexType name= "rights">
  <all>
    <element name="rights_id" type ="Integer"/>
    <element name="reading" type  ="String"/>
    <element name="writing" type ="String"/>
    <element name="description" type="String"/>
    <element name="documentation" type ="String"/>
    <element name="publishing" type  ="String"/>
    <element name="copying" type  ="String"/>
    <element name="registration" type ="String"/>
    <element name="authorization" type ="String"/>
    <element name="rights.autorized_for" maxOccurs="1">
      <complexType>
          <element ref="User"/>
      </complexType>
    </element></all>
</complexType>

<!-- *************************************************************************** -->
<!-- **START  OF VISITING  GROUP  CLASS *** -->
<!-- *************************************************************************** -->

<element name= "visitingGroup" type = "visitingGroup"/>
<complexType name= "visitingGroup">
  <all>
    <element name="grp_id" type ="Integer"/>
    <element name="visiting_Date" type  ="Date"/>
    <element name="purpose" type ="String"/>
    <element name="num_visitors" type ="Integer"/>
    <element name="origin" type ="String"/>
    <element name="description" type ="String"/>
    <element name="dsp_Dsp_id" type ="Integer"/>
    <element name="institution_id" type ="Integer"/>
    <element name="visitingGroup.isOneOf" minOccurs="0">
      <complexType>
        <choice minOccurs="0" maxOccurs="unbounded">
          <element ref="Visitor"/>
        </choice>
      </complexType>
    </element>
    <element name="visitingGroup.visited_by" minOccurs="0">
      <complexType>
        <choice minOccurs="0" maxOccurs="unbounded">
          <element ref="publicDisplay"/>
        </choice>
      </complexType>
    </element>
  </all>
</complexType>
```

```
<!-- ****************************************************************** -->
<!--S T A R T   O F   e n u m e r a t i o n   C L A S S E S -->
<!-- **************************************************************** -->
  <simpleType name="Kind">
   <restriction base="String">
      <enumeration value="Tourism"/>
      <enumeration value="Museum"/>
      <enumeration value="Univeristy"/>
      <enumeration value="OTHER"/>
   </restriction>
  </simpleType>

  <!-- ****************************** -->
  <!-- enumeration for P_Status -->
  <!-- ****************************** -->
 <simpleType name="P_Status">
  <restriction base="String">
     <enumeration value="printed"/>
     <enumeration value="personal"/>
     <enumeration value="WWW"/>
     <enumeration value="Unpublished"/>
  </restriction>
 </simpleType>

  <!-- ******************************************* -->
  <!-- enumeration for Acquisition Method -->
  <!-- ****************************************** -->
 <simpleType name="Acquisiton">
  <restriction base="String">
     <enumeration value="field"/>
     <enumeration value="donated"/>
     <enumeration value="exchange"/>
     <enumeration value="visiting"/>
     <enumeration value="purchase"/>
     <enumeration value="unknown"/>
     <enumeration value="other"/>
  </restriction>
 </simpleType>

 <!-- ************************************ -->
 <!--    enumeration for Board       -->
 <!-- ************************************ -->
 <simpleType name="Board">
  <restriction base="String">
     <enumeration value="ownership"/>
     <enumeration value="executive"/>
     <enumeration value="employment"/>
     <enumeration value="contracting"/>
     <enumeration value="associated"/>
     <enumeration value="partnership"/>
     <enumeration value="advisory"/>
     <enumeration value="monitoring"/>
     <enumeration value="other"/>
  </restriction>
 </simpleType>
```

```
<!-- *********************************** -->
<!--    enumeration for country       -->
<!-- *********************************** -->
<simpleType name="country">
  <restriction base="String">
     <enumeration value="UK"/>
     <enumeration value="USA"/>
     <enumeration value="Bulgium"/>
     <enumeration value="Germany"/>
     <enumeration value="Bulgaria"/>
     <enumeration value="Italy"/>
     <enumeration value="Portugal"/>
     <enumeration value="Norway"/>
     <enumeration value="Netherlands"/>
     <enumeration value="Romania"/>
     <enumeration value="Ireland"/>
     <enumeration value="Spain"/>
     <enumeration value="Sweden"/>
     <enumeration value="Swiss"/>
     <enumeration value="Hungary"/>
     <enumeration value="Other"/>
  </restriction>
</simpleType>

<!-- *********************************** -->
<!--  enumeration for J o b            -->
<!-- *********************************** -->
<simpleType name="Job">
  <restriction base="String">
     <enumeration value="Teacher"/>
     <enumeration value="Student"/>
     <enumeration value="Engineer"/>
     <enumeration value="Researcher"/>
     <enumeration value="Administrative"/>
     <enumeration value="Executive"/>
     <enumeration value="Techician"/>
     <enumeration value="Expert"/>
     <enumeration value="OTHER"/>
  </restriction>
</simpleType>

 <!-- ************************************* -->
<!-- enumeration for Location           -->
<!-- ************************************* -->
<simpleType name="Normal_location">
  <restriction base="String">
     <enumeration value="excavated"/>
     <enumeration value="found"/>
     <enumeration value="uncovered"/>
     <enumeration value="trapped"/>
     <enumeration value="other"/>
  </restriction>
</simpleType>
```

```
        <!-- *************************************** -->
        <!-- enumeration for Reference type   -->
        <!-- **************************************  -->
        <simpleType name="Reference_location">
          <restriction base="String">
              <enumeration value="article"/>
              <enumeration value="book"/>
              <enumeration value="collection"/>
              <enumeration value="document"/>
              <enumeration value="note"/>
              <enumeration value="proceedings"/>
              <enumeration value="report"/>
              <enumeration value="online"/>
              <enumeration value='communication'/>
              <enumeration value="other"/>
          </restriction>
        </simpleType>

       <!-- **************************  -->
       <!-- enumeration for S E X  -->
       <!-- **************************  -->
        <simpleType name="Sex">
          <restriction base="String">
              <enumeration value="M"/>
              <enumeration value="F"/>
          </restriction>
        </simpleType>

        <!-- *********************************     -->
        <!-- enumeration for Rights            -->
        <!-- *********************************     -->
        <simpleType name="Rights">
          <restriction base="String">
              <enumeration value="NO"/>
              <enumeration value="YES"/>
              <enumeration value="N/A"/>
              <enumeration value="OTHER"/>
          </restriction>
        </simpleType>
      <!--E N D  O F   e n u m e r a t i o n   C L A S S E S-->
</XMLSchema>

 <!--E N D  O F  XMLSchema  -->
```

APPENDIX B

JAVA CLASSES (JAVA SOURCE CODE)

Implementation of online and offline system

1) Java (Classes) For Generating of architecture components:

The following Java classes used for generating different components as shown below.

Theses classes receives an XMLSchema and generating components as:

i) SQL Schema

ii) XSL stylesheet

iii) XQuery interpreter

```
/*****************************************************************************************
 * i) GenSQLSchema.java                                                                  *
 * This program for generating SQL DDL from XMLSchema by using XSLT                      *
 * the XSLT is designed for working with any XMLSchema that means has                    *
 * extensions                                                                            *
 * *.xml, *.xsd and *.xsl                                                                *
*****************************************************************************************/
import org.w3c.dom.*;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.DOMException;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import javax.xml.transform.*;
import javax.xml.transform.stream.StreamSource;
import javax.xml.transform.stream.StreamResult;
import java.io.*;

public class GenSQLSchema
{
  static Document document;
  public static void main (String argv [])
  {
    if (argv.length != 2) {
      System.err.println ("Usage: java GenSQLSchema stylesheet xmlSchema");
      System.exit (1);
    }
    DocumentBuilderFactory factory =
      DocumentBuilderFactory.newInstance();
    try {
      File stylesheet = new File(argv[0]);
      File datafile   = new File(argv[1]);
      DocumentBuilder builder = factory.newDocumentBuilder();
      document = builder.parse(datafile);
      TransformerFactory tFactory = TransformerFactory.newInstance();
      StreamSource stylesource = new StreamSource(stylesheet);
      Transformer transformer = tFactory.newTransformer(stylesource);
      DOMSource source = new DOMSource(document);
      StreamResult result = new StreamResult(System.out);
      transformer.transform(source, result);
    } catch (TransformerConfigurationException tce) {
```

```
    /*-----------------------------------------*
     * Errors generated by the parser *
     *-----------------------------------------*/
    System.out.println ("\n** Transformer Factory error");
    System.out.println("   " + tce.getMessage() );
     /* ------------------------------------------- *
      * Use the contained exception, if any  *
      * ------------------------------------------- */
    Throwable x = tce;
    if (tce.getException() != null)
       x = tce.getException();
    x.printStackTrace();
  } catch (TransformerException te) {
    // Error generated by the parser
    System.out.println ("\n** Transformation error");
    System.out.println("   " + te.getMessage() );
    Throwable x = te;
    if (te.getException() != null)
       x = te.getException();
    x.printStackTrace();
   } catch (SAXException sxe) {
    Exception  x = sxe;
    if (sxe.getException() != null)
       x = sxe.getException();
    x.printStackTrace();
  } catch (ParserConfigurationException pce) {
     pce.printStackTrace();
  } catch (IOException ioe) {
    // I/O error
    ioe.printStackTrace();
  }
}
```

```
/*****************************************************************************************
 *   ii) GenerateXSL.java                                                                 *
 * This program for generating XSL stylesheet of XMLSchema through   XSLT                 *
 * The generated XSL stylesheet can be then used to integrate components                  *
 *****************************************************************************************/
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.*;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.w3c.dom.DOMException;
import javax.xml.transform.*;
import java.io.*;
public class GenerateXSL
{
  static Document document;
  public static void main (String argv [])
  {
    if (argv.length != 2) {
      System.err.println ("Usage: java Stylizer stylesheet xmlSchema");
      System.exit (1);
    }
    try {
      File stylesheet = new File(argv[0]);
      File datafile   = new File(argv[1]);
      DocumentBuilderFactory factory =
                                    DocumentBuilderFactory.newInstance();
      DocumentBuilder builder = factory.newDocumentBuilder();
      document = builder.parse(datafile);
      TransformerFactory tFactory =
        TransformerFactory.newInstance();
      StreamSource stylesource = new StreamSource(stylesheet);
      Transformer transformer = tFactory.newTransformer(stylesource);
      DOMSource source = new DOMSource(document);
      StreamResult result = new StreamResult(System.out);
      transformer.transform(source, result);
      walk ( );
    } catch (TransformerConfigurationException tce) {
      System.out.println ("\n** Transformer Factory error");
      System.out.println("   " + tce.getMessage() );
      Throwable x = tce;
      if (tce.getException() != null)
        x = tce.getException();
      x.printStackTrace();
    } catch (TransformerException te) {
      System.out.println ("\n** Transformation error");
      System.out.println("   " + te.getMessage() );
      Throwable x = te;
    if (te.getException() != null)
        x = te.getException();
      x.printStackTrace();
    }
   catch (SAXException sxe) {
      Exception  x = sxe;
      if (sxe.getException() != null)
        x = sxe.getException();
      x.printStackTrace();
    } catch (ParserConfigurationException pce) {
```

```
      pce.printStackTrace();
    } catch (IOException ioe) {
      ioe.printStackTrace();
    }
  }
  private void walk(Node node)
  {
   int type = node.getNodeType();
   switch(type)
   {
     }//end of document
     case Node.ELEMENT_NODE:
     {
        System.out.print('<' + node.getNodeName() );
        NamedNodeMap nnm = node.getAttributes();
        if(nnm != null )
        {
           int len = nnm.getLength() ;
           Attr attr;
           for ( int i = 0; i < len; i++ )
           {
              attr = (Attr)nnm.item(i);
              System.out.print(' '
                  + attr.getNodeName()
                  + "=\""
                  + attr.getNodeValue()
                  + "" );
           }
        }
        System.out.print('>');
        break;
     }//end of element
     case Node.ENTITY_REFERENCE_NODE:
     {
       System.out.print('&' + node.getNodeName() + ';' );
       break;
     }//end of entity
     case Node.TEXT_NODE:
     {
        System.out.print(node.getNodeValue());
        break;
     }
     case Node.PROCESSING_INSTRUCTION_NODE:
     {
        System.out.print("<?"
           + node.getNodeName() ) ;
        String data = node.getNodeValue();
        if ( data != null && data.length() > 0 ) {
           System.out.print(' ');
           System.out.print(data);
        }
        System.out.println("?>");
        break;
      }
   }//end of switch

   for(Node child = node.getFirstChild(); child != null; child =
         child.getNextSibling())
    {
      walk(child);
```

```
      }
      //without this the ending tags will miss
      if ( type == Node.ELEMENT_NODE )
      {
        System.out.print("</" + node.getNodeName() + ">");
      }

    }//end of walk
} // end class

/***************************************************************************
 * iii) GenerateXSLXQuerySQL.java                                          *
 *                                                                         *
 * This program for generating XSL Stylesheet of XMLSchema                 *
 *  for transforming any XQuery to SQL query                               *
 ***************************************************************************/
import org.w3c.dom.*;
import javax.xml.parsers.*;
import org.xml.sax.*;
import javax.xml.transform.*;
import java.io.*;
public class GenerateXSLXQuerySQL
{
  public static void main (String argv [])
  {
    if (argv.length != 2) {
       System.err.println ("Usage: java GenSQLSchema stylesheet xmlSchema");
       System.exit (1);
    }
    DocumentBuilderFactory factory =
       DocumentBuilderFactory.newInstance();
     try {
       File stylesheet = new File(argv[0]);
       File datafile   = new File(argv[1]);
       DocumentBuilder builder = factory.newDocumentBuilder();
       document = builder.parse(datafile);
       TransformerFactory tFactory =  TransformerFactory.newInstance();
       StreamSource stylesource   =   new StreamSource(stylesheet);
       Transformer transformer = tFactory.newTransformer(stylesource);
       DOMSource source = new DOMSource(document);
       System.out.println ("\n");
       StreamResult result = new StreamResult(System.out);
       transformer.transform(source, result);
    } catch (TransformerConfigurationException tce) {
      System.out.println ("\n** Transformer Factory error");
      System.out.println("   " + tce.getMessage() );
      Throwable x = tce;
      if (tce.getException() != null)
         x = tce.getException();
      x.printStackTrace();
    } catch (TransformerException te) {
      System.out.println ("\n** Transformation error");
      System.out.println("   " + te.getMessage() );
      Throwable x = te;
      if (te.getException() != null)
         x = te.getException();
      x.printStackTrace();
     } catch (SAXException sxe) {
      Exception  x = sxe;
```

```
      if (sxe.getException() != null)
         x = sxe.getException();
      x.printStackTrace();
   } catch (ParserConfigurationException pce) {
       pce.printStackTrace();
   } catch (IOException ioe) {
      ioe.printStackTrace();  }
//walk the DOM tree
private void walk(Node node)
{
   int type = node.getNodeType();
   switch(type)
   {
      case Node.ELEMENT_NODE:
      {
         System.out.print('<' + node.getNodeName() );
         NamedNodeMap nnm = node.getAttributes();
         if(nnm != null )
         {
            int len = nnm.getLength() ;
            Attr attr;
            for ( int i = 0; i < len; i++ )
            {
               attr = (Attr)nnm.item(i);
               System.out.print(' '
                   + attr.getNodeName() + "=\""
                   + attr.getNodeValue()  +  "" ); } }
         System.out.print('>');
         break;
       }//end of element
      case Node.ENTITY_REFERENCE_NODE:
      {
        System.out.print('&' + node.getNodeName() + ';' );
        break;
      }//end of entity
      case Node.TEXT_NODE:
      {
         System.out.print(node.getNodeValue());
         break;
      }
      case Node.PROCESSING_INSTRUCTION_NODE:
      {
         System.out.print("<?"
            + node.getNodeName() ) ;
         String data = node.getNodeValue();
         if ( data != null && data.length() > 0 ) {
            System.out.print(' ');
            System.out.print(data);
         }
         System.out.println("?>");
         break;}
   }//end of switch
   for(Node child = node.getFirstChild(); child != null; child = child.getNextSibling())
   {
      walk(child); }
   if ( type == Node.ELEMENT_NODE )
   {
      System.out.print("</" + node.getNodeName() + ">");
   }
}//end of walk
```

2) Java classes for integrated architecture of distributed information

The following Java classes are used for integrating offline components with the online components. In other words, for testing the generated components (offline) with (online). Theses classes used for transforming XML documents to different formats and storing in database, presentation, querying and XQuery interpreters for retrieval.

i) **XMLSQLServletDB** (transform XML to SQL browse on Tomcat server and store the result in DB)

ii) **BDXMLHTMLServlet** (presentation in HTML format on Tomcat server)

iii) **DbaseSQLXMLServlet** (transform SQL to XML and browse on Server)

iv) **XQuerySQLServletDB** (transform XQuery to SQL and retrieve data of DB and display the result on servlet in XML format)

