A FLEXIBLE APPROACH FOR MAPPING BETWEEN OBJECT-ORIENTED DATABASES AND XML

A TWO WAY METHOD BASED ON AN OBJECT GRAPH

by

Taher Ahmed Jabir Naser

Submitted to School of Computing, Informatics and Media

in Partial Fulfilment of the Requirements

for the Degree

Doctor of Philosophy

in Computer Science

University of Bradford

Bradford, United Kingdom

November 2011

Acknowledgments

First and foremost I am very grateful to Allah the Almighty for the blessings, bounties, grace, and guidance He has bestowed upon me throughout my whole life and throughout the journey of the completion of my research and thesis.

Also I would like to express my deep gratitude and sincere appreciation to my thesis advisors; **Mr. Mick Ridley and Professor Reda Al Hajj** whom so patiently supervised and encouraged me throughout my research journey. Their continuous support and valuable advises were very decisive. My great thanks go to both of them for their full support during my study journey and for their valuable suggestions and guidance during my research. Their precious advises regarding thesis structure and suggestions for different corrections was indispensable and inevitable. Without their guidance and encouragement, I would not been able to accomplish this mission.

My gratitude also goes to my parents, my wife Eman, and to my children Mohammad, Hana'a, Abeer, Lamis, and Abdullah for their supplication, patience and support.

Abstract

One of the most popular challenges facing academia and industry is the development of effective techniques and tools for maximizing the availability of data as the most valuable source of knowledge. The internet has dominated as the core for maximizing data availability and XML (eXtensible Markup Language) has emerged and is being gradually accepted as the universal standard format for platform independent publishing and exchanging data over the Internet. On the other hand, there remain large amount of data held in structured databases and database management systems have been traditionally used for the effective storage and manipulation of large volumes of data. This raised the need for effective methodologies capable of smoothly transforming data between different formats in general and between XML and structured databases in particular. This dissertation addresses the issue by proposing a two-way mapping approach between XML and object-oriented databases. The basic steps of the proposed approach are applied in a systematic way to produce a graph from the source and then transform the graph into the destination format. In other words, the derived graph summarizes characteristics of the source whether XML (elements and attributes) or object-oriented database (classes, inheritance and nesting hierarchies). Then, the developed methodology classifies nodes and links from the graph into the basic constructs of the destination, i.e., elements and attributes for XML or classes, inheritance and nesting hierarchies for object-oriented databases. The methodology has been successfully implemented and illustrative case studies are presented in this document.

Table of Contents

List of Tables

List of Figures

Publications Associated with this Research

1. Taher Naser, Keivan Kianmehr, Reda AlHajj, and Mick J. Ridley. "Transforming Object-Oriented Database into XML". *In IEEE International Conference on Information Reuse and Integration (IRI-2007),* August 2007.

2. Taher Naser, Reda AlHajj, and Mick J. Ridley. "Reengineering XML into Object-Oriented Database". *IEEE International Conference on Information Reuse and Integration (IRI-2008)*, July 2008.

3. Taher Naser, Reda AlHajj, and Mick J. Ridley "Flexible approach for representing object-oriented databases in XML format". *Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services. ACM SIGWEB,* pp 430-433, 2008.

4. Taher Naser, Reda AlHajj, and Mick J. Ridley. "Two-Way Mapping between Object-Oriented Databases and XML". *Informatica 33*, pp 297–308, 2009.

5. T. N. Jarada, A. M. Elsheikh, T. Naser, K. Chung, A. Shimoon, P. Karampelas, J. Rokne, M. Ridley and R. Alhajj, "Rules for Effective Mapping between two Data Environments: Object Database Language and XML". *In Recent Trends in Information Reuse and Integration. Springer-Verlag,* 2011.

Chapter 1

Introduction

1.1 Markup Languages Overview

The World Wide Web (WWW) has made a revolution in disseminating information over the internet. Nowadays, business information, library information, research papers, software downloads, and all other types of information are smoothly exchanged over the internet. People have already accomplished major steps towards moving into the electronic world; the classical paper based storage, transfer and sharing of information is gradually becoming part of the history. Electronic governments are even possible in many of the developed countries. The information to be electronically shared have been traditionally published using the Hyper Text Markup Language (HTML) [36, 59].

HTML was originally designed by Berners Lee at CERN (European Council for Nuclear Research). It is based upon the well-known meta-language Standard Generalized Markup Language (SGML) [35]. HTML is composed of a fixed set of tags that specify how to display information (presentation) rather than identifying the content of the data. This markup language tells the browsers how to manipulate and display a document: header, title, font, colour, paragraph and the like. HTML has limitations on segregating the presentation and the content (data) that makes it difficult to manage dynamic web information. The search for a piece of data in HTML is very difficult because there is no indication for the meaning of the data, so the results could be misleading. For instance, someone looking for the world "left"

will get links involving different semantics for the word “left” including left as direction indicator, left-wings, left the place, left brain and so forth. Due to HTML shortcomings and drawbacks, the semi-structured and self-describing eXtensible Markup Language (XML) has emerged to fill a gap and satisfy a need for platform independent language. XML has some other attractive features including extensibility, flexibility, etc.

The World Wide Web Consortium (W3C) has defined XML [76]. It is a meta-language derived from the well-known SGML meta language. XML can be used to define other application/domain centric markup languages. XML is a text-based markup language that is fast becoming the standard for data interchange on the web. It is the best choice for transferring data cross-platform over the internet and so is becoming a standard for platform independent data exchange. In addition, it is extendable in the sense that any desirable tag can be defined, so this will not rely on browsers makers to incorporate new tags in their products if needed. Rather, XML uses tags to identify and classify the content of the data; in other words, the tags are intended to show the contextual meaning of the data. For best understanding, tags are analogous to field names in a conventional data file or to column names in a relational table. Combining XML with the traditional HTML provides an attractive markup tool for publishing data on the web. However, data exists in different forms ranging from completely unstructured text to totally structured databases. While HTML in isolation as well as the combination of HTML and XML have been successfully used for dealing with unstructured and semi-structured documents, they are less used to directly handle structured databases. Hence, the research community has realised the need for tools capable of transforming between XML and the

different structured database platforms; object-oriented and relational are the two most popular data models and hence are the basic models that has to have dominated the XML-based transformation.

1.2 Motivation

Academia and industry have widely accepted the fact that XML is now accepted as the standard format for publishing and exchanging data over the Internet. Due to being platform independent, XML has been used widely as a standard for data interchange on the web. There is a real need for generating XML documents from the databases because of the following facts. First, users may want to publish their data in their web applications. Since XML is the standard format for data exchange over the World Wide Web, users want to provide an XML view of their data. Second, web applications may need to exchange their data with other web applications over the Internet.

As described in the literature, research on XML is mainly focusing on storing and managing XML data using both structured database (mostly relational and object-oriented) and native XML databases. There are generally three main ways to store, query and manage XML documents:

- Store XML as a Text File:

  With XML documents stored as text file, external index can be created and maintained, and a query mechanism could be used to improve the retrieval of data.