```
/**********************************************************
 * XMLSQLServletDB.java                                  *
 * This program for transforming XML document  *
 * to SQL by using XSLT Browse on the servlet,  *
 * then stored the data into database            *
 *********************************************************/
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import java.sql.*;
import oracle.jdbc.*;
import oracle.jdbc.driver.*;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import org.xml.sax.*;
import org.w3c.dom.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.*;
public class XMLSQLServletDB extends HttpServlet
{
   public void init(ServletConfig config) throws ServletException
    {
      super.init(config);      }
      static Document document;
      Connection conn=null;
     public void doGet(HttpServletRequest request,
                   HttpServletResponse response)
        throws ServletException, IOException
     {
       String stylesheet = request.getParameter("stylesheet");
       String datafile   = request.getParameter("datafile");
```

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
 try {
        Class.forName("oracle.jdbc.OracleDriver");
     }
catch (ClassNotFoundException e)
   {
    System.err.println("Could not load the JDBC");
    return;
   }
try
{
  DocumentBuilderFactory factory =
   DocumentBuilderFactory.newInstance();
  DocumentBuilder builder = factory.newDocumentBuilder();
  document = builder.parse(datafile);
}catch (Exception e) {
  e.printStackTrace();
 }
try {
      DriverManager.registerDriver
     (new oracle.jdbc.driver.OracleDriver());
     conn = DriverManager.getConnection
     ("jdbc:oracle:thin:@localhost:1521:orcl","userId","password");
     TransformerFactory tFactory = TransformerFactory.newInstance();
     StreamSource stylesource = new StreamSource(stylesheet);
     Transformer transformer = tFactory.newTransformer(stylesource);
     DOMSource source = new DOMSource(document);
     CharArrayWriter out = new CharArrayWriter();
     StreamResult result = new StreamResult(out);
     transformer.transform(source, result);
     out.write(b);
     String s= new String(b);
     Statement insertStatement = conn.createStatement();
     insertStatement.execute(s);
      insertStatement.execute("COMMIT");
     insertStatement.close();
     conn.close(
    } catch (Exception e)
     {
      out.write(e.getMessage());
      e.printStackTrace(out);
     }
  }
}
```

```
/*****************************************************************************
 * ii) HTMLHTMLServlet.java                                                  *
 * This program for transforming XML document to HTML document              *
 *   by using XSLT, browser the output on servlet                            *
 *****************************************************************************/
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.w3c.dom.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.*;
public class XMLHTMLServlet extends HttpServlet
{
  static Document document;
  public void init(ServletConfig config) throws ServletException
   {
     super.init(config);
   }


  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
        throws IOException, ServletException
    {
          String stylesheet = request.getParameter("stylesheet");
          response.setContentType("text/html");
          PrintWriter out = response.getWriter();
         try
         {
           DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
           String datafile = request.getParameter("datafile");
           DocumentBuilder builder = factory.newDocumentBuilder();
           document = builder.parse(datafile);
         }catch (Exception e) {
           e.printStackTrace();
          }
         try {
            TransformerFactory tFactory = TransformerFactory.newInstance();
            StreamSource stylesource = new StreamSource(stylesheet);
            Transformer transformer = tFactory.newTransformer(stylesource);
            DOMSource src = new DOMSource(document);
            StreamResult result = new StreamResult(out);
            transformer.transform(src, result);
         } catch (Exception e)
           {
            out.write(e.getMessage()); e.printStackTrace(out);
           }
  }
}
```

```
/*****************************************************************
 *                                                               *
 * iii) DbaseSQLXMLServlet.java                                  *
 * This program for generation  XML of Database by using         *
 * XSL stylesheet transformation                                 *
 *  then the result browsed on the servlet in XML format         *
 *                                                               *
 *****************************************************************/
 import java.io.*;
 import javax.servlet.*;
 import javax.servlet.http.*;
 import java.util.*;
 import org.w3c.dom.*;
 import java.sql.*;
 import java.sql.PreparedStatement;
 import java.sql.DriverManager;
 import java.sql.ResultSetMetaData;
 import oracle.jdbc.*;
 import javax.xml.parsers.*;
 import org.xml.sax.*;
 import javax.xml.transform.*;
public class DbaseSQLXMLServlet extends HttpServlet
{

    public void init(ServletConfig config) throws ServletException
     {
       super.init(config);
     }
       static Document document;
       Connection conn=null;
       Statement stmnt = null;
      public void doGet(HttpServletRequest request,
                       HttpServletResponse response)
           throws ServletException, IOException
     {
        String stylesheet  = request.getParameter("stylesheet");
        String XMLdocument = request.getParameter("XMLdocument");
            response.setContentType("text/xml");
            PrintWriter out = response.getWriter();
            try {
                    Class.forName("oracle.jdbc.OracleDriver");
               }
            catch (ClassNotFoundException e)
               {
                 System.err.println("Could not load the JDBC");
                 return;
               }
          try
          {
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();
            document = builder.parse(XMLdocument);
          }catch (Exception e) {
            e.printStackTrace();
           }
          try {
                 DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver());
                   conn = DriverManager.getConnection
                  ("jdbc:oracle:thin:@localhost:1521:database","userId","password");
```

```
                TransformerFactory tFactory = TransformerFactory.newInstance();
                StreamSource stylesource = new StreamSource(stylesheet);
                Transformer transformer = tFactory.newTransformer(stylesource);
                DOMSource src = new DOMSource(document);
                StreamResult result = new StreamResult(out);
                transformer.transform(src, result);
                stmnt = conn.createStatement();
                String select = "";
                  ResultSet tion = stmnt.executeQuery(select);
                out.write("<?xml version=\"1.0\"?>\r\n");
                out.println("<DomRoot>");
                while( tion.next() )
                {
                  String id=tion.getString(); getNodeName
                  String name = tion. getNodeValue();
                  name = escapeText(name);
                  String type=tion.getString("type");
                  type = escapeText(type);
                  out.write(" <Element>"+tion.getString("id") +"</Element>\n");
                  out.write(" <Element>"+tion.getString("name") +"</Element>\n");
                  out.write(" <Element>"+tion.getString("type")
                  +"</Element>\n");
                }
                 out.println("</DomRoot>"); stmnt.close();  conn.close();
              } catch (Exception e) {
              out.write(e.getMessage());
              e.printStackTrace(out);
             }
    }
    public static String escapeText(String s) {
    if (s.indexOf('&') != -1 || s.indexOf('<') != -1) {
      StringBuffer result = new StringBuffer(s.length() + 5);
     for (int i = 0; i < s.length(); i++) {
       char c = s.charAt(i);
       if (c == '&') result.append("&amp;");
       else if (c == '<') result.append("&lt;");
       else if (c == '"') result.append("&quot;");
       else if (c == '\'') result.append("&apos;");
       else result.append(c);
     }
     return result.toString();
    }
    else {  return s; }
  }
}
```

```
/*****************************************************************************
 * v) XQuerySQLServletDB.java                                               *
 * This program for transforming  XQuery to SQL by using                    *
 * XSL stylesheet transformation and display the result on servlet          *
 *****************************************************************************/
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import java.sql.*;
import javax.xml.transform.*;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.dom.*;
import java.sql.DriverManager;
 public class XQuerySQLServletDB extends HttpServlet {
   static Document document;
   Connection conn=null;
   Statement stmnt = null;
   public void init(ServletConfig config) throws ServletException {
      super.init(config);   }
     public void doGet(HttpServletRequest request,
                   HttpServletResponse response)
          throws IOException, ServletException
    {
          String stylesheet = request.getParameter("stylesheet");
          response.setContentType("text/html");
          PrintWriter out = response.getWriter();
         try {
              DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver());
              conn = DriverManager.getConnection
              ("jdbc:oracle:thin:@localhost:1521:database","userId","password");
              DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
              String datafile = request.getParameter("datafile");
              DocumentBuilder builder = factory.newDocumentBuilder();
              document = builder.parse(datafile);
            } catch (Exception e)   {
              e.printStackTrace(); }
         try {
              TransformerFactory tFactory = TransformerFactory.newInstance();
              StreamSource stylesource = new StreamSource(stylesheet);
              Transformer transformer = tFactory.newTransformer(stylesource);
              DOMSource src = new DOMSource(document);
              StreamResult result = new StreamResult(out);
              transformer.transform(src, result);
              stmnt = conn.createStatement();
              String select = "";
              ResultSet tion = stmnt.executeQuery(select);
            } catch (Exception e) { out.write(e.getMessage());
             e.printStackTrace(out); }
    }
}
```

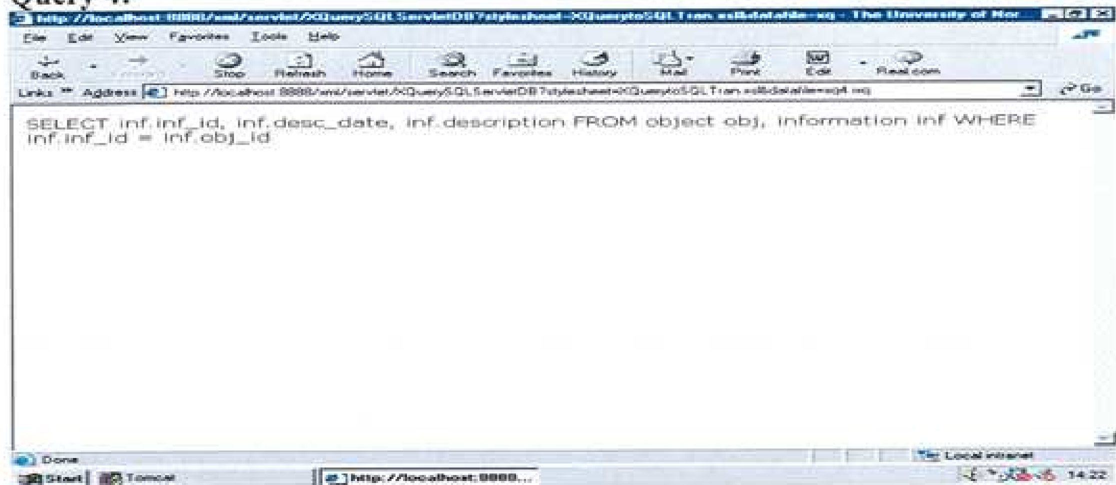
APPENDIX C:

IMPLEMENTATION OF TRANSFORMING XQUIERS TO SQL QUERIES

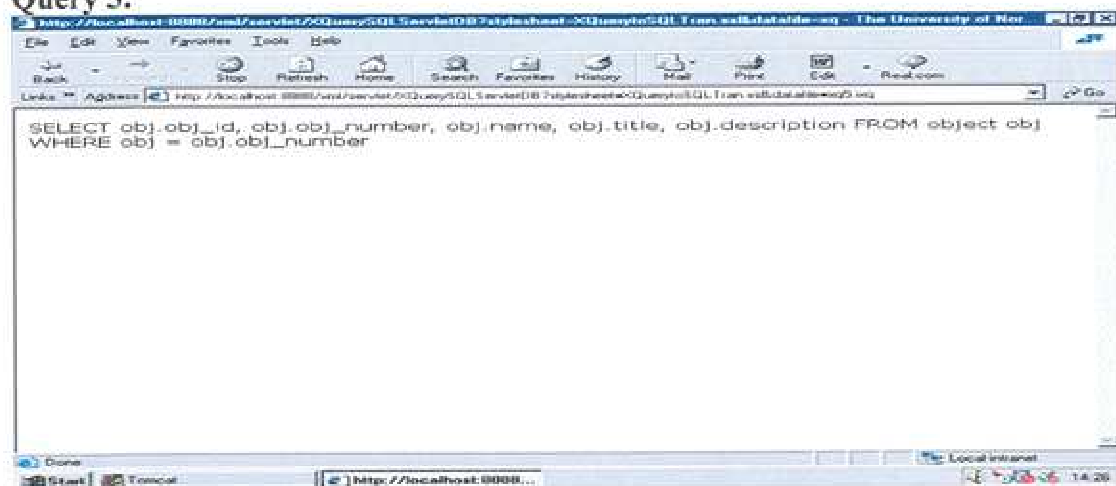
Implementation of transforming XQuiers to SQL queries

The following screens showed the implementation of the transforming (Queries 4, 5, 7, 8, 9,10,11 and 13) XQueries to SQL queries as in Table 10.

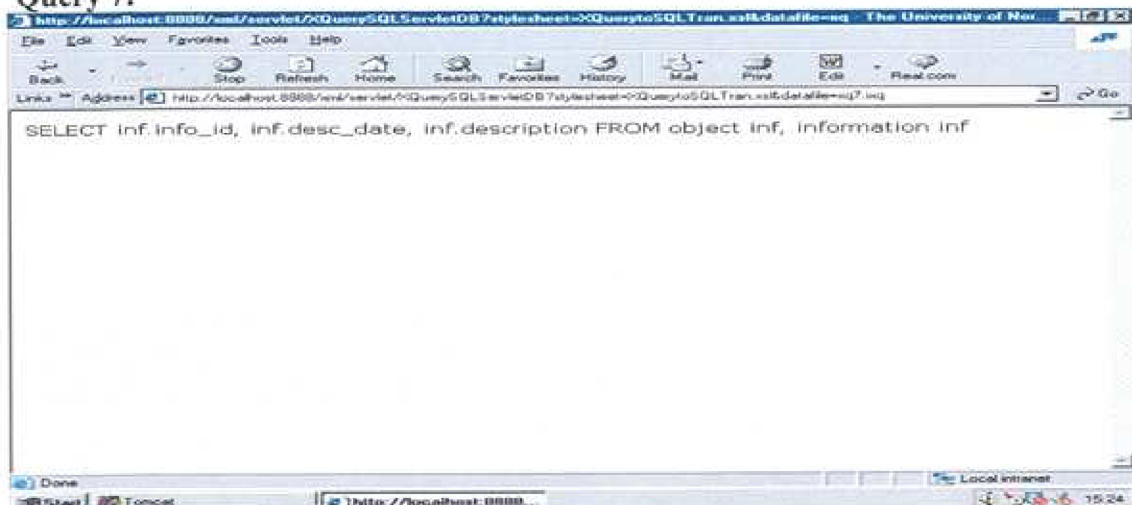
Query 4:



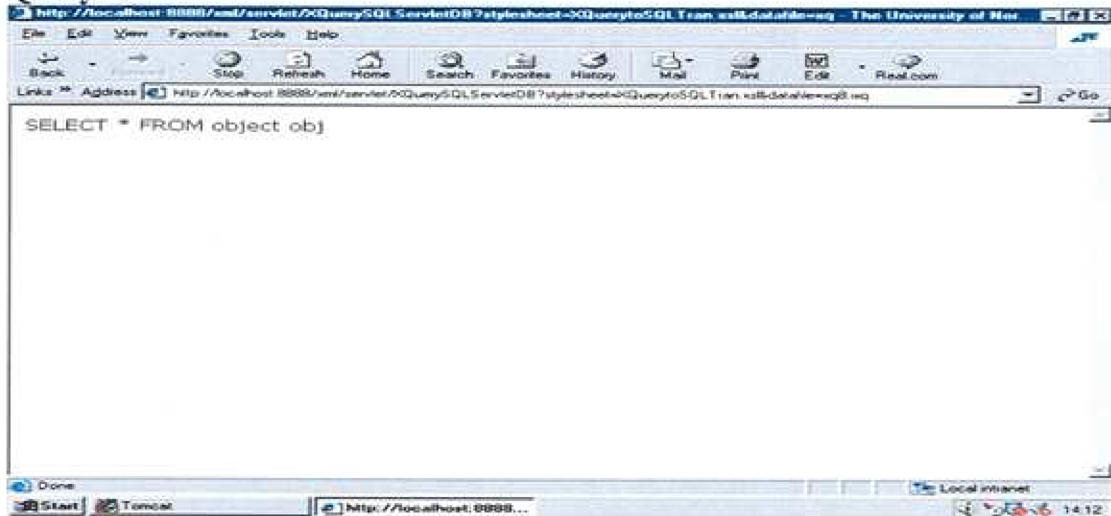
Query 5:



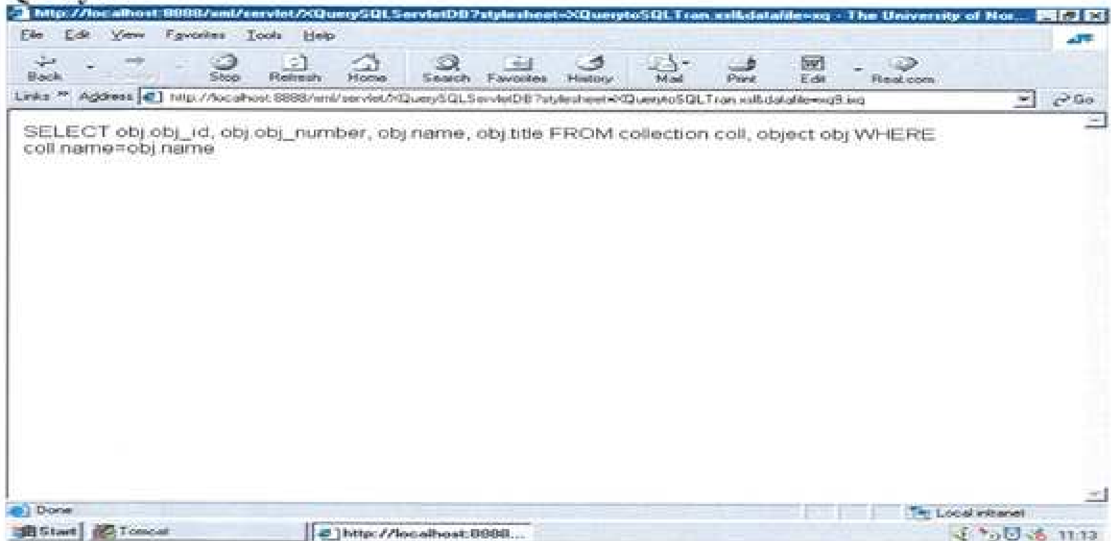
Query 7:



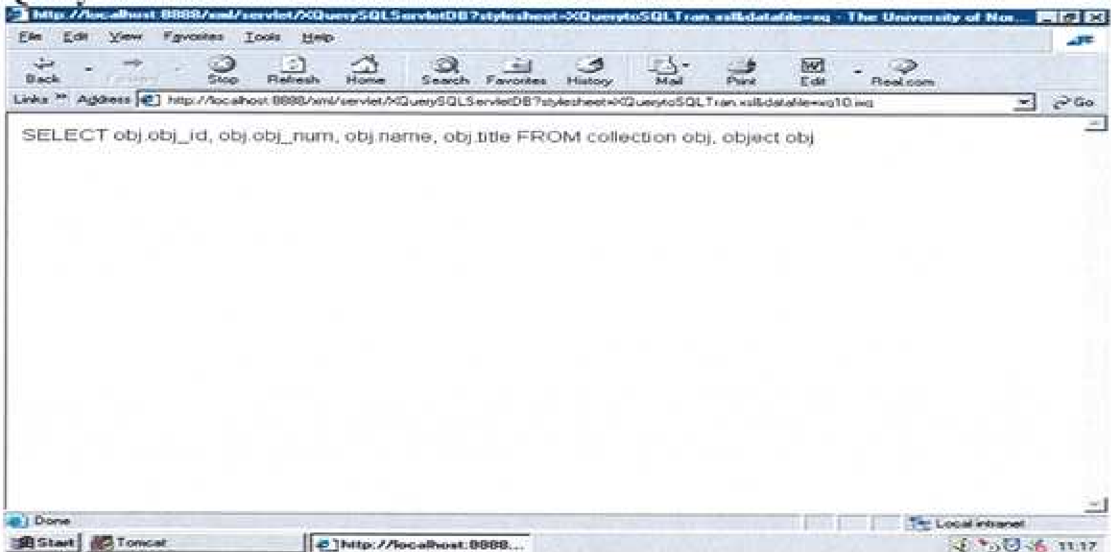
Query 8:



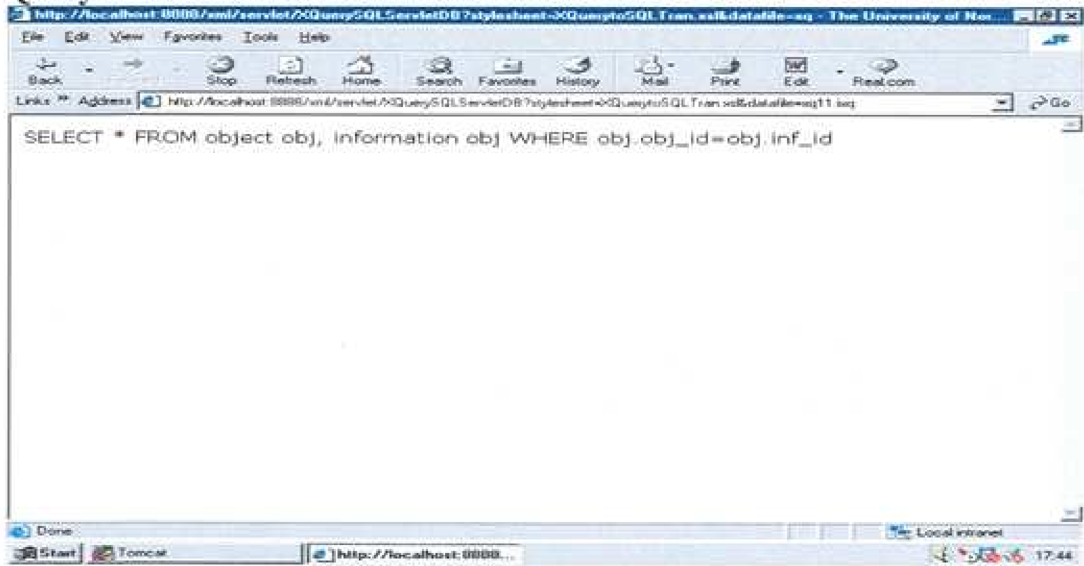
Query 9:



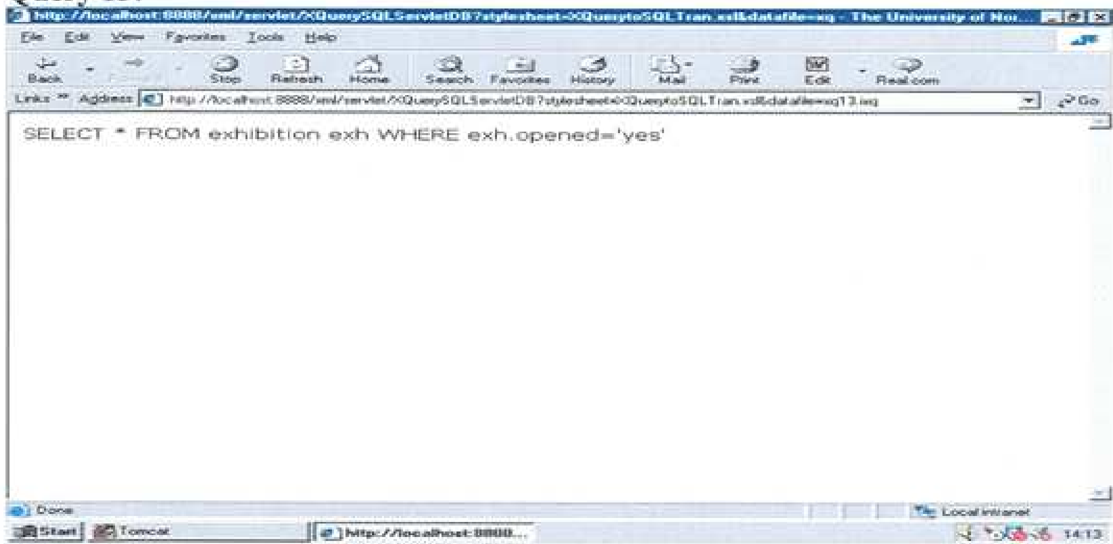
Query 10:



Query 11:



Query 13:



APPENDIX D:

ANALYSIS OF USE CASE, ACTIVITY DIAGRAM AND SEQUENCE DISGRAMS

- **Diagrams for analysis of use case, activity diagrams and sequence diagrams for the proposed system**

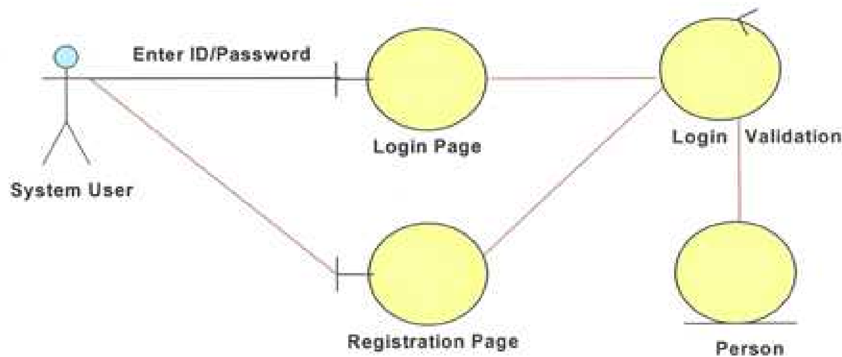
In the following more information on important diagram such as analysis of use case diagram, activity diagram and sequences diagram that shows how the system processes this activity.

1. Analysis of use cases diagram

The following diagrams explain the analysis of each use case for the museum system

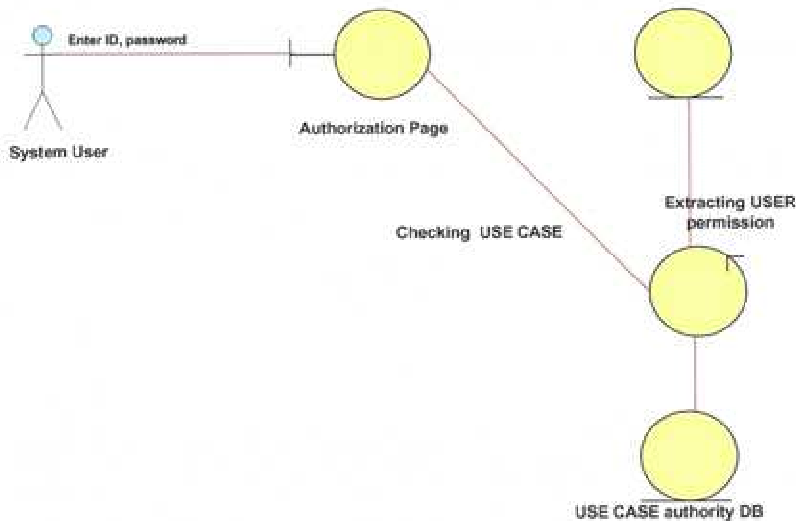
1.1 Login diagram

Handling login activities such as Registration procedure for new system users (Curator and staff) and login procedure



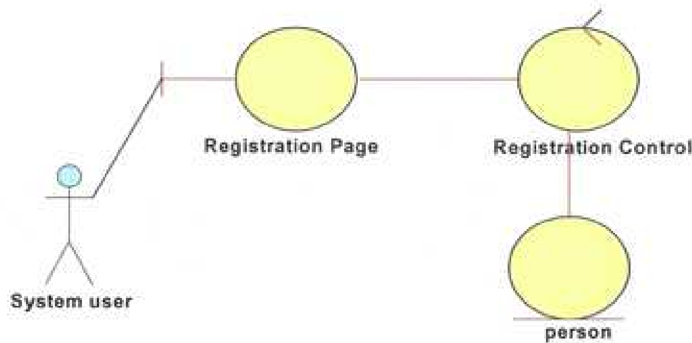
1.2 Authorization diagram

Authorization activity is responsible for (who can do what) on the system



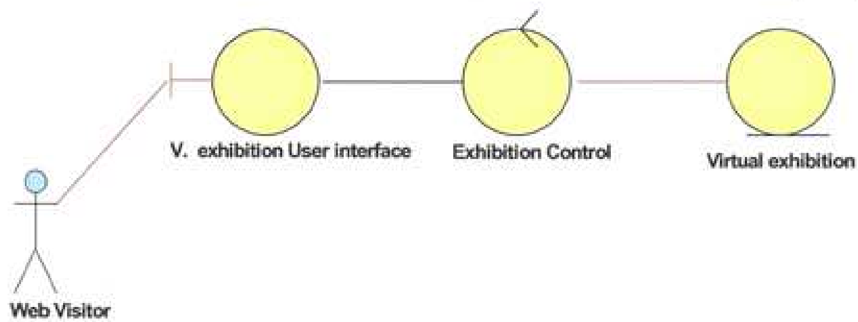
1.3 Registration diagram

Such extended activity is required when Web visitors want to *Sign in* for the first time.



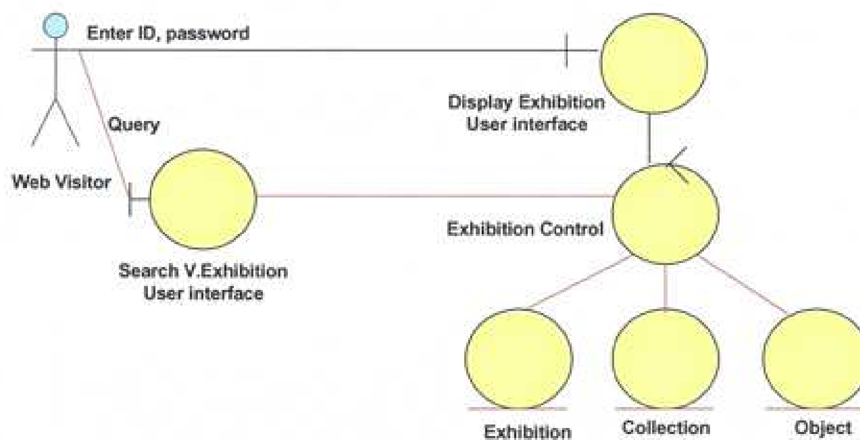
1.4 Browsing virtual exhibition

Allow Web visitors to browse the Virtual Exhibition over the web (freely)



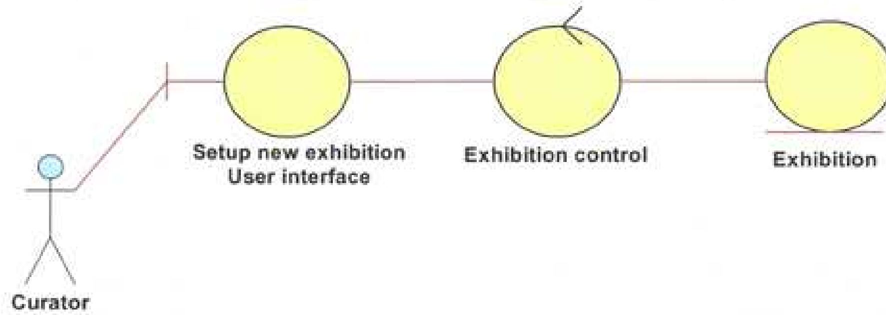
1.5 Search and display virtual exhibition

This activity for Search and Browse Virtual Exhibition



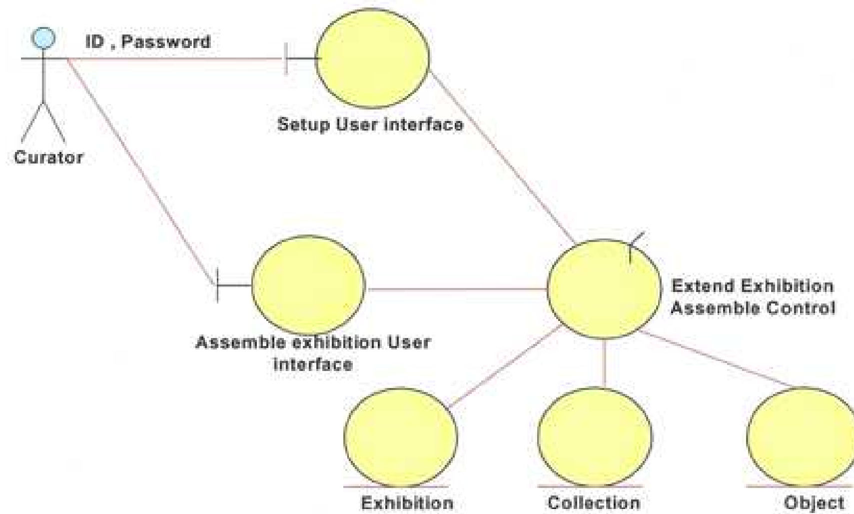
1.6 Setup new exhibition

This activity includes setting up new exhibition, providing the system



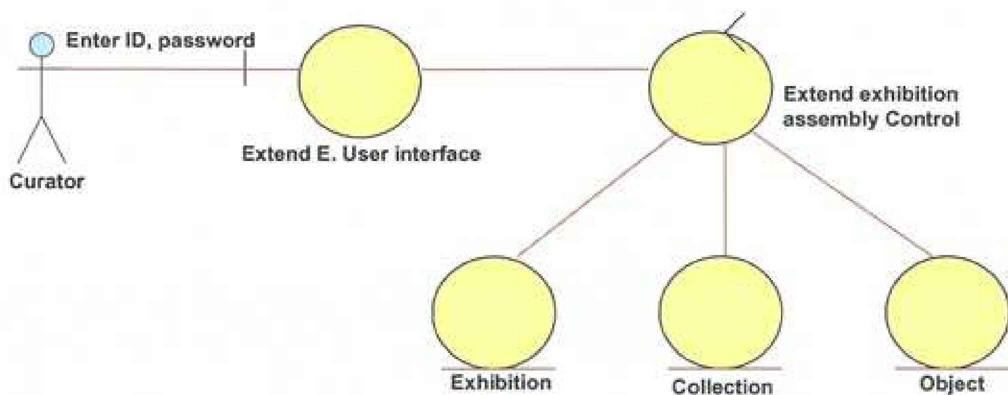
1.7 Assemble exhibition

This activity includes assembling published objects and collections for the intended exhibition