- Store XML into XML Enabled Database:

  XML Enabled databases extend traditional relational or object-relational

databases with middleware, mediator, or with an extension to allow storing XML documents, publishing relational data as XML document and querying XML views over the underlying data. These types of XML databases are mostly used for data-centric applications (structured documents and less commonly semi structured documents). Data-centric documents [14] are those XML documents that are well structured and can be easily fragmented and fit into structured relational database (relational tables have been the most widely used structure). Examples for data-centric applications include invoice payments, medical records, stock shares and so on. The middleware layer, mediator, or other database extensions map the logical structure of the XML document presented by XML schemas such as DTD [8] or XML Schema [75] into an equivalent relational database schema. This allows the DB to execute XML queries against the underlying relational database. A XML query is translated into a SQL query [27] which is then executed by the DBMS. The results presented by tuples from the database are transformed into XML format by tagging them as elements and/or attributes to generate the XML document. XPERANTO [19] and SilkRoute [31] are two typical examples for the middleware solutions, and Agora [47] is an example of an XML mediator.

- Store XML into Native XML Database:

  The term Native XML database first became popular from the campaign of the XML server Tamino [69], a product from Software AG produced in 1999. The internal model of this database is based on XML structure. The fundamental data structure unit is XML document. This means that an XML document can be stored as a whole rather than segmented into pieces as is the case for XML

documents stored in XML Enabled databases. A native XML database like other databases should support transactions, security, multi-user environment, XML query language, etc. Native XML databases are mostly used to store document-centric documents. Document-centric documents [14] are the unstructured and mostly the semi-structured documents that are designed for human use such as books, emails, newspaper, advertisements, news reports, images and so on. Mostly they are read-only documents. They have irregular structure, which means no strict data format or data structuring is manipulating the content other than being well organized as required by the particular domain/application. For instance, a book is divided into chapters; each chapter is divided into sections; a section may be divided into subsections; and both sections and subsections may consist of paragraphs. This structure may differ from one book to another. Also a good example is the newspaper pages where each page has a different structure. The location, size, shape of advertisements, news, weather forecast and so on, do not have a regular structure over different pages.

As described in the literature, there are many approaches for transformation between relational databases and XML documents. IBM DB2 XML Extender [24] is an example where user should provide the tool with the relational input schema as well as the XML output schema; also the user has to map the relational and the XML schemas. SilkRoute, XPERANTO, Oracle 10g, and Agora are also examples of tools that work as middleware between the users and the database systems. The user provides the relational schema and the XML queries. The tool translates the XML queries into SQL queries. These are then executed by the RDBMS. The result is

tagged as XML document. As XML query languages and SQL [27] do not have the same semantics, the mapping is not perfect.

The mapping from XML into object-oriented database facilitates storing XML documents using object-oriented structure is more attractive than the relational or object relational structure as there is more overlap between XML Schema and the object-oriented paradigm than between XML Schema and the relational paradigm (explained with example in Section 3.1). So, due to this similarity of the structure for both object-oriented database and XML, it is more efficient to store and manage XML documents using object-oriented databases. This mapping process is expected to preserve as much information as possible during the transformation.

The transformation of object-oriented databases into XML has received little attention. Since the business data currently stored and maintained in object-oriented database management systems is increasing steadily, it is important to automate the process of generating XML documents containing information from those existing databases. The object-oriented-to-XML transformation involves mapping names of the classes and attributes into XML names of elements and attributes, mapping the inheritance and nesting hierarchies into XML hierarchies, and processing values in an application specification manner.

The similarity of the structure between XML Schema and object-oriented database schema encourages developing an approach for both ways of transformation between XML and object-oriented databases. As a result of the research efforts that produced this dissertation, two basic steps are identified in the process of transforming object-oriented databases into XML. First, an intermediate graph from the given object-

oriented database is constructed. This process requires knowing the meta-data of that object-oriented database schema. The meta-data should be extracted by using the underlying object-oriented database management system constructs or by composing it through scanning the entire database data (refer to Section 5.4). Second, the obtained intermediate graph is to be transformed into XML Schema. The reverse process of transferring XML Schema into object-oriented database is also performed into two steps. First, the XML complexTypes elements are extracted to construct an intermediate graph. Second, the obtained intermediate graph is to be transformed into an object-oriented schema. Detailed description of this process is found in Chapters 3 and Chapter 4.

1.3 Problem Statement

Storing, managing and retrieving XML documents has been a main challenge since XML emerged and adapted for data exchange. Most of the XML documents are now stored in relational or object relational databases. These are the concerns facing the process of managing and storing XML document into relational or object relational databases:

- The mapping from the relational schema into XML Schema and vice versa is specified by human experts. Therefore, when a large relational schema and the corresponding data need to be translated into XML documents, a significant investment of human effort is required to initially design the target schema. This process is error-prone, time consuming and tedious.
- XML document should be fragmented into several pieces before it can be stored into relational or object-relational database. It should be reassembled when the document is retrieved. Because of a lack of referential integrity

between parts and the significant time it takes for reassembling the XML document, query response will be very slow and the process is inefficient.

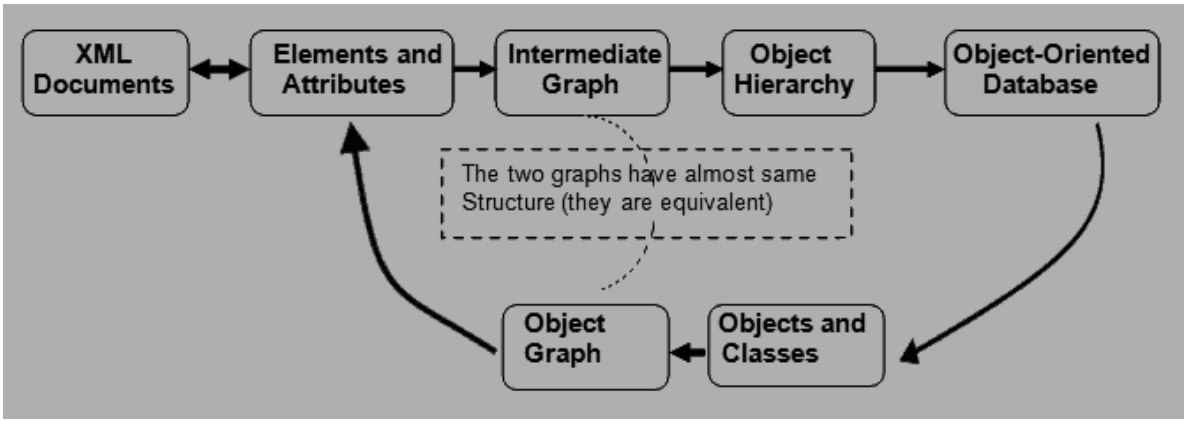


Figure 1.1: Block Diagram for Two Way Mapping Between XML and OODB

The block diagram in Figure 1.1 summarizes the core and the spirit of this research. The process starts with analysing the XML Schema and corresponding XML document. It breaks down the schema into the core components of XML that may hold data; elements and attributes. Then the object graph is constructed from elements and attributes. The object graph is then used as an input to construct the object structure. The object structure is stored as an object oriented database. The reverse process starts by extracting the object structure into object graph and then the object graph is used to construct the XML Schema and related document. The generated graphs are equivalent in term of vertices and edges but they are different in content. Because of the similarities between the structure of the XML Schema and the object-oriented schema, most of the data is preserved during the mapping. The proposed approach could be classified under database reengineering. However, each side of the mapping is considered as independent reengineering process as both

models are classified as recent technologies. In other words, the reengineering of XML into object-oriented database as well as the reengineering of object-oriented database into XML is handled. Each of the two reengineering processes consists of a reverse engineering step and a forward engineering step. The former step derives the basic characteristics of the source model to be reengineered and the latter step derives the target model.

This work argues that the mapping between XML and object-oriented databases is the most natural because of the very narrow semantic gap between the two models as compared to the semantic gap between XML and each of the relational and the object-relational model. The mapping between the object-relational schema and the XML Schema is inefficient because it is completed in two steps. First, the XML Schema is mapped to an object schema. Second, the object schema is mapped to a relational schema. To sum up, the problem tackled in this dissertation is simply the two way mapping between XML and object-oriented database.

1.4 Research Issues

When we are seeking solutions for efficient mapping between XML structure and object-oriented paradigm, many challenges and critical issues had been raised. These challenges are explained as:

a. Existing research for mapping between XML and OODB is inadequate (refer to Section 2.3.2).
b. XML Schema is a complex structure comparing to other XML schemas such as DTD.

c. Different object oriented databases have different data model schema structure.
d. Mapping should support nesting and inheritance.
e. An intermediate structure such as a directed object graph is preferred to be incorporated to clearly demonstrate the mapping process.

Due to mentioned challenges, the following research questions have been raised:

1. Can we define a generic object oriented database model that can be a subset of most of objects oriented databases models?
2. Can we have a successful two way mapping between the complex XML Schema structure and a generic object oriented database schema structure using an object graph? Why and how to incorporate a directed object graph into the mapping process?
3. Does the mapping process support inheritance and nesting, and can this process generate flat and nested XML schemas and their corresponding documents?
4. How to minimize the user involvement and how to recover database meta-data if not exists?
5. Can the concepts of this approach apply successfully to other way of mapping (mapping between XML and ODL as an example)?

1.5 Contributions of the Research

Contributions of the work described in this dissertation could be enumerated as follows.

- Analysing the schema (meta-data) and the content of object oriented database and then extract an intermediate graph called Object Graph (OG) - similar to the entity-relationship diagram in relational databases - from that database. This research proved that object graph can be used successfully for mapping between object oriented databases and XML. A sample object graph is shown in Figure 1.2. For more details, refer to Chapter 3.

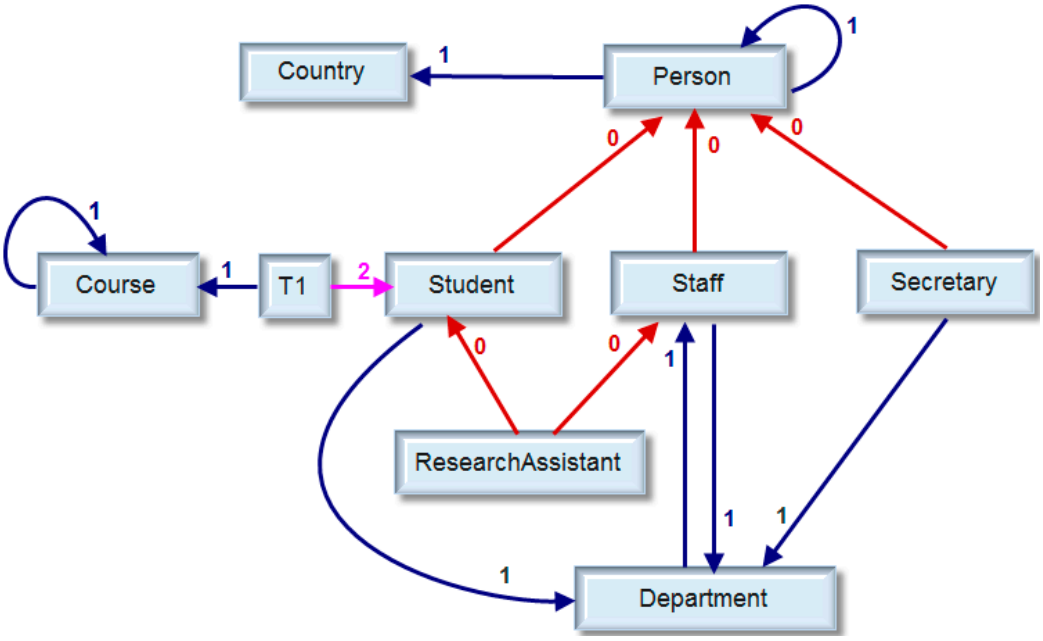


Figure 1.2: Object Graph

- Transforming the derived object graph to flat or nested XML Schema as specified by the user; and then transfer database content accordingly.
- Transforming an XML Schema into object-oriented schema, and then transfer the XML document(s) to the developed object-oriented database. This is a reverse process of object-oriented schema into XML Schema transformation. This is performed by creating an OG derived from the XML Schema. The OG

is then transformed into the object-oriented schema and then the XML document is produced accordingly.

All these contributions are integrated into a rigorous framework capable of providing a workable solution for the defined problem. The produced solution is described in this dissertation according to the outline given in the next section.

1.6 Outline of the Thesis

The thesis is composed of 7 chapters, including this introduction chapter. Chapter 2 gives an overview of the necessary background and describes the related work. It describes XML primer, XML schemas, XML databases, XML query languages, object oriented concepts, and mapping approaches between XML and different databases such as relational, object-relational, and object-oriented. Chapter 3 explores the details of transforming an object-oriented database into XML Schema and XML related document. This chapter includes the details of the approach that has been followed in this transformation, the definition of the object graph, and the algorithms used for this purpose. It discusses the XML Schema structures and the mapping techniques used in this work. Also, it shows how to generate flat and nested XML Schemas and documents. Chapter 4 explores the reverse process of the approach followed in Chapter 3. It shows the details of the work done about the transformation of XML Schema and corresponding document into an object-oriented database. Chapter 5 describes the implementation of the work presented in Chapter 3 and chapter 4 exposing a flexible framework for representing the object-oriented database into XML format. Chapter 6 discusses a new mapping between object-oriented schema described in object definition language (ODL) and XML Schema. This work also discusses the two way transformation of data between ODL and

XML, and then the steps for the implementation of this work. Chapter 7 includes the conclusion and highlights the future research directions.

Chapter 2

Background and Related Work

This chapter provides an overview of various XML technologies, object databases and object oriented concepts, and the mapping between XML and different databases. As stated in Section 2.3.1, little research of mapping between XML and object oriented databases exists. This is mainly due to the emergence of object-relational databases. As relational databases are still the dominating model and because object relational model is benefiting from the matured relational model, researchers argue that object relational model is the most appropriate solution for handling object oriented data and for storing XML data. Therefore, research communities concentrate on mapping object-relational with XML. We argue that object oriented databases are more appropriate because of their sufficient level of maturity [ODMG 3.0], robustness, and because of their complementary model to XML structure; object oriented model and XML structure share more features and therefore the semantic gap is limited. Because of the similarity between the object oriented and XML structures, storing XML documents into object oriented databases preserve the XML structure. Also, there is a steadily increase in using these databases, so a good amount of data is stored in object oriented databases and this data is required to be exposed into XML format. Another benefit that can be gained from storing XML into OODB is the ability to apply an object oriented query language on stored XML data. Furthermore, as object relational model is based on relational structure, storing objects will suffer from the impedance mismatch problem (Section 2.2.5).