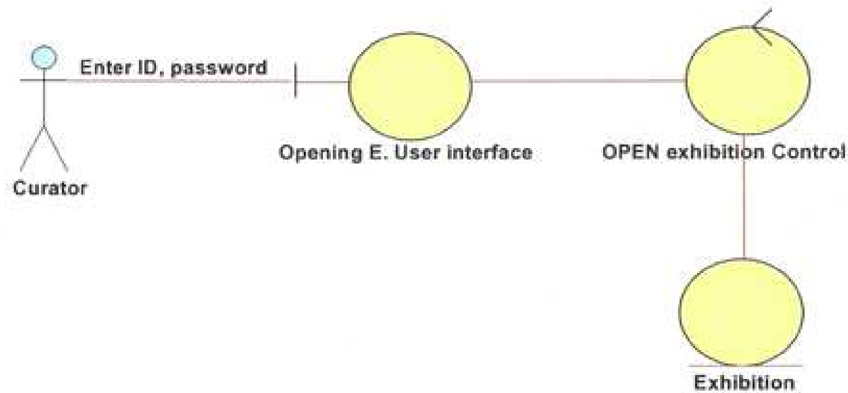
1.8 Extending an exhibition

This activity includes updating Exhibition activities



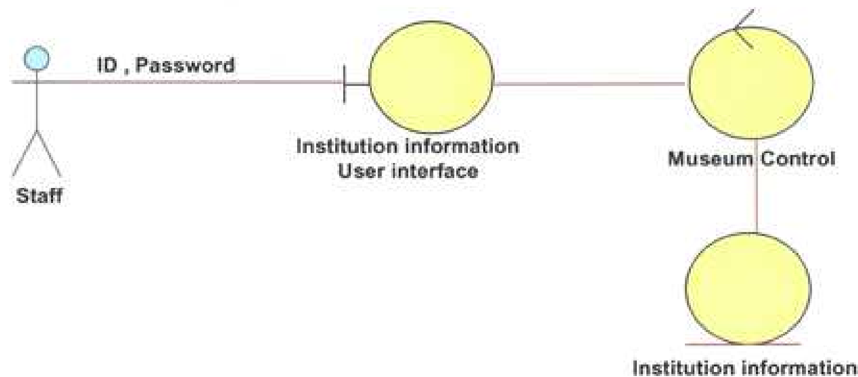
1.9 Open an exhibition

This activity is responsible for opening the exhibition. This includes the kind of exhibition (any information about virtual exhibition or public display)



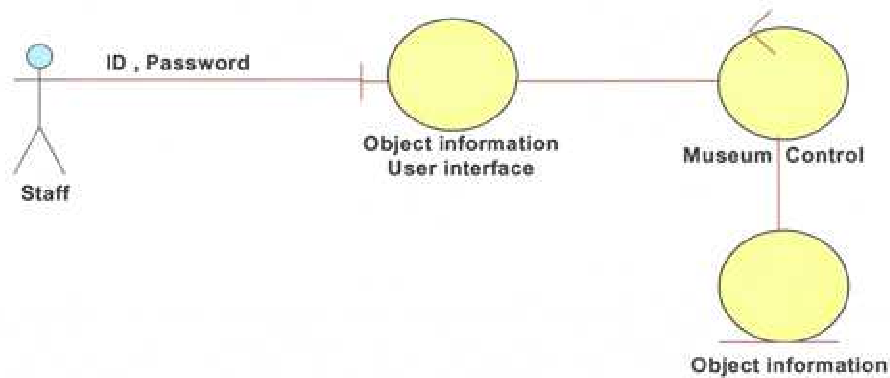
1.10 Institutions information

This activity includes updating Institutions information (any information related to institutions)



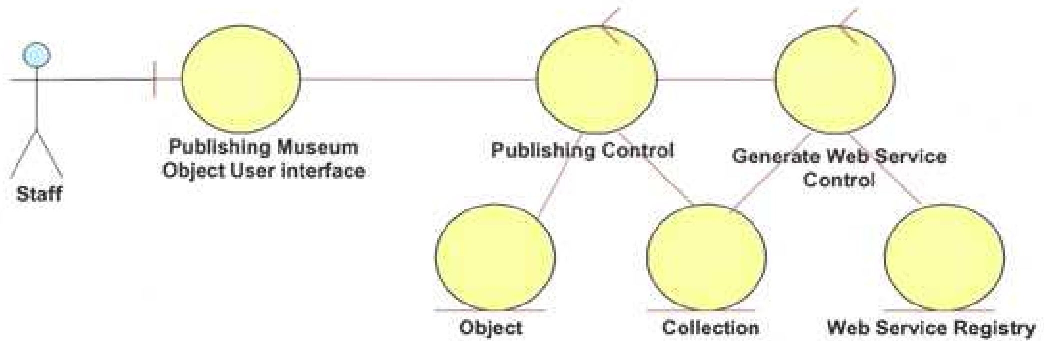
1.11 Object information

This activity includes updating museum objects information



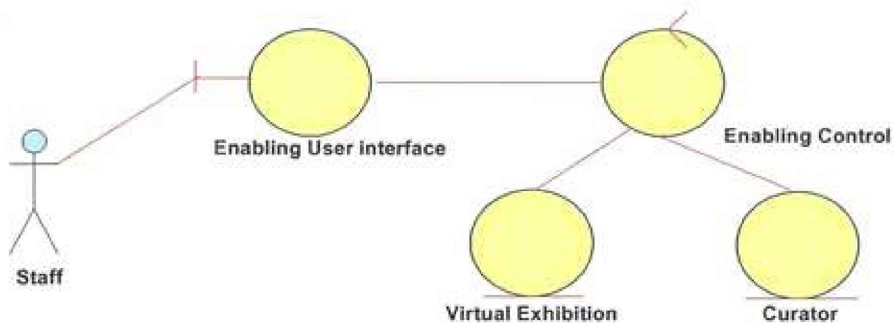
1.12 Publishing museum objects

This activity includes publishing museum collections, and also, this activity includes constructing collections (set of objects)



1.13 Enabling museum activities

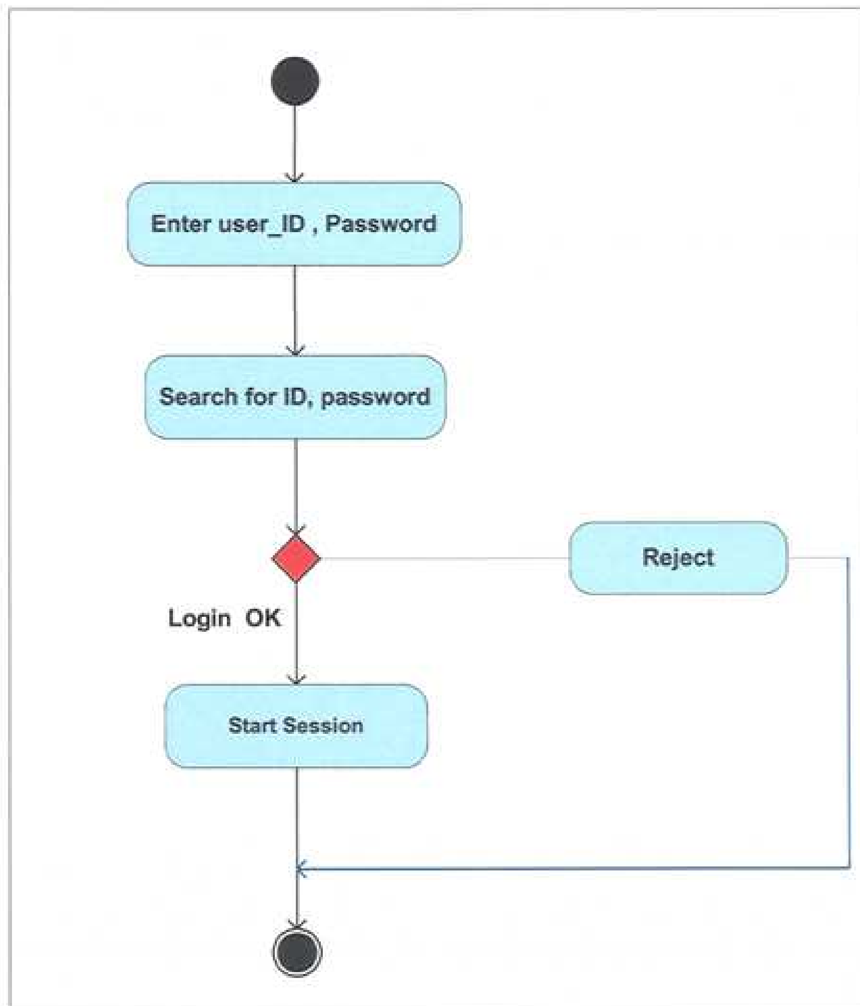
This activity includes constructing collections (set of objects)



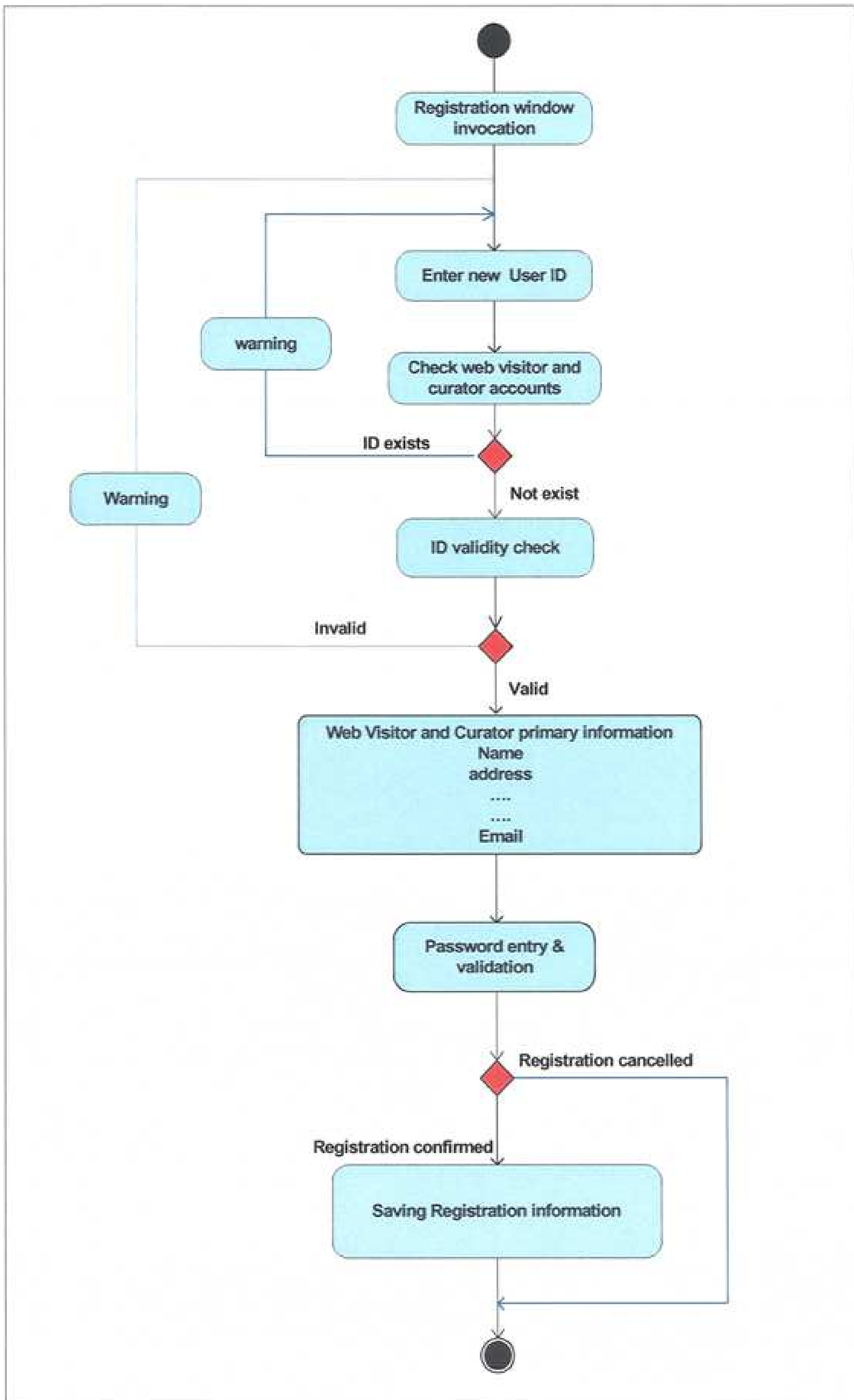
2: Activity Diagrams for all the use case

The following diagrams explain in more details the activity diagram for all the use case in the museum system.

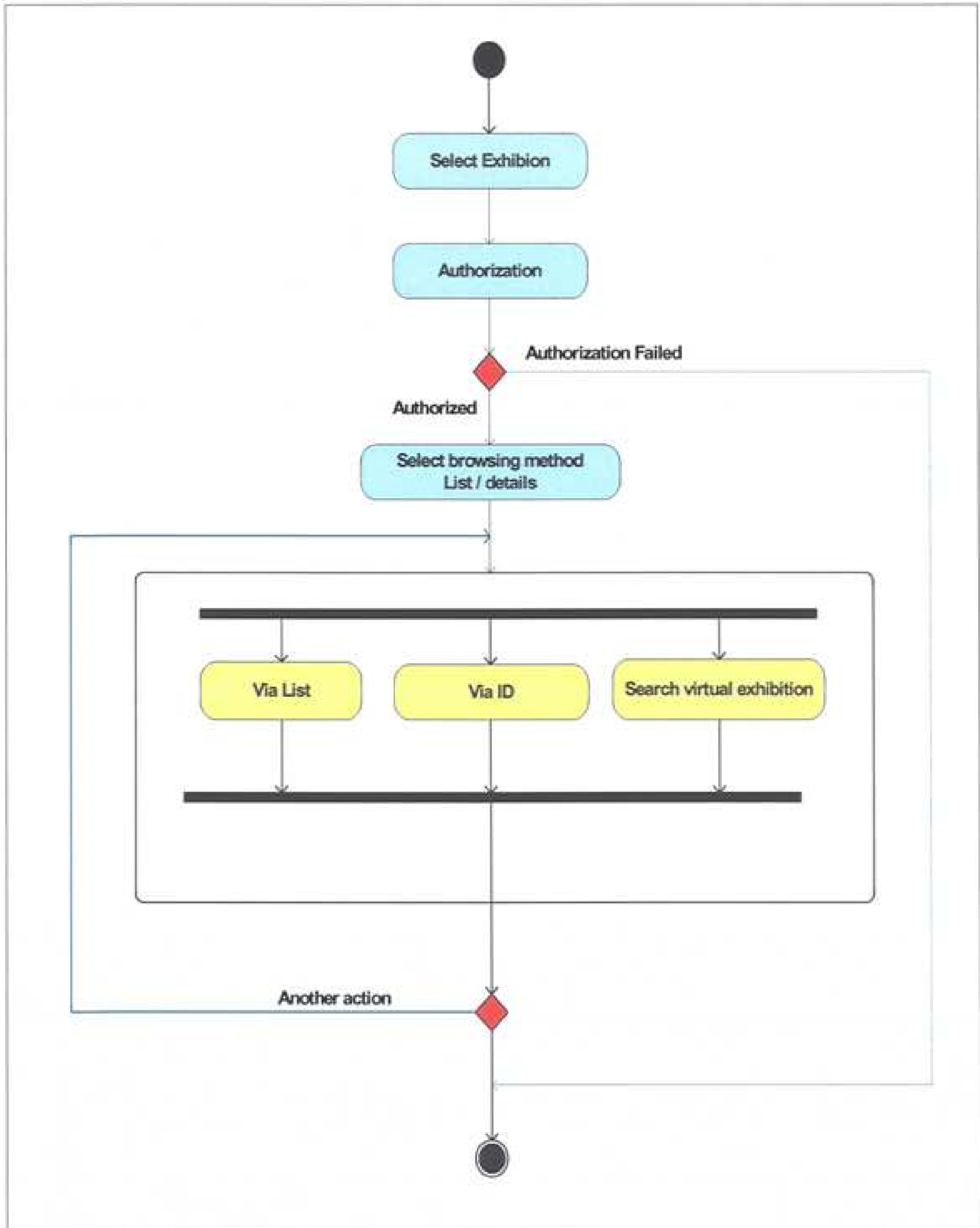
2.1 Login Activity Diagram



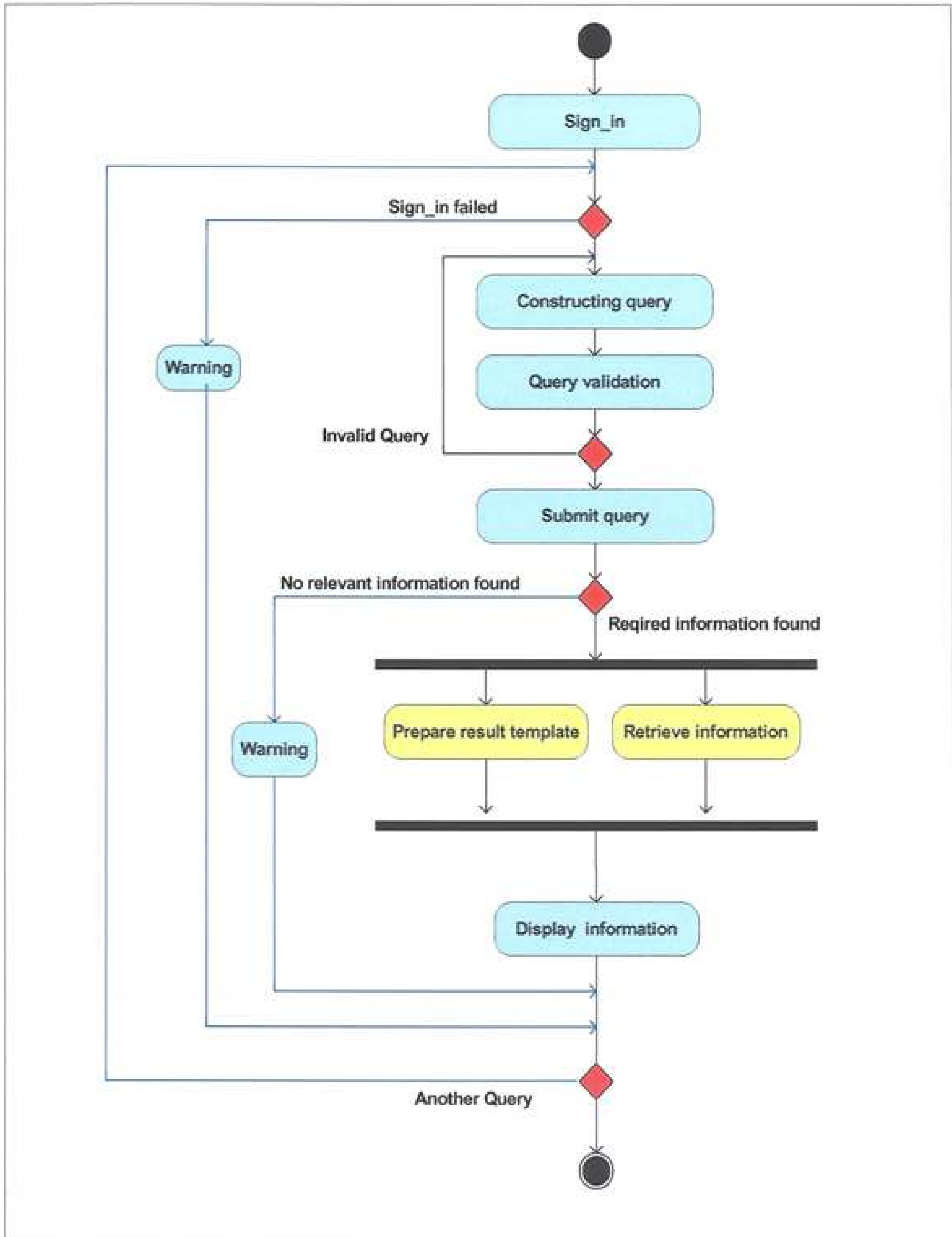
2.2 Registration Activity Diagram



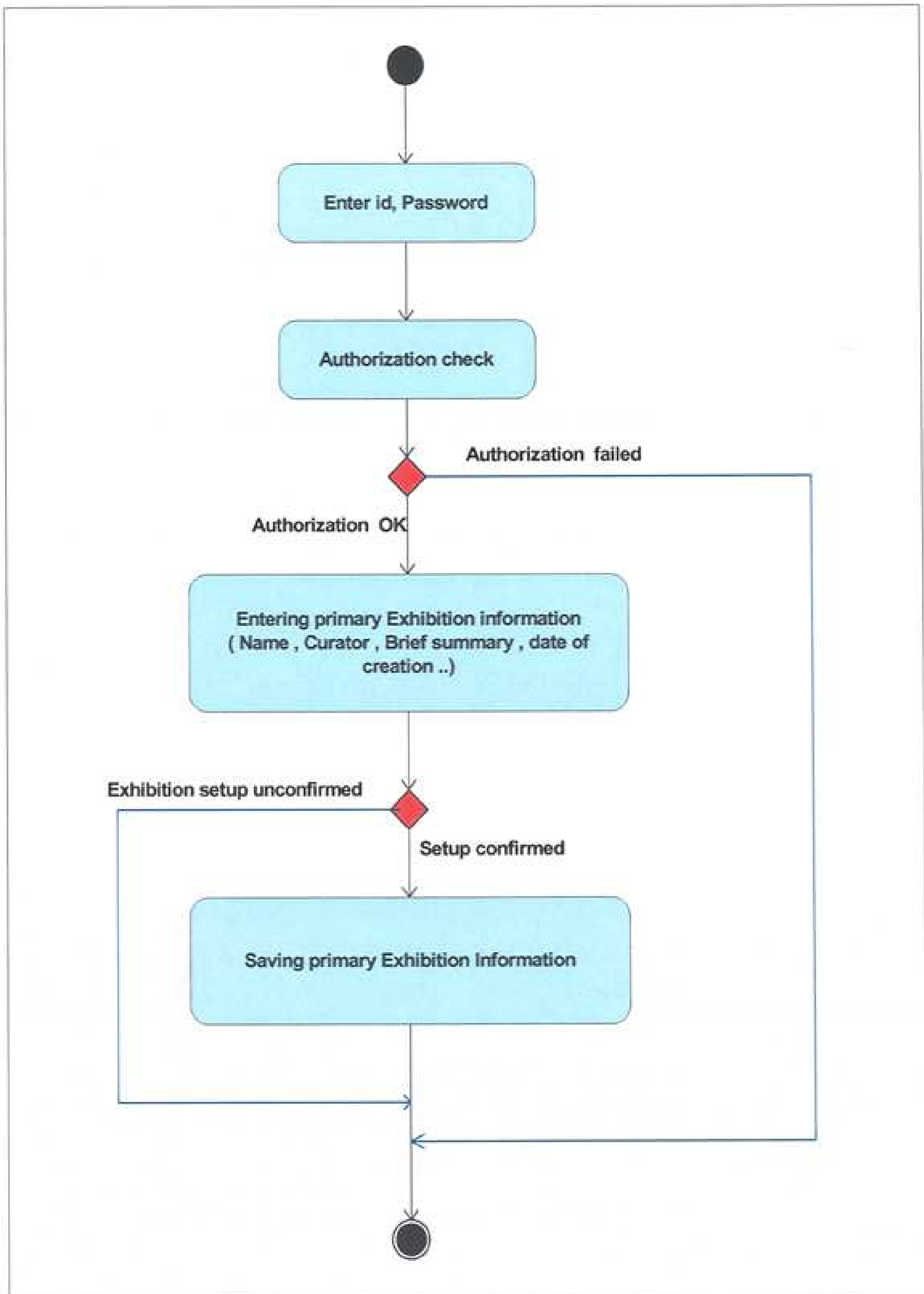
2.3 Browse Virtual Exhibition Activity Diagram



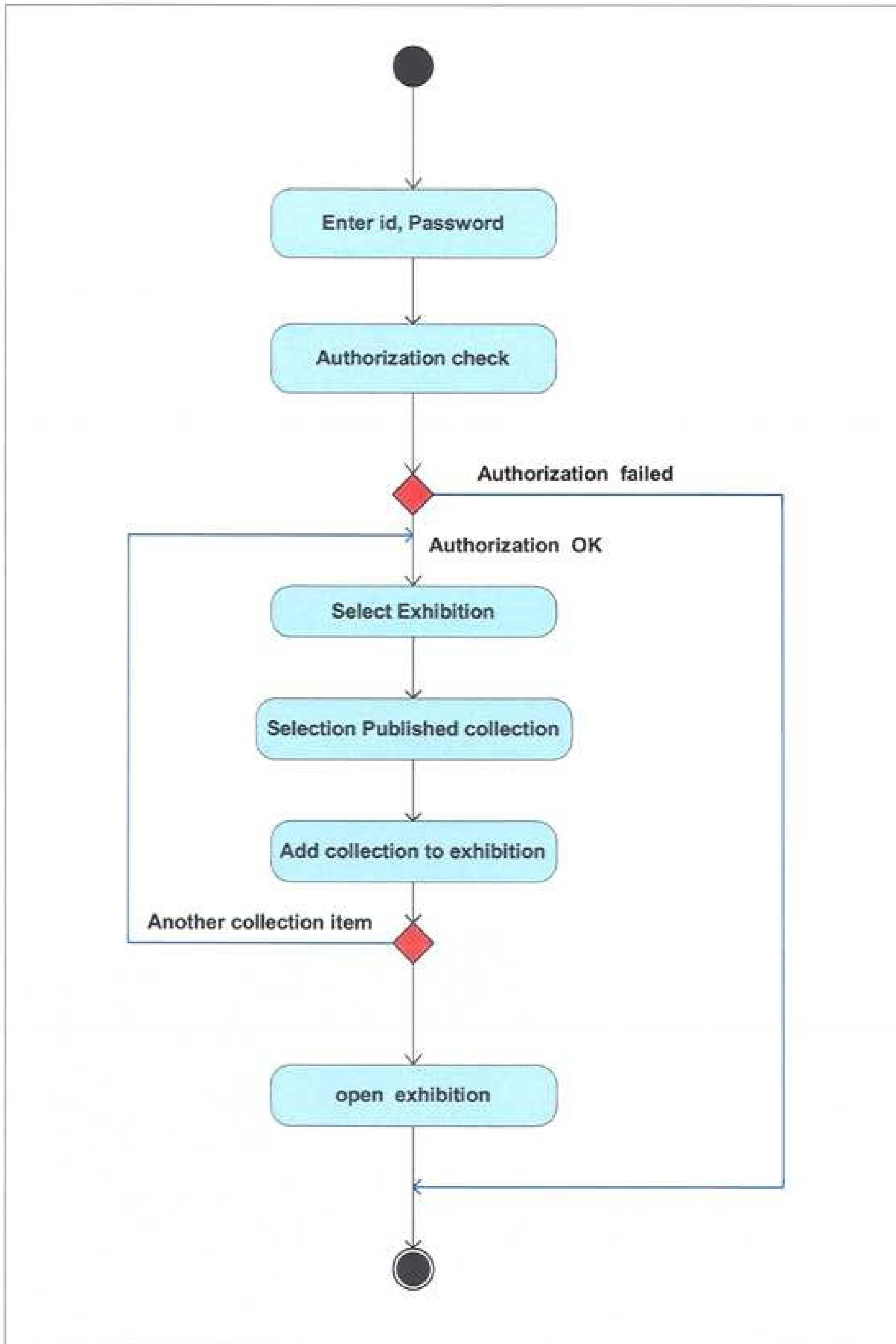
2.4 Search and Display Object Collection Information Activity Diagram



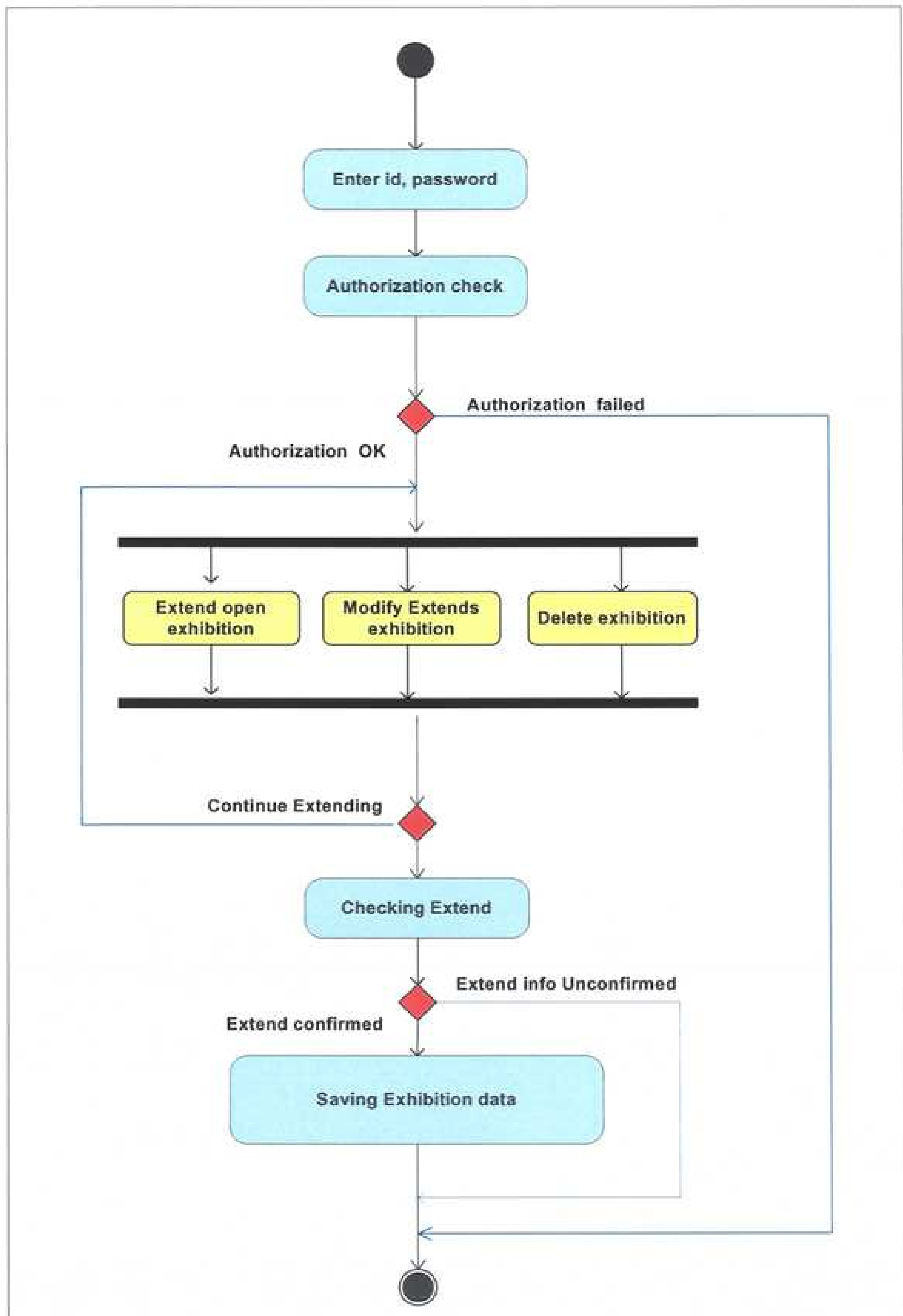
2.5 Setup New Exhibition Activity Diagram



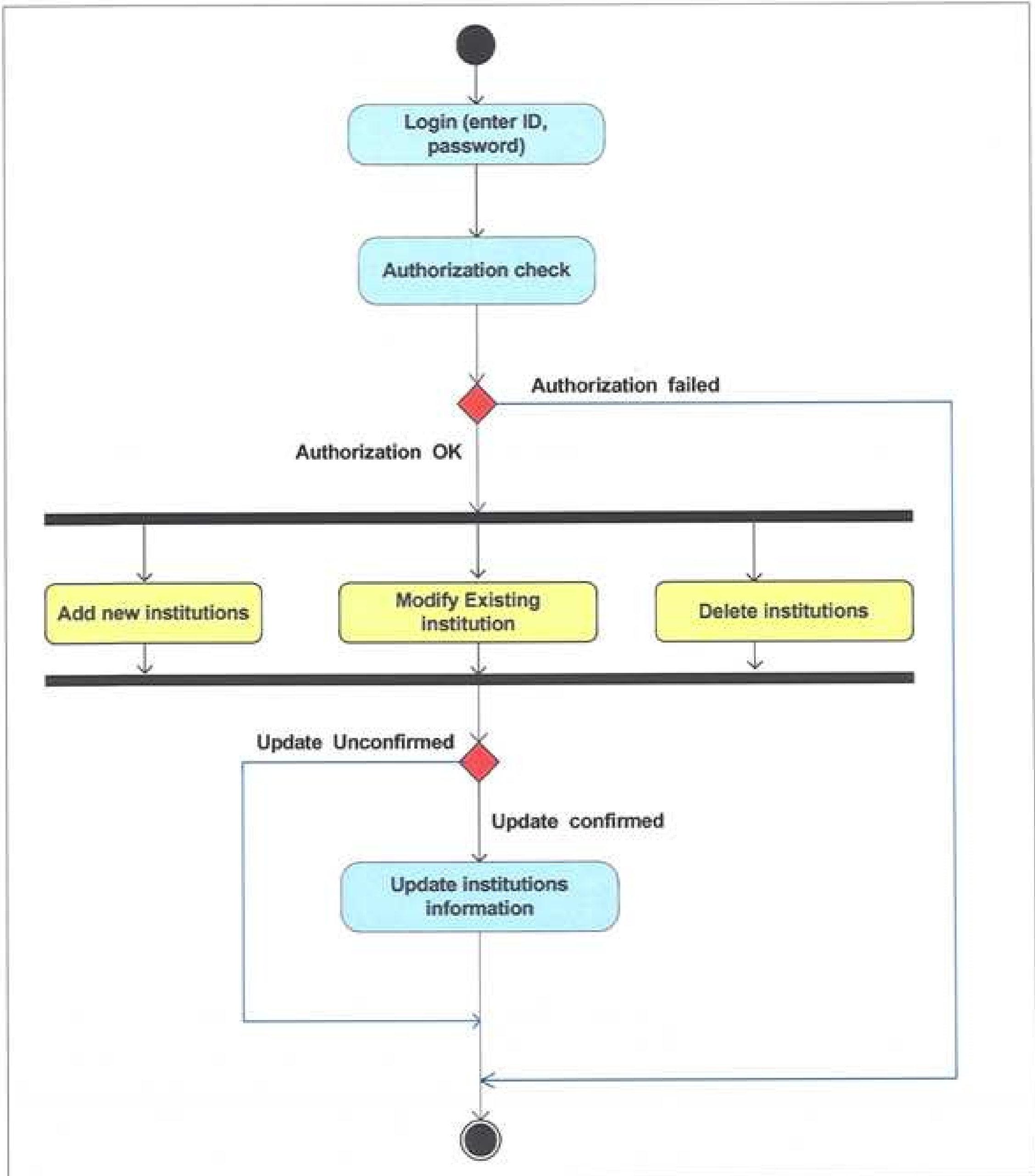
2.6 Assemble Exhibition Activity Diagram