Section 2.1 explains the structure of an XML document, a description of some XML schema languages, types of XML databases, and different XML query languages. Section 2.2 discusses the object databases, the history of their development, definition of object and class, and the object oriented concepts such as inheritance, encapsulation, polymorphism, persistence objects and so on. Mapping between XML and different types of databases is explained in Section 2.3. This section discusses the mapping between XML and relational, object-relational, object-oriented databases, and the indirect mapping between XML and databases.

2.1 XML Review

2.1.1 Introduction

XML (eXtensible Markup Language) [76] has been defined by the World Wide Web Consortium (W3C). XML is emerging as the standard universal format – as a text-based markup language - for data exchange and presentation over the internet. As XML is a text-based markup language, it is accepted as a convenient choice for transferring data cross-platform over the internet. Being a markup language, the roots of XML may go back to the 60s as the history of the markup languages started from the late 60's when IBM created the GML (General Markup Language) to facilitate text management in large information systems. Then in 1978 the American National Standards Institute (ANSI) created its first version of SGML [35] (Standard General Markup Language), and the first standard release of SGML was published in 1986. SGML is a meta language; a language that can be used to define other languages. Example for this is HTML [36, 59].

HTML was originally designed in 1990 by Tim Berners-Lee at CERN (European Laboratory for Particle Physics) to allow "physics nerds" to communicate with each other. It was first released in December 1990 within CERN, and then became available for the public in 1991. The HTML markup language is composed of fixed set of predefined tags that show how to manipulate and display information rather than how to identify the content of the data. It directs the browsers on how to display paragraphs, headers, colours, fonts and so on.HTML does not allow users to define their own tags; the set of tags allowed in HTML is rather predefined. Further, HTML is static; that means it does not show dynamic information. For example, suppose some static HTML web pages include information about weather, what could happen if the temperature or wind speed is continuously changing? It is required to re-edit the page and update the changes. It is not practical to make changes in the web page constantly without the use of automated tools. For instance, a scripting language such as Java server Pages (JSP) can be used to grab the weather information from an updated database and apply it to the weather web pages. This script should be loaded and executed on a web application server. Also AJAX (Asynchronous JavaScript and XML) can be used for accessing dynamic data through HTML. AJAX is a new approach of using programming standards that have been populated by Google. It is based on making HTTP (Hyper Text Transfer Protocol) request from JavaScript to the database server so as to get the requested information. It does not need to reload the web page as other web programming scripting languages do. It still needs to call a script loaded on the internet application server side that can extract the requested data.

HTML is also characterized by having little semantic structure. It tells how to display information, but it does not tell what the information is about. The search for a piece of data in HTML is very difficult because there is no indication for the meaning of the data, so the search results could be misleading. When people started using HTML, they quickly started realising the limitations of this markup language.

Due those shortcomings of HTML, researchers, designers, and developers started thinking of using a markup language that can overcome the above mentioned problems. Because SGML is a very difficult markup language to use, the semi-structured and self-describing eXtensible Markup Language (XML) [76] has emerged. As XML is simpler than SGML and has very powerful features comparing to HTML, it becomes a dominant markup language.

2.1.2 XML Primer

XML [76] can be seen as a simple, flexible meta language derived from SGML. XML is originally designed to meet the challenges of large-scale electronic publishing. It is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere. It uses a tree-like document representation model. XML provides a simple, yet extensible and platform independent syntax that has made it the preferred option for data interchanges on the Internet. This due to its flexibility and extendibility, XML is becoming the format of choice for data across many fields. All XML documents must abide to grammars and constraints defined in their XML schema languages such as Document Type Definition (DTD) or XML Schema. The schema languages are very flexible that allows users to define an arbitrary complex document in a tree-like structure. As has been already mentioned,

XML is a root-like tree representation of data.

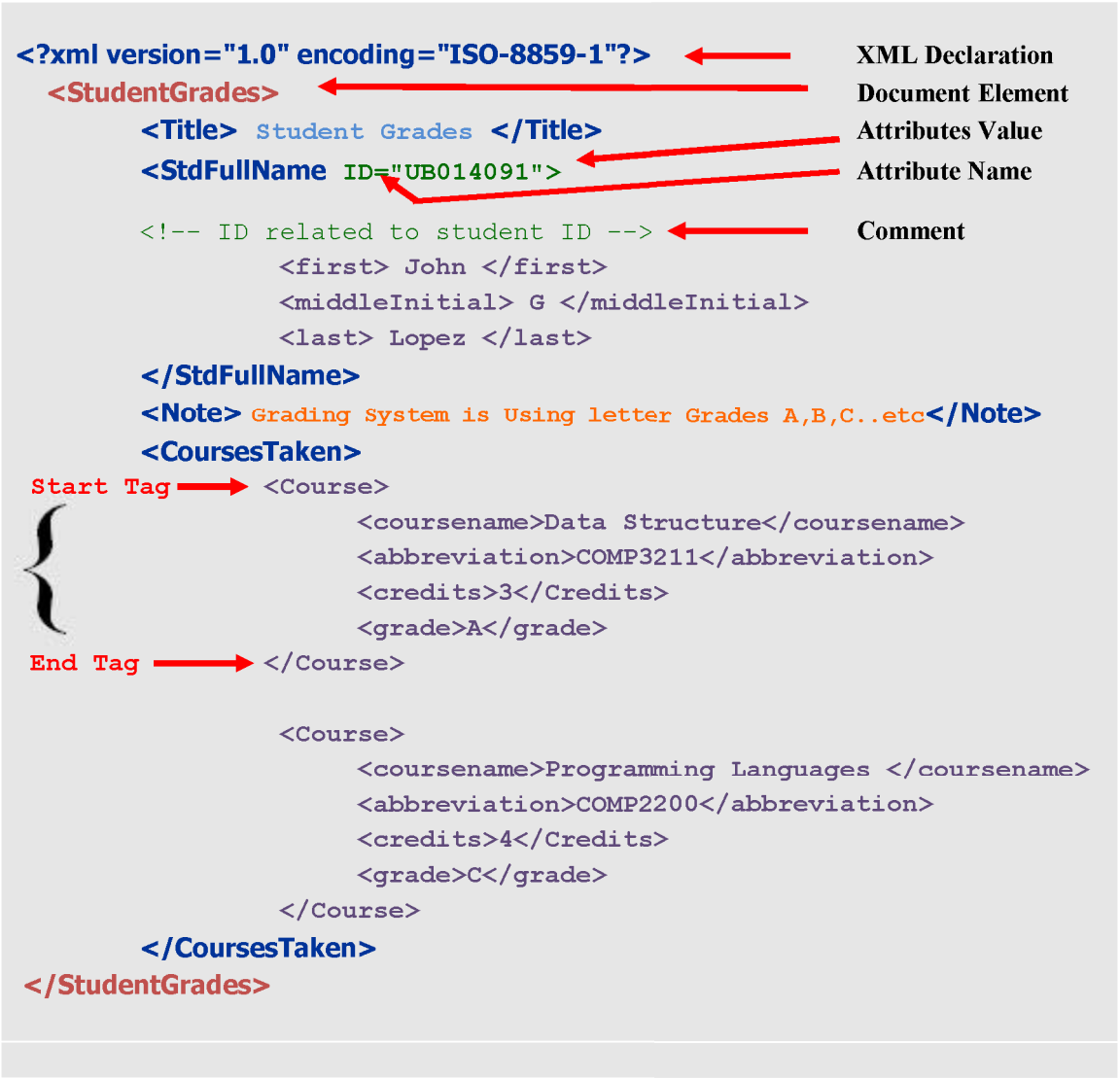


Figure 2.1: XML Document Example

In Figure 2.1, an XML document is representing student grades. An XML document is composed of a few constructs. The most used constructs are **element, attributes, and values.** Constructs are defined in tags. In the example given in Figure 2.1, the XML document starts with XML declaration **<?xml version="1.0" encoding="ISO-8859-1"?>**. Declaration includes XML version and the encoding

used. The next line of the document describes the root element StudentGrades. Title, StdFullName, Note, and CoursesTaken are the subelements of the root element StudentGrades. StudentGrades, StdFullName, and CoursesTaken elements are called complexType elements because they contain other subelements. ID is an attribute of StdFullName element and "UB014091" is the value of ID attribute. XML attributes normally provide additional information about element. This information is not necessarily being part of the data. In the next XML segment, "gender" is an attribute name for the employee element and "Male" is the value for this attribute. Attribute values should be quoted by either single or double quotes.

```
<employee gender = "Male">
      <firstname>Taher</firstname>
      <lastname>Naser</employe e>
</lastname>
```

The next XML segment is equivalent to the above one; that means they provide the same information. The latter segment uses an element instead of attribute to describe gender.

```
<employee>
     <gender>Male</gender>
     <firstname>Taher</firstname>
     <lastname>Naser</lastname>
</employee>
```

In this thesis, element approach instead of attribute approach will be used. This is useful because if the attribute is stored into the database, it is required either to specify a class that holds all attributes and a link from the attribute to the original class or to define an instance variable into the original class to hold the attribute value and a flag to tell that this is an attribute for the stored element. The problem is that; when this database is retrieved by another client, he may not understand such structure and may not realise that this instance is an attribute and the defined flag is not an element. So, although this approach may cause partial loss of the original

XML document structure, still it is much better to handle the attribute as element rather than retrieving incorrect XML structure.

Tags could be depicted as equivalent to a column name in a relational database table, an instance variable (attribute) in object-oriented databases or a name of a field in a conventional data file; a tag may even be used to define a table, a class, a database, a file, etc. For example this XML tag <firstname>...</firstname> can be easily identified as a column named firstname in a relational table or a field named firstname in a conventional data file.

Figure 2.2 is an example of a start and end tags of an element and Figure 2.3 represent the tree-like structure of the XML document presented in Figure 2.1.

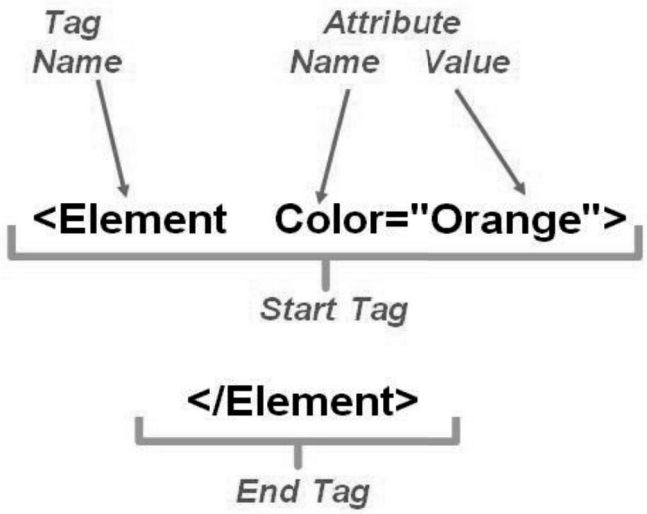


Figure 2.2: Definition Format for Start Tag, End Tag, and Attribute

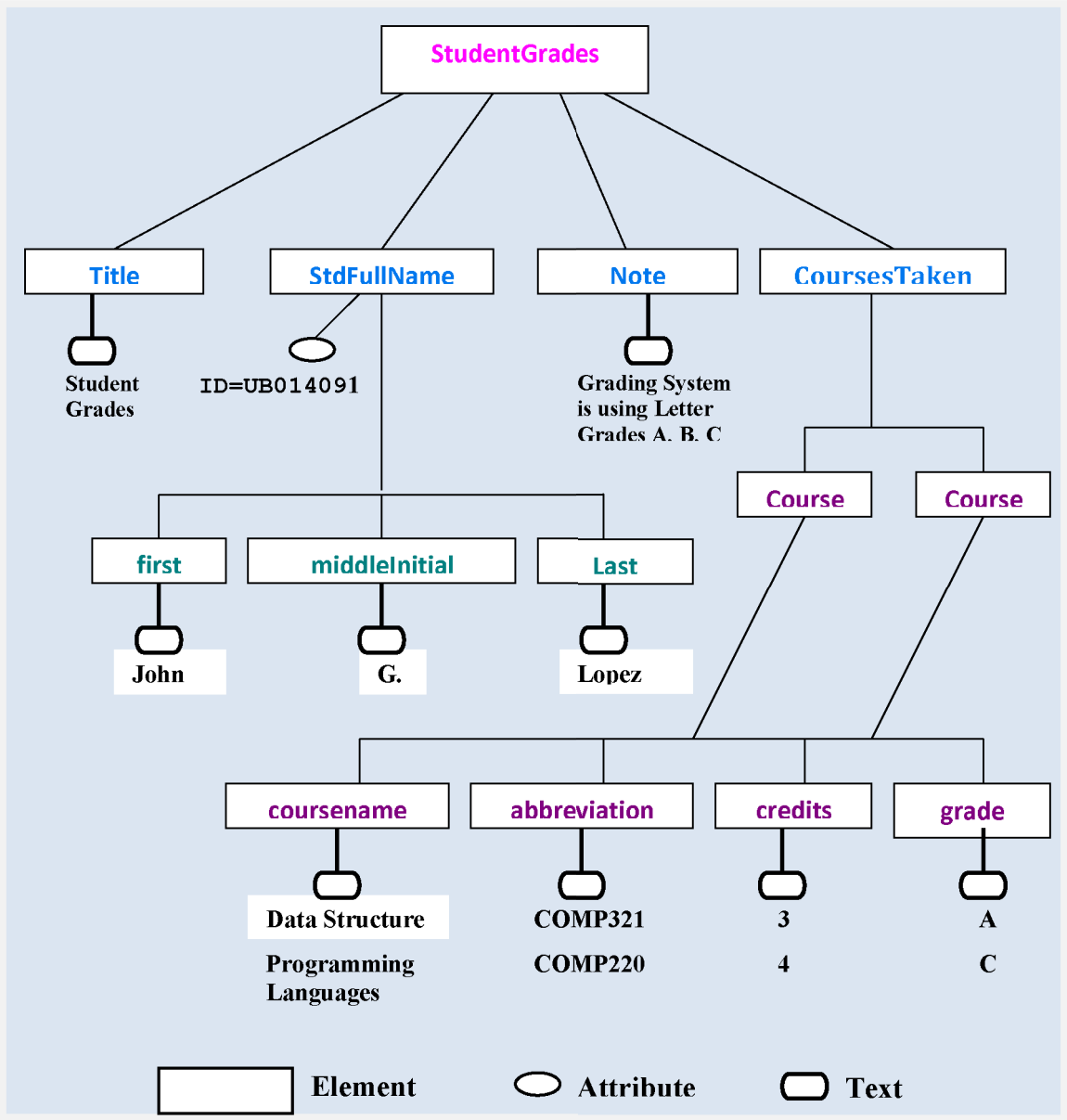


Figure 2.3: Tree Presentation for StudentGrades

XML documents should be well-formed. A well-formed document is the one that conform to XML syntax and structural rules. This implies that a XML document must begin with XML declaration, should have one root element, no unclosed tags, no overlapping tags, attribute values must be enclosed in quotes, and the tags symbols(<, >, “) should be represented by terminators when they are shown inside the document text, and so on. In other words, the document should be parsable.