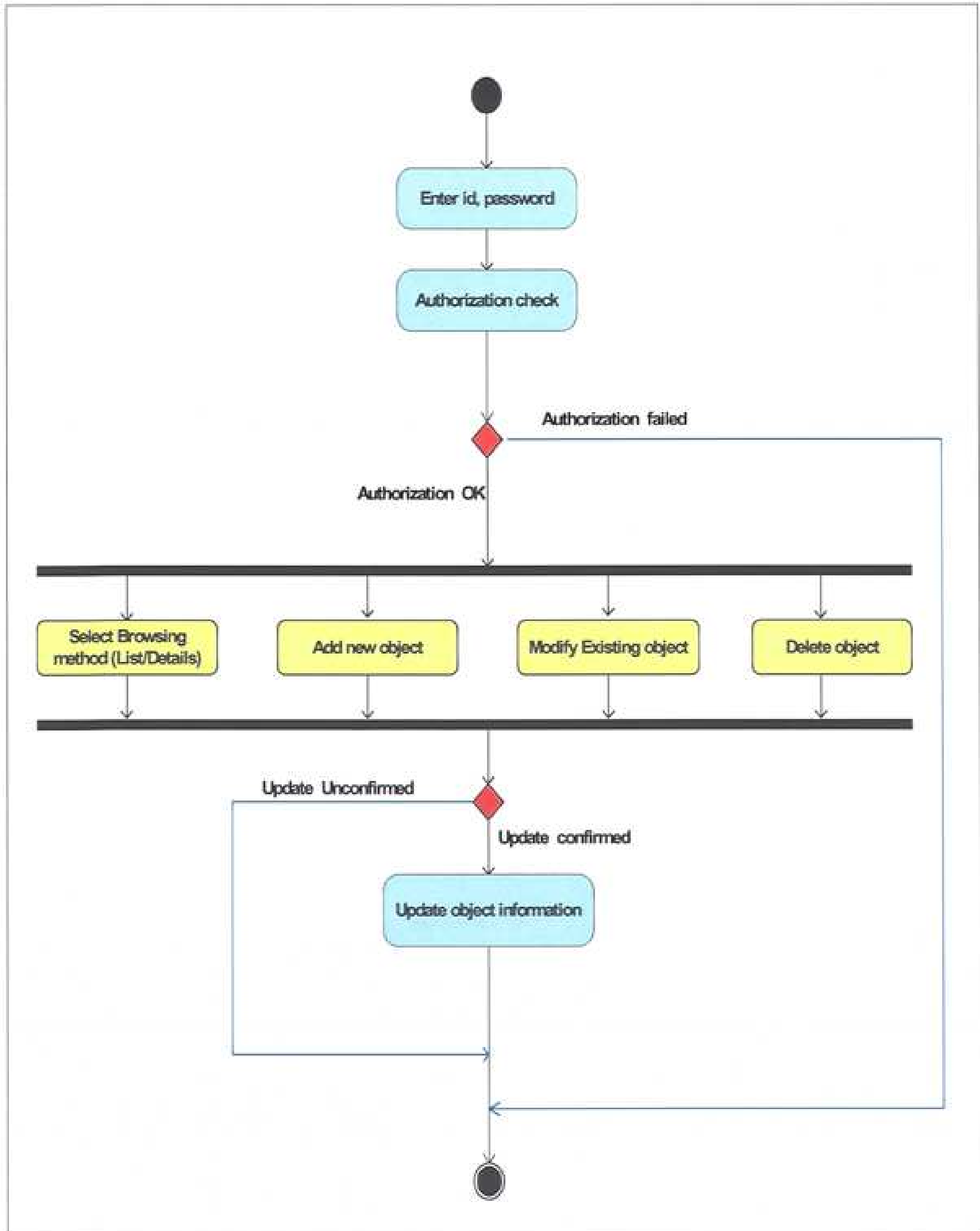
2.7 Extend Exhibition Activity Diagram



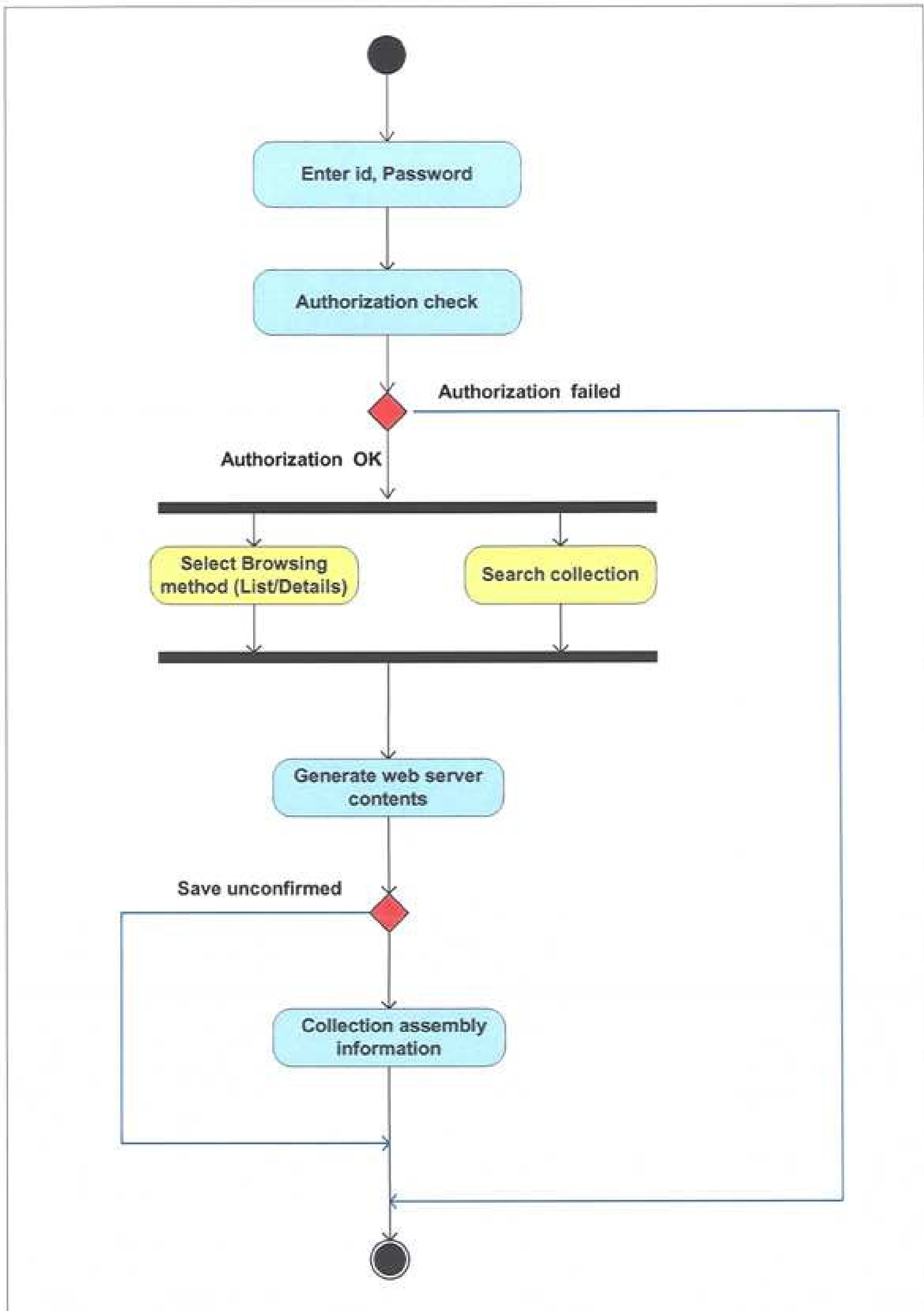
2.8 Institution Information Activity Diagram



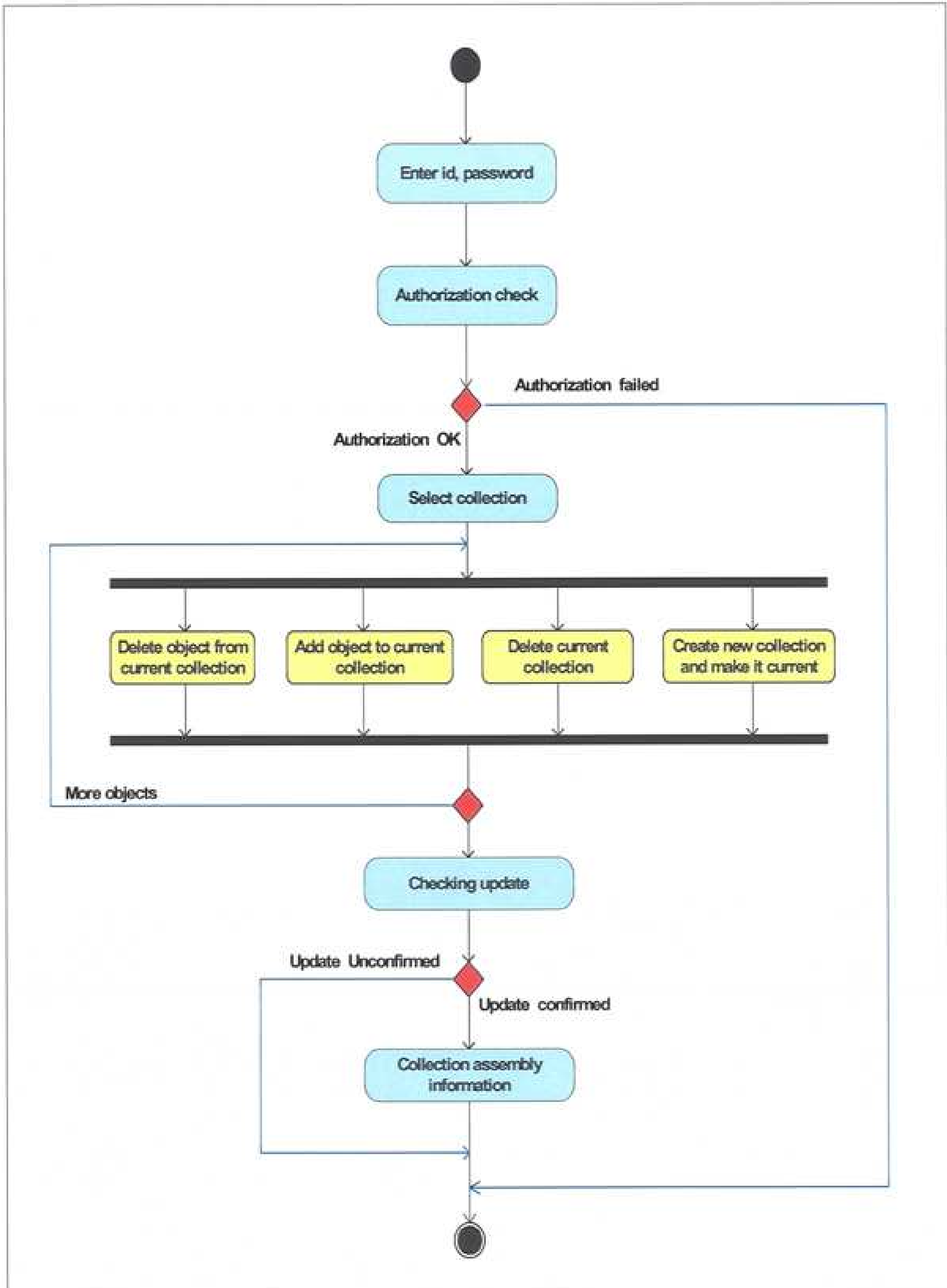
2.9 Object Information Activity Diagram



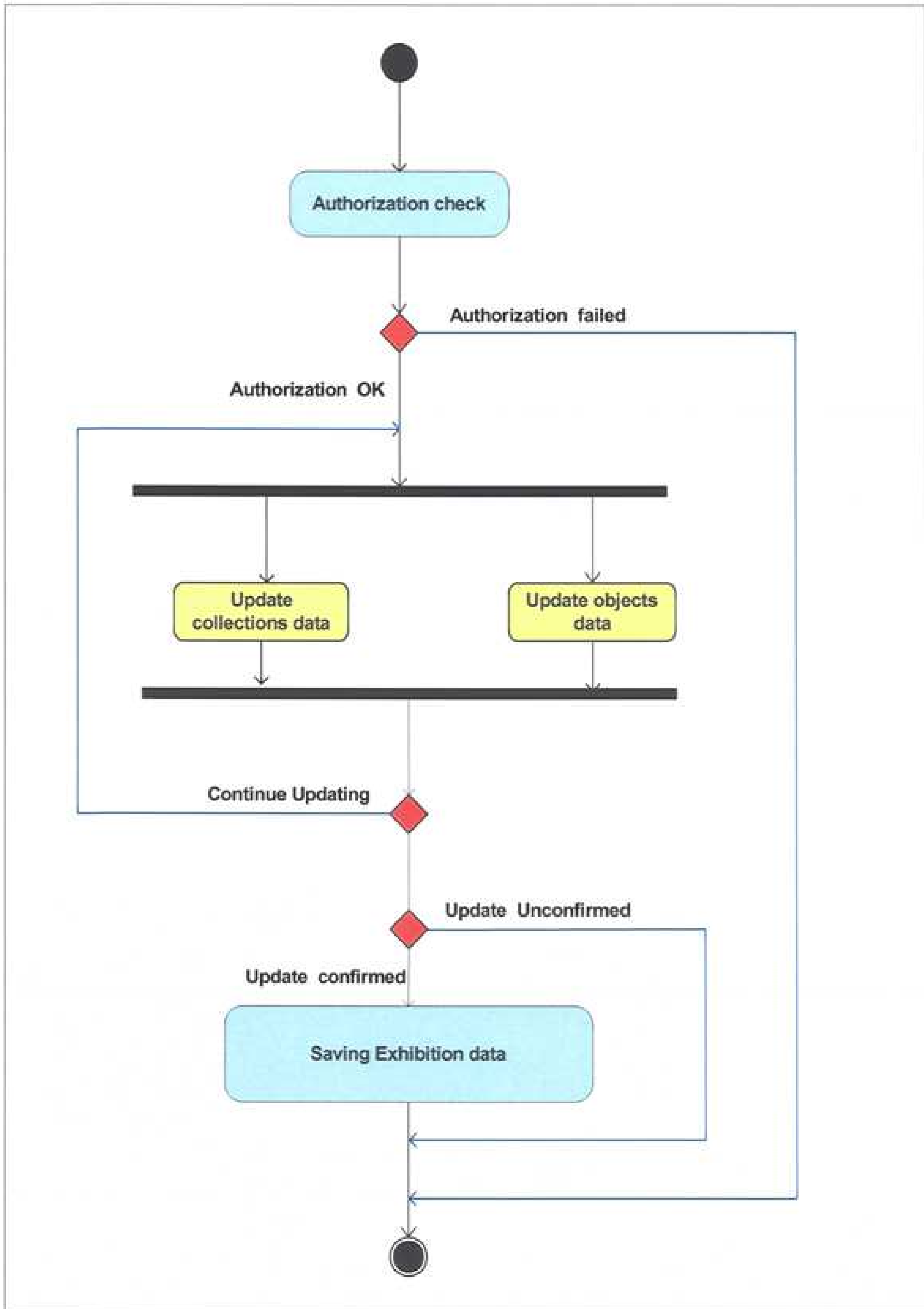
2.10 Publication Museum Objects Activity Diagram