The next XML segment is not a well-formed XML segment because there is an overlap between "employee" element tag and "lastname" element tag. They are nested incorrectly. The "lastname" element tag should be closed before the "employee" element tag. Also the value "Male" for the attribute "gender" should be quoted by single or double quotes.

```
<employee gender = Male>
   <firstname>Taher</firstname>
   <lastname>Naser</employee>
</lastname>
```

The next XML segment is the correct well-formed version of the previous segment

```
<employee gender = "Male">
   <firstname>Taher</firstname>
   <lastname>Naser</lastname>
</employee>
```

All modern browsers have a built-in XML parser that can be used to read and manipulate XML. Most XML parsers are based on Document Object Model (DOM). The parser reads XML into memory and converts it into an XML DOM object.

The well-formed xml document may not be a valid document. Looking into the following example, the XML document is a well-formed document, but it is confusing and meaningless. The value for attribute **unit** of element **Qty** is defined as "ml" and "gm". This is meaningless, so most of XML parsers detect such errors.

```
<Ingredient>
   <Qty unit="ml">100</Qty>
   <Qty unit="gm">50</Qty>
   <Item>Orange Juice</Item>
</Ingredient>
```

So, the valid XML document is the document that can conform with the grammar that is represented by the XML document schema. In our case, the document should

meet the XML Schema structure and rules. XML schemas are explained in details in the next Section.

2.1.3 XML Schema Languages

An XML document is typically composed of the schema and the document content (data). The schema reflects the basic constructs of the model together with the rules and semantics that control the content of the document. There are many XML schema languages recently available such as Data Type Definition (DTD), XML-Schema, and RELAX NG and so on. The following is a description for the most commonly used schema languages DTD and XML Schema.

2.1.3.1 Document Type Definition (DTD)

Document Type Definition (DTD) [8] is an XML schema language that defines the grammar and constraints of the structure of the XML document. It defines the document structure by defining the root element, subelements, element attributes, types of elements and type of attributes, the allowed values for attributes and so on. The DTD has a different structure and syntax from XML documents compared to XML Schema (Section 2.1.3.2) which itself is an XML language. The DTD can be used to specify the order and the occurrences of the elements. Figure 2.4 is a fragment DTD Schema that represents the XML document shown in Figure 2.4.

```
<!-- This is a DTD Example for the XML Document in Figure 2.4 -->
<!ELEMENT StudentGrade (StdFullName, Note?, CoursesTaken)>
<!ELEMENT StdFullName (first, middleInitial?, last)>
<!ELEMENT first last #PCDATA #REQUIRED>
<!ELEMENT middleInitial (#PCDATA)>
<!ATTLIST StdFullName ID CDATA #REQUIRED>
<!ELEMENT Note (#PCDATA)>
<!ELEMENT CoursesTaken (Course+)>
<!ELEMENT Course (coursename, abbreviation, credits, grade)>
<!ELEMENT coursename(#PCDATA) #REQUIRED >
<!ELEMENT abbreviation (#PCDATA) #REQUIRED >
<!ELEMENT credits PCDATA #REQUIRED >
```

Figure 2.4: DTD Schema Example

The main building blocks (constructs) of DTD structure are elements and attributes. Elements are defined using the keyword <!ELEMENT> while attributes are specified using the <!ATTLIST> keyword. DTD can realise a XML document as made up of elements, attributes, entities, and character data. Entities are the special characters that have a special meaning in the XML document. Start and end XML tags <, >, &, ", ' are entities defined and have a meaning for an XML document. Character data can be categorized as parsable and non-parsable data. The keyword PCDATA is used for describing the Parsable Character DATA while the keyword CDATA is used for Character DATA. While PCDATA is examined by the parser against the entities and markup tags, CDATA is not checked by the parser and as a result entities and markup tags are considered as normal text. The #REQUIRED keyword means the value of the element cannot be null.

To understand better the work of DTD, let us have a look into the next example DTD fragment.

```
<!ELEMENT StudentGrade (StdFullName, Note?, CoursesTaken+)>
<!ELEMENT StdFullName (first, middleInitial*, last)>
<!ELEMENT middleInitial (#PCDATA)>
```

In DTD, elements and subelements are declared in a given sequence. In this line of DTD segment **<!ELEMENT StudentGrade (StdFullName, Note?,**

CoursesTaken)>, a declaration for an element called “StudentGrade” that have three declared subelements named “StdFullName”, “Note”, and “CoursesTaken is defined. The three subelements “StdFullName”, “Note”, and “CoursesTaken” should appear in corresponding XML document in the same order. “StdFullName” is the first, “Note” if exist (as denoted by the “?” symbol which means it is optional) is the second, and “CoursesTaken” is the last. DTD has the facility to define elements occurrences. The indicator (+) means one or many occurrences of the element, the indicator (?) means 0 or 1 occurrences, the indicator (*) means 0 or many occurrences, and the element between two brackets (element name) means one occurrence only. In the following examples, “StdFullName” should occur only once, “Note” should occur 0 or 1 time, “CoursesTaken” should occurs once or more, and middleInitial* should occur 0 or more times.

2.1.3.2 XML Schema (XSD)

XML Schema is a W3C recommendation [75]. It contains rules, constraints and semantics that describe the structure of the content model of XML documents. It supports many data types and allows of creating custom data types. It has been widely accepted by the software industry.

The XML Schema fragment shown in Figure 2.5 below specifies PersonClass complexType element. The elements SSN, name, age, sex, spouse and nation are called simple elements because they do not contain any other elements. PersonClass element is called a complex element because it include simple elements (subelements). There is a basic difference between complex types which allow

elements in their content and may carry attributes, and simple types that cannot have element content and cannot carry attributes.

```
<xsd:schemaxmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:complexType name="PersonClass">
        <xsd:sequence>
                <xsd:element name="SSN" type="xsd:int"/>
                <xsd:element name="name" type="xsd:string"/>
                <xsd:element name="age" type="xsd:int"/>
                <xsd:element name="sex" type="xsd:char"/>
                <xsd:element name="spouse" type="PersonClass"/>
                <xsd:element name="nation" type="CountryClass"/>
        </xsd:sequence>
        <xsd:key name="PersonPK">
                <xsd:selector xpath="PersonClass"/>
                <xsd:field xpath="SSN"/>
        </xsd:key>
        <xsd:keyref name="spouseRef" refer="PersonPK">
                <xsd:selector xpath="PersonClass/>
                <xsd:field xpath="spouse"/>
        </xsd:keyref>
        <xsd:keyref name="nationRef" refer="CountryPK">
                <xsd:selector xpath="PersonClass/>
                <xsd:field xpath="nation"/>
        </xsd:keyref>
</xsd:complexType>
</xsd:schema>
```

Figure 2.5: XML Schema Example

As described, XML Schema is a complex description language that offers strong control over the document structure and text data. Here are some of the main features of XML Schema.

- **Predefined and User-defined Types.**
  It provides predefined simple types but also allows the user to define Customised complex types derived from the simple types. The next XML fragment shows the employee complexType element which is composed of firstname and lastname simple type elements.

```
<employee>
        <firstname>July</firstname>
        <lastname>Moore</lastname>
</employee>
```

  Element "employee" complexType can be used as a new Customised type in the XML document.

- **Parsable**
  It is written in XML syntax, so any XML parser can parse it. Also there is no need to learn another language.

- **Type inheritance**
  It allows the reuse of any part of an XML Schema in other XML Schema definitions. In addition, it allows us to reuse reference multiple schemas in an XML document.

- **Namespaces**
  Namespaces are used to provide unique names for elements and attributes in the XML documents. XML Schema has very good control over namespaces and their definitions.

- **Uniqueness Constraints, Keys, and References (Keyrefs).**
  It is possible to define uniqueness constraints and key/foreign key declarations for elements and attributes using "key" and "keyref" constructors.

- **Occurrence Constraints**
  XML Schema allows us to define the cardinality of an element (number of

possible occurrences). It can be defined by using two XML Schema build in occurrence constrains attributes (also called indicators) named : “minOccurs” and “maxOccurs”. The default value for “minOccurs” is 1.The appearance of an element is optional when the value of the minOccurs attribute in its declaration is 0 and the appearance is required when it has the value 1 or more. The “maxOccurs” attribute takes 1 as a default value and should take a value 1 or greater. The maximum number of times an element may appear is determined by the value of a maxOccurs attribute in its declaration. It should be greater or equal to minOccurs attribute. If both minOccurs and maxOccurs are omitted, the element must appear exactly once. The “unbounded” value indicates that there is no maximum number of occurrences.

The features that have been discussed above make XML Schema the most appropriate option to be mapped with relational, object-relational, or object-oriented databases schemas and data. Detailed specification for the XML Schema language can be found in [75].

2.1.4 XML Databases

An XML database is a collection of XML documents that can be stored and manipulated. XML documents are defined in two main categories: *data-centric* (structured documents) and *document-centric (*unstructured and often semi-structured documents). Data-centric documents are those that have a highly regular data structure. Examples for data-centric are patients records, flight schedules, stock shares, and invoice payments. The physical structure of data-centric documents, such

as the order of sibling elements is often unimportant. Sibling elements are the elements that share the same parent element. Object-oriented databases do not have the concept of sequence among their properties, and similarly relational databases do not have this concept in their column properties. So when storing document-centric XML data with mixed content into an object-oriented or relational database, it is very significant to store sibling order of elements in a data repository either within the database or within the elements.

Document-centric [14] (unstructured or semi-structured documents) are those that have irregular structure, such as in user's manuals, advertisements, newspapers, and marketing brochures. They are characterized by irregular structure and mixed content; and their physical structure is important.

There are two main categories of XML databases, namely XML Enabled databases explained in Section 2.1.4.1 and Native XML databases explained in Section 2.1.4.2. Oracle Database 11g with the extension Oracle XML DB [56] could be counted as a third type of XML databases. It is a hybrid database that can manage both XML-centric and relational data. XML-centric means both the data-centric and the document-centric XML documents. For more information refer to Section 2.1.4.3.

2.1.4.1 XML Enabled databases

XML Enabled databases are constructed with an underlying traditional database, such as relational, object-relational, or object-oriented databases that have a capability to store and retrieve XML schemas and documents. XML Enabled databases are managing, storing and querying XML documents in different ways. Firstly, an extension is built for databases such as Oracle Database 11g [57], and

IBM DB2 [24]. Secondly, middleware software is built over the database, as in SilkRoute [31] and XPERANTO [19]. Thirdly, a mediator that can manage XML documents and that has been built over different databases, as in Agora [47]. XML Enabled databases are mostly appropriate for the data-centric documents and applications.

Databases with extension are responsible for storing and retrieving XML Schemas and documents to and from the underlying database. During the storage of the XML document, XML schema (DTD or XML Schema) is mapped into the database schema and then the data is transferred accordingly. Data is shredded into many parts and stored in relations. During the retrieval of an XML document, the database schema is mapped into XML Schema and the XML document is extracted and composed accordingly. XML Enabled databases with extensions enable users to execute XML-based queries against the underlying database. XML requests (queries) are translated into SQL queries which are executed by the DBMS. The tabular query results are converted and tagged into XML, and then the XML document is generated and returned back to user.

The middleware software is providing query-able XML views over the underlying relational or object relational database. Users can then query and (re)structure XML data using an XML query language, without having to deal with the underlying SQL tables and SQL query language. XPERANTO [19] and SilkRoute [31] are well-known examples for middleware projects.

2.1.4.2 Native XML databases

Native XML databases [38, 49, 56, 69] are databases that have an internal model or structure that have the capability to store any XML Schema and document. The term *native* means that documents are stored in data structures that have been designed for only XML data, not in the form of relations as in traditional DBMSs. Like other databases, they support features like transactions, security, crash recovery, query languages, and so on. Native XML databases preserve document identity, document order, processing instructions, and comments. Also, this type of XML database could have better integration as all pieces of XML documents are stored in a known structure that do not need any mapping to other structures. So, native XML databases are mostly useful for storing document-centric documents. They support XML query languages that allow the execution of complex queries such as this example: "Give me all documents that have the bold word "**food**" with "red" colour located in the second section of the fourth paragraph". It is not an easy task to perform such query in XML Enabled databases since it is necessary to fragment and store the XML document in many relational tables. Tables will be used to store the content, font style (bold), colour, location (section, paragraph) and other attributes. When trying to execute this query against a relational database system, it is first required to translate the XML query into SQL and then to execute the generated SQL against the underlying RDBMS. This will require many joins across tables, many indexes and reads lookups to retrieve the data where that will be very costly.

A typical example of a Native XML database is the Tamino XML server [67, 69], Developed by Software AG in 1999. Tamino has a complete native XML database management system that is supporting transactions, security, multi user access,

logging, crash recovery etc. It supports XQuery [9] and full text retrieval functionality, handles document-centric documents regardless of their structure, and has the facility to store other types of data such as MS Word documents, HTML pages, images, sound files, etc. It has metadata repository that include XML schemas such as DTDs and XML Schema, and Style Sheets.

Lore [49] is a database management system designed to handle semi-structured data. Lore's data model, Object Exchange Model (OEM) is self-describing and nested object model. It supports a query language called Lorel, multiple indexing techniques, a cost-based query optimizer, logging, and recovery. One of Lore's features is the DataGuide, which is a structural summary of the managed database. DataGuide are used to explore the structure of the database and to formulate queries. They are also used to store statistics and guide query optimization.