2.11 Collection Assembly Activity Diagram

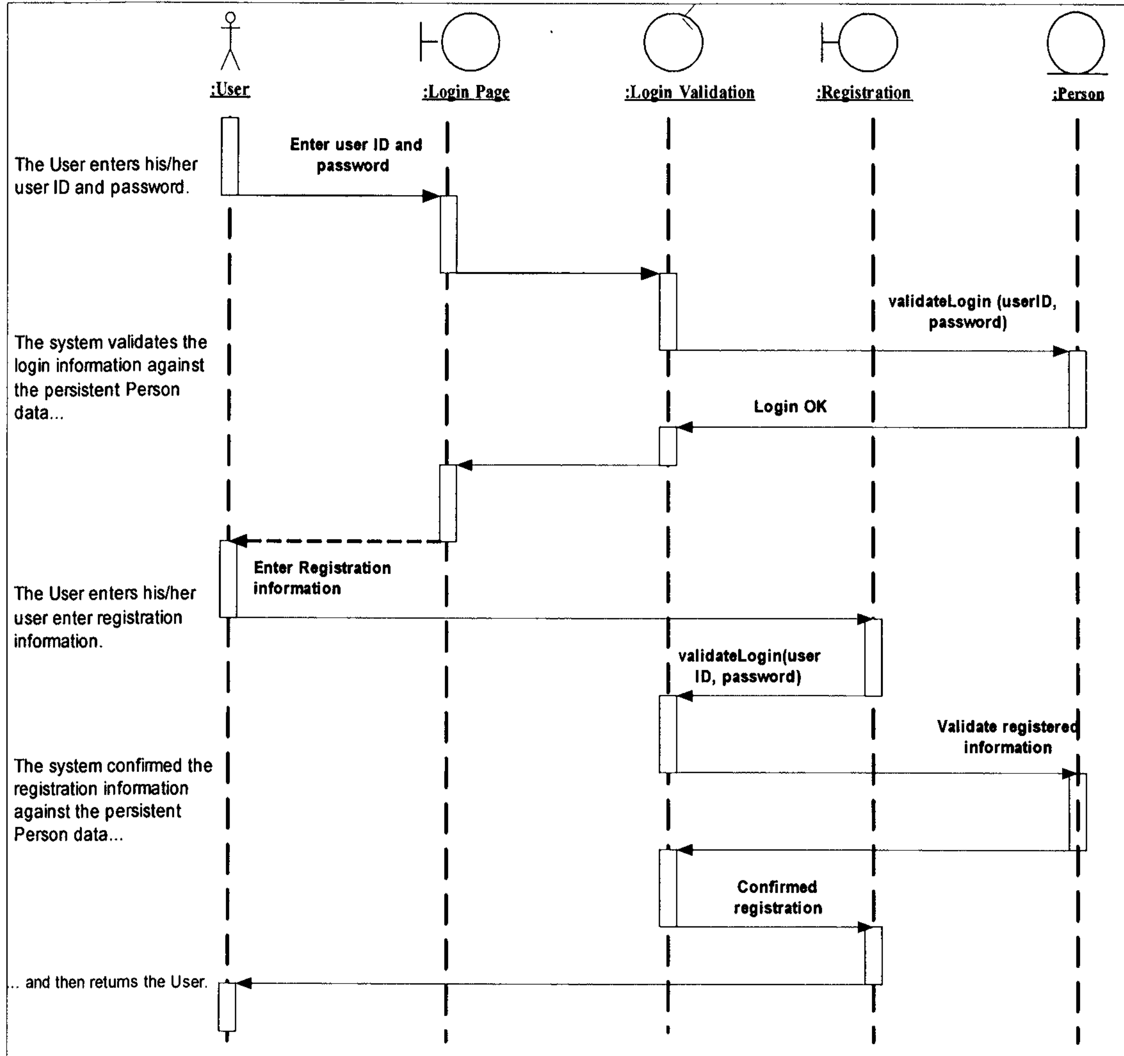


2.12 Update Exhibition Activity Diagram

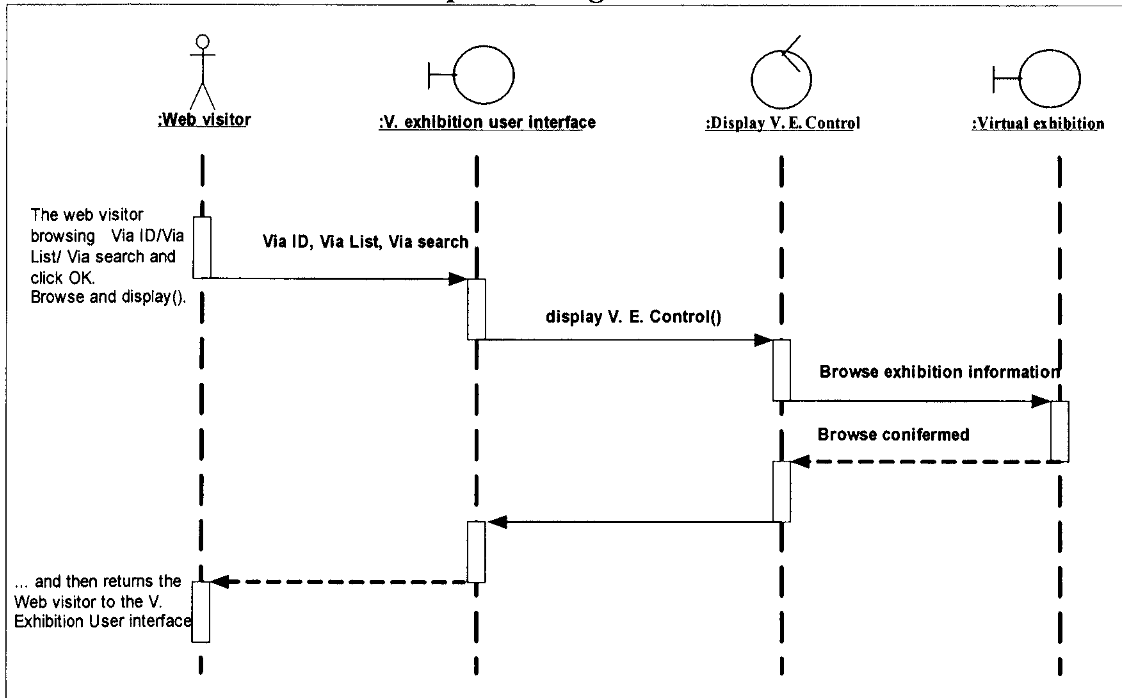


3. Sequence Diagrams

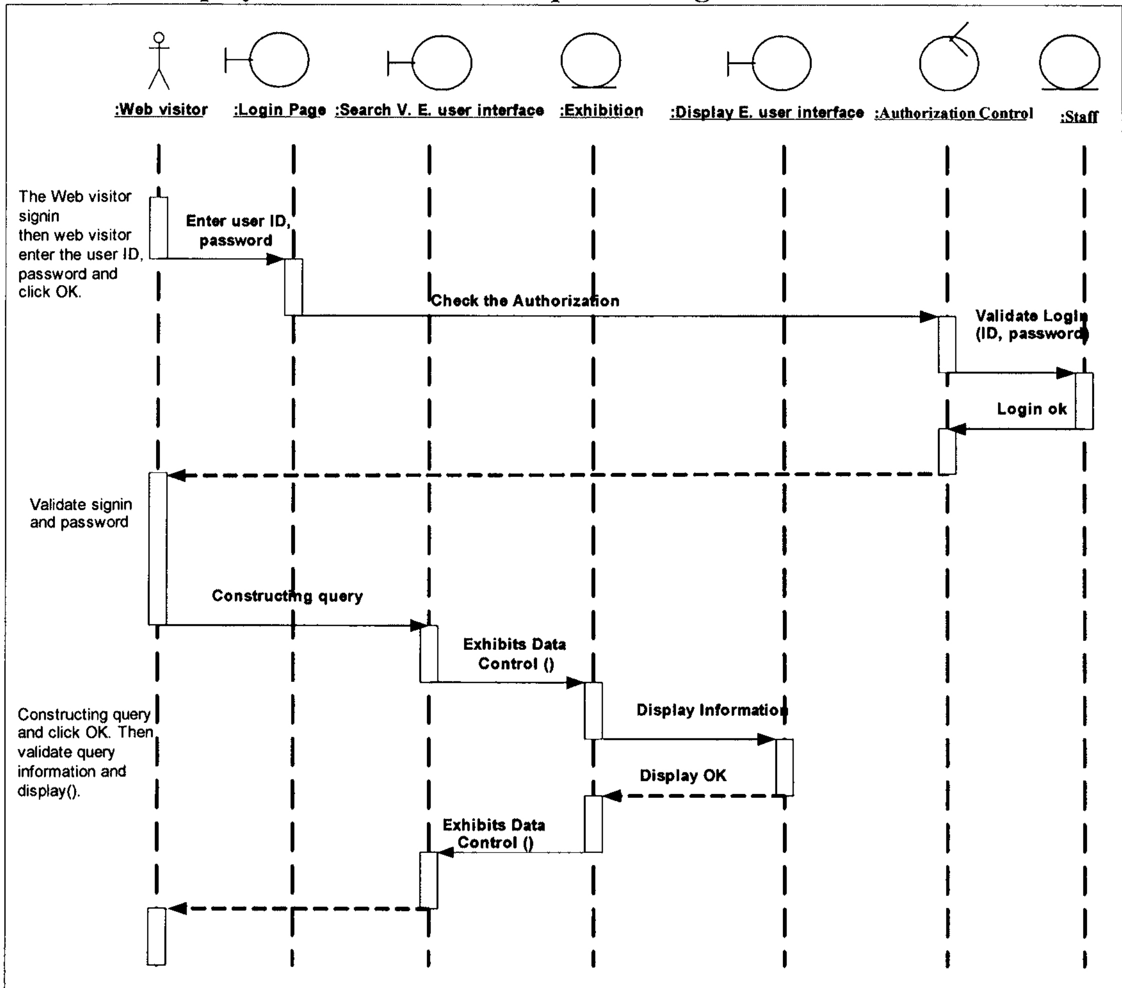
3.1 Log in Sequence Diagrams



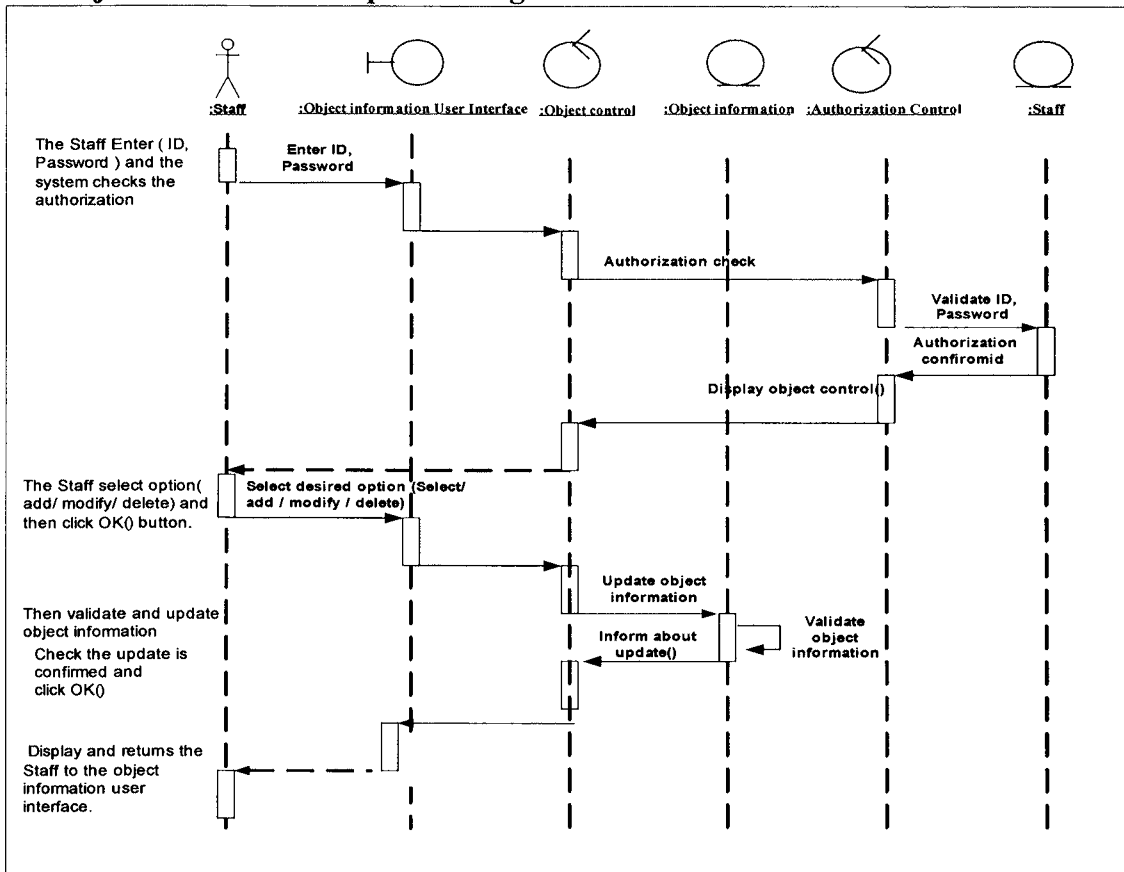
3.2 Browse virtual exhibition Sequence Diagram



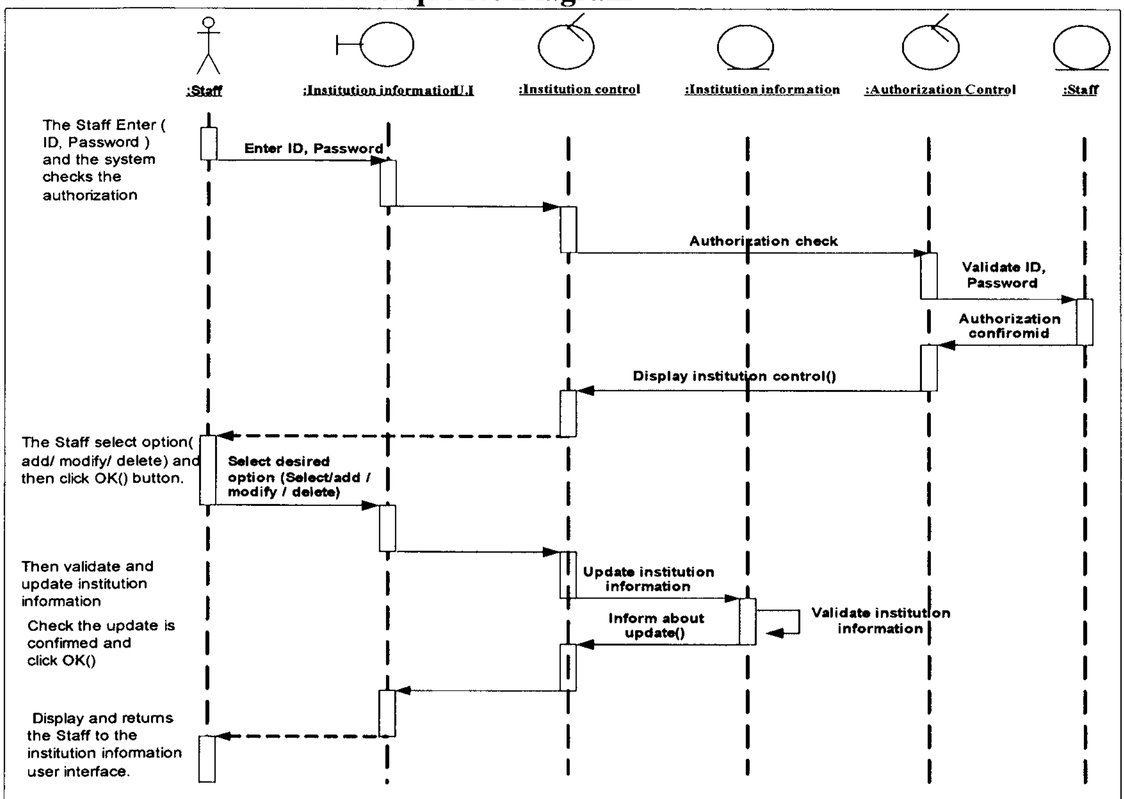
3.3 Search/Display virtual exhibition Sequence Diagram



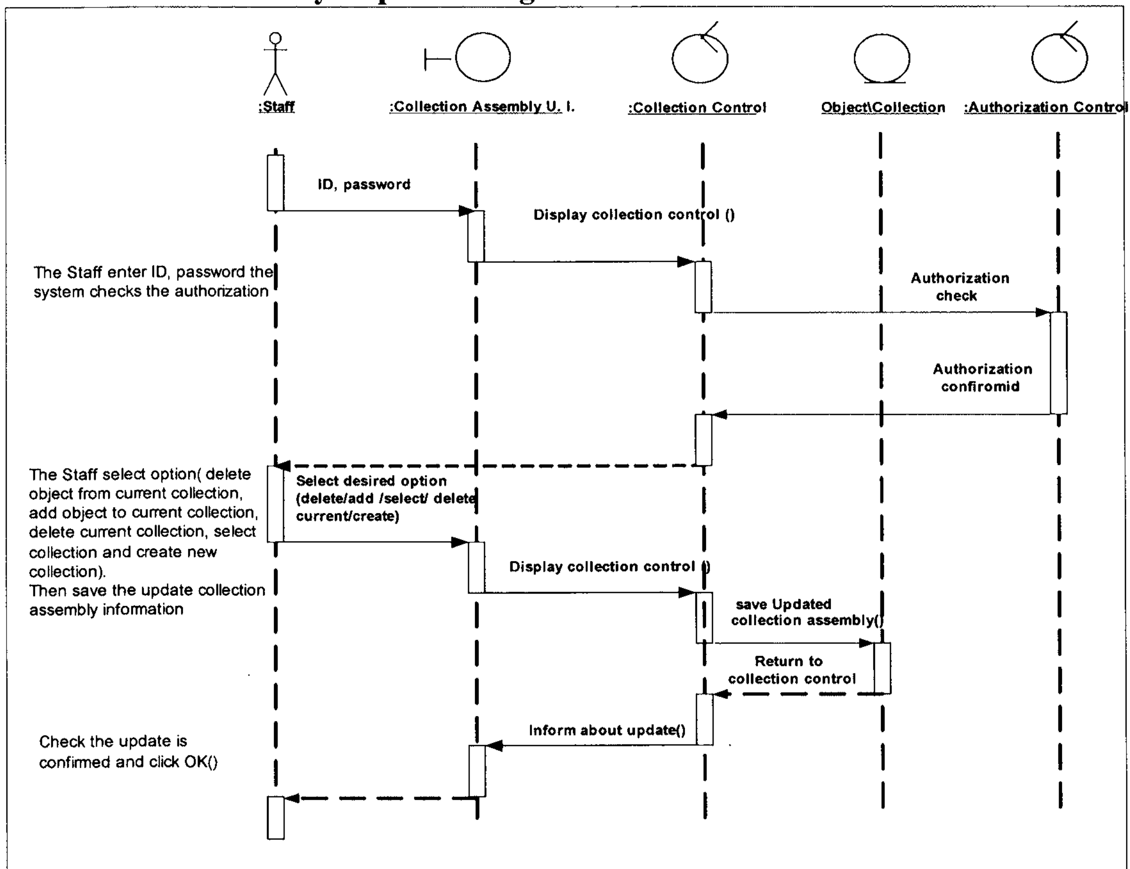
3.4 Object Information Sequence Diagram



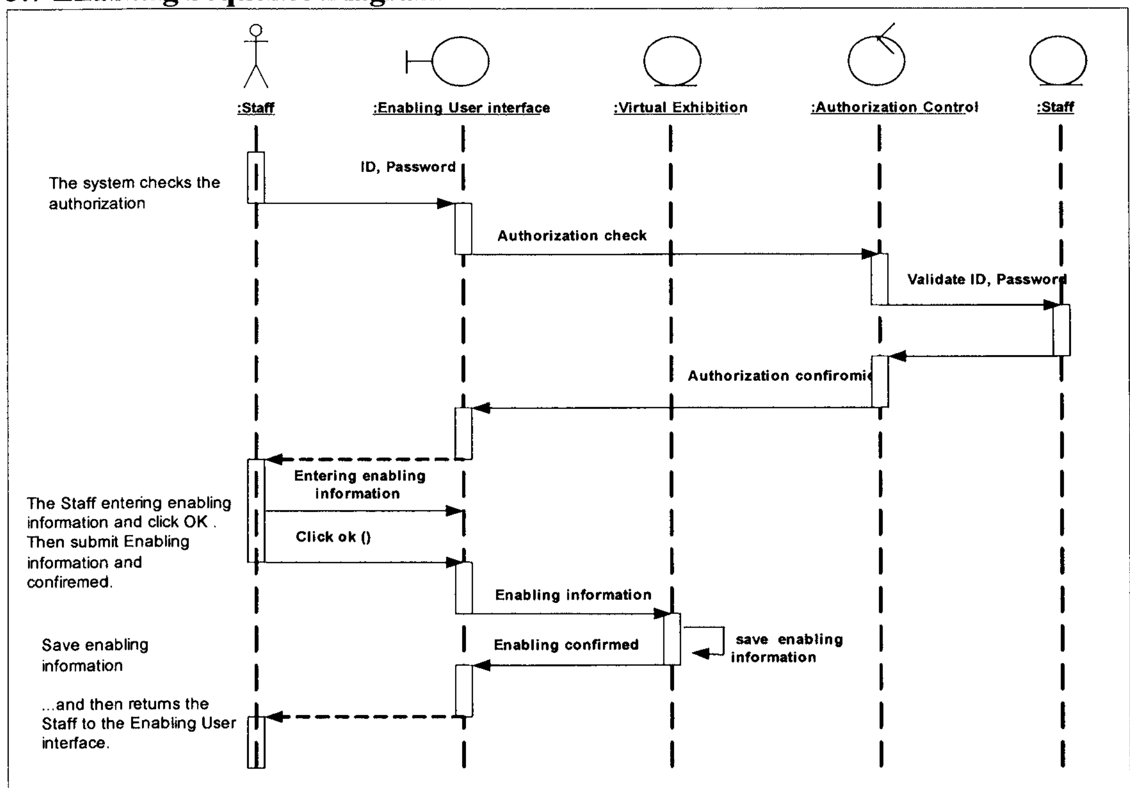
3.5 Institution Information Sequence Diagram



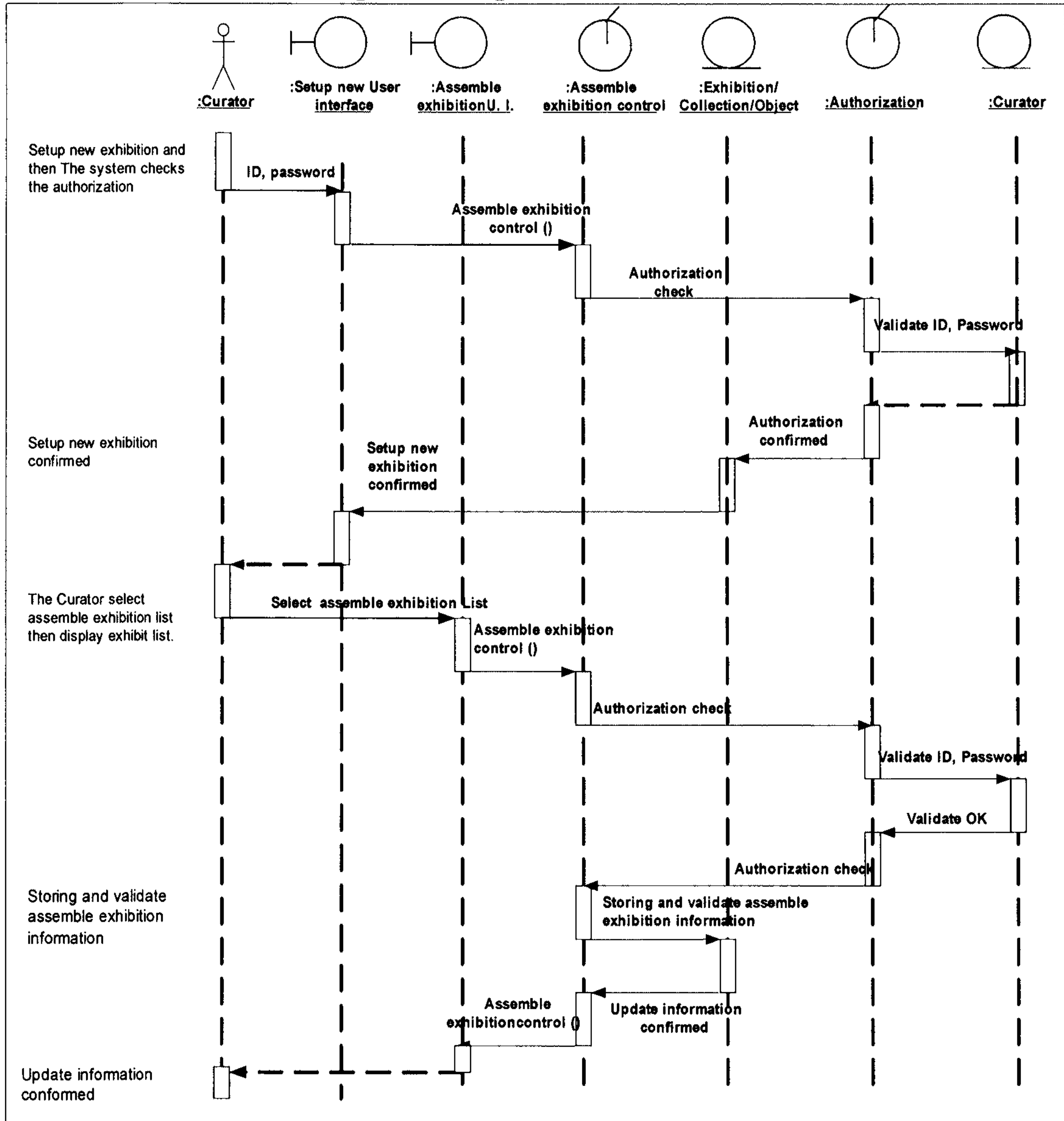
3.6 Collection Assembly Sequence Diagram



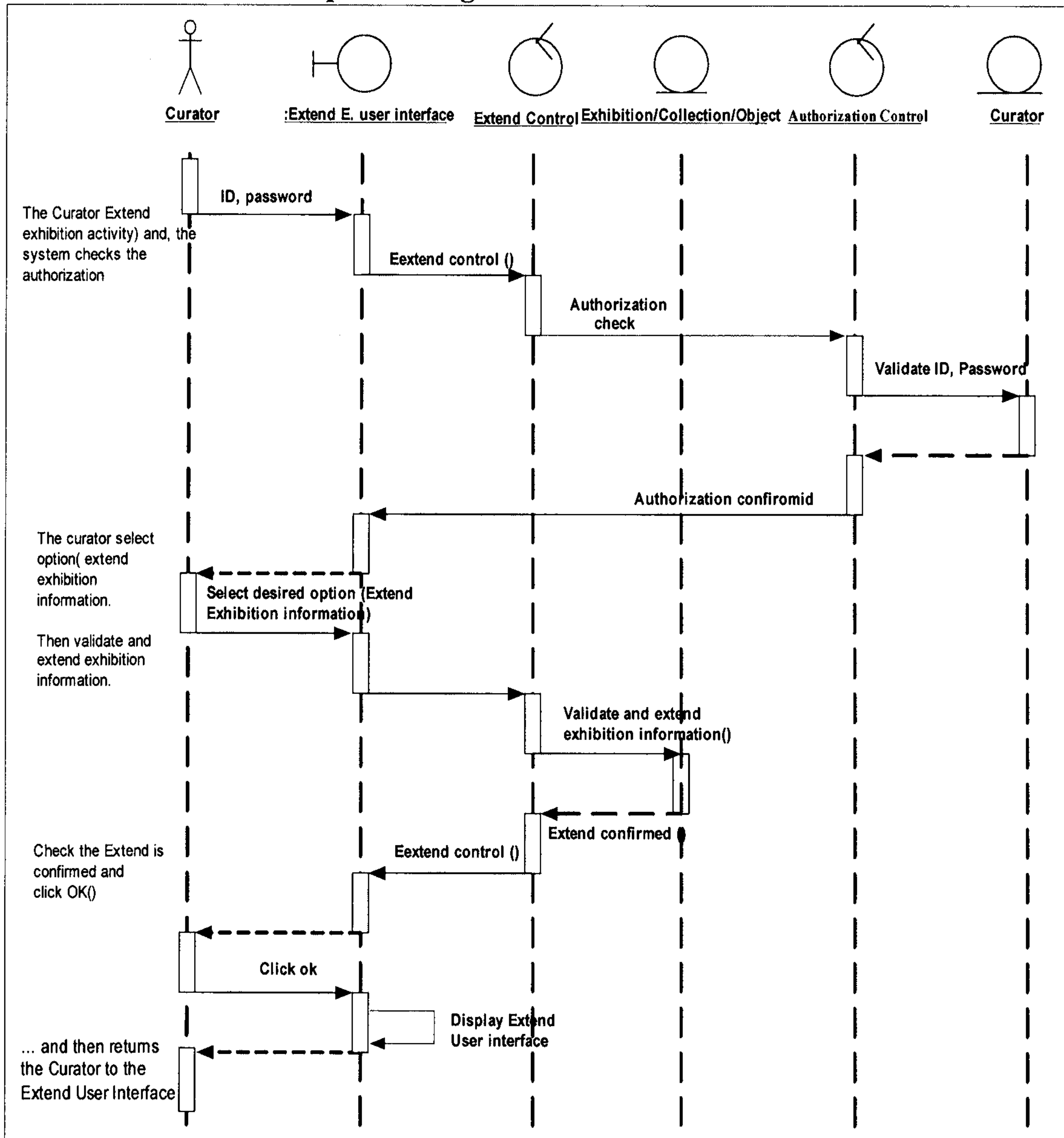
3.7 Enabling Sequence Diagram



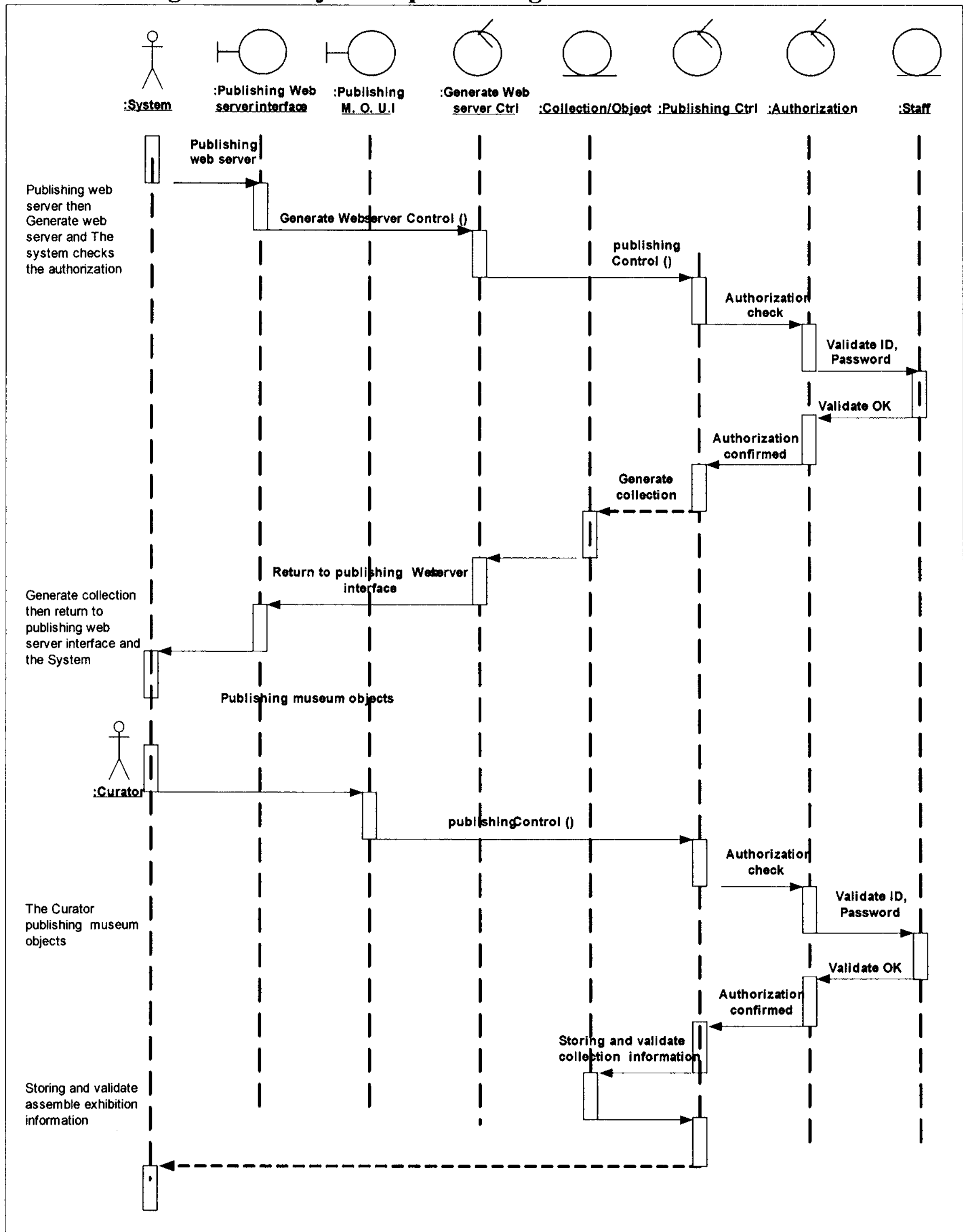
3.8 Assemble exhibition Sequence Diagram



3.9 Extend exhibition Sequence Diagram



3.10 Publishing museum objects Sequence Diagram



3.11 Opening an exhibition Diagram