Timber [38] is a native XML database developed at the University of Michigan [37] and build on top of SHORE (Scalable Heterogeneous Object Repository) [17]. It has been developed based on the relational model considering that many components of a standard database system can be reused with no change (such as Transaction Management Facilities). Some components have been modified to accommodate the new data model and query language. For example Timber extends XQuery (see 2.1.5.3) with functions for inserting, updating or deleting nodes, attributes or their contents. Cost estimation and query optimization techniques have also been developed. Timber is based upon set-at-a-time processing that can manipulate sets of ordered, labelled trees and natively stores XML. Each operator in the algebra would take one or more sets of trees as input and produce a set of trees as output.

2.1.4.3 Hybrid XML databases

Oracle Database 11g developed an extension called Oracle XML DB [56] that supports managing XML documents. Oracle Database, with Oracle XML DB provides a hybrid database for managing both XML-centric and relational data. Oracle XML DB is built on the core components of XMLType abstraction for storing, querying, accessing, and manipulating XML data. XMLType is an abstract data type that allows different storage models to best fit XML data. Oracle XML DB supports three main storage models for XML data; structured, unstructured, and binary storage models. It is recommended that data-centric (structured) XML documents can be stored in object-relational storage model with B-Tree indexes while semi structured XML documents can be stored in hybrid object-relational and CLOB (Character Large Object) with B-Tree, XML, and full text indexes. Also, document-centric structured XML documents can be stored in binary XML or CLOB storage with XML and full text indexes, while document-centric unstructured XML documents can best fit binary XML or CLOB storage model with XML and full text indexes.

2.1.5 XML Query Languages

Since XML emerged, many XML query languages have been proposed. Among those, three XML query languages (namely Lorel, XML-QL, and XQuery) have been chosen to represent the spectrum of most XML query languages. Lorel [4] is the known XML query language designed for the semi-structured data. XML-QL [25] is the first query language in XML syntax. XQuery [9] is the first proposal of a W3C standard query language for XML; it was accepted as the standard in February 2007.

XQuery includes the experience of the previously defined query languages and highly imitates SQL. The following is a brief review of those three query languages:

2.1.5.1 Lorel

Lorel [4] query language was originally designed for querying semi-structured data. It was then implemented as the query language of the native XML Lore database management system. It is a user friendly language in the SQL/OQL (Object Query Language) style. For wide applicability, the simple object model underlying Lorel can be viewed as an extension of the ODMG (Object Data Management Group) data model and the Lorel language as an extension of OQL.

2.1.5.2 XML-QL

XML-QL was designed at AT&T Labs; it has been developed based on other query languages (UnQL and Strudel) for semi-structured data [25]. XML-QL language extends SQL with an explicit CONSTRUCT clause for building the document resulting from the query and uses the element patterns (patterns built on top of XML syntax) to match data in an XML document. XML-QL can express queries as well as transformations, for integrating XML data from different sources [13].

2.1.5.3 XQuery

XQuery is the W3C proposal for a standard XML query language, published in February 2001, revised in June 2001 [16] and in February 2007 [9]. It is a query language designed to express queries across all kind of XML data sources, whether they are stored in XML documents or in databases and can be viewed as XML. XQuery is derived from an XML query language called Quilt [23], which in turn

inherited features from several other languages, including XPath [7], XML-QL [25], SQL, and OQL [15].

2.1.6 Other XML Technologies

2.1.6.1 DOM

The XML DOM (Document Object Model) [74] is an API that allows programs and scripts to dynamically access and update the content, structure and style of XML documents. In DOM, XML documents are represented as objects (nodes) in a tree structure; i.e. it reads the entire XML document into the memory to build the tree structure. Their content (text and attributes) can be modified or deleted, and new elements can be created. DOM defines interfaces for each different entity in an XML document (elements, attributes, etc.), and specifies methods for manipulating the structure and the content of the document. Large documents are difficult to process as it will require large amount of RAM.

2.1.6.2 SAX

Simple API for XML (SAX) [62] is an event based API used to break the structure of an XML document into a linear stream of events. A compatible SAX XML parser uses a set of callback methods when an event occurs, such as when the parser encounters the start tag and end tag of elements. It does not allow navigation and backtracking of the XML document. SAX can handle XML documents of any size as it does not need to construct a tree structure for the XML, but it place a high load on the application developers as they have to handle the events received.

2.1.6.3 XSLT

Extensible Stylesheet Language Transformation (XSLT) [42] defines a language for transforming an XML document into a different data presentation based on the designer of that stylesheet. This presentation can construct a new result tree from the XML source tree, and produce a formatted result suitable for presentation on the web.

2.1.6.4 XML Namespaces

Elements in XML documents are defined by the developers [12]. When having documents with the same element names but different content, this causes conflict when those documents are gathered in one XML application. To overcome this problem, namespaces are used. Namespaces are prefixes defined before the element name to distinguish it from any other element with the same name. The namespace declaration is defined in the start tag of the element; it has this syntax: xmlns:prefix="URI". This assures that there will be no conflict between elements names for the whole application.

2.1.6.5 XPATH

XPath [7, 18] is a path-based language designed and standardized by W3C since 1999; it forms one of the basic constructs of XQuery. It views an XML document as a tree structure consisting of nodes (element, attributes, etc.), each representing the entities of a document. XPath uses path notation for navigating through the hierarchical structure of an XML document and then addressing and returning the required parts of the data. Its syntax is simple and efficient, so it is easy to learn. Conditions can be applied within a path to extract the required information and filter

unwanted data. It is used by various XML technologies such as XSLT (XML transformation language), XQuery, and XLink (XML linking language).

2.2 Object Databases and Object Oriented Concepts

2.2.1 Introduction

Relational model is a successful structure used since the mid-1980s for developing traditional business applications. However, there are clear shortcomings when it is required to design and develop complex database applications for systems such as computer-aided design (CAD), computer-aided Software engineering (CASE), geographical information systems (GIS), and multimedia data such as audio, video, graphics, images and so on. These complex structures require a database model that can support, manage, and handle these data. As the available traditional databases do not support complex data types and query languages and operations that can apply to these complex models, object oriented databases emerged. Another reason for the creation of object oriented databases is the domination of object oriented programming languages such as C++, Java, Smalltalk in building business applications. [EN97].

An object database uses a database model that support storing, manipulating, and handling complex data represented as objects. For example complex objects could include photos, video, audio, and images. Object oriented database management systems (OODBS) can be defined as a combination of database capabilities with the combination of object-oriented programming language capabilities.

An object is the key element of understanding the object-oriented technology and their concepts. In real-world, object is a physical thing than can be seen around such as car, television, bicycle, and PC. Also conceptual things can be counted as objects such as temperature, air pressure, and human feeling. For the purpose of modeling, a teaching staff could see students, curriculum, class room, and text book as objects for a taught course. An automotive engineer could view body, engine, automatic transmission, and tires as objects for a model of a car.

An object usually has two components; state and behaviour. State is the internal value (data) of an object while behaviour is the operation (method which are similar to function in traditional programming languages) that apply to the object. For example, the state of an engine object could be (on, off) and behaviour could be (starting engine, stopping engine), the state of a car transmission could be (Drive, Reverse, Park, related speed) and behavior could be (current gear-shift lever, current speed), and the state of human being could be (name, eye colour, length, weight, hungry) and behaviour (eat, walk, sleep). Also, object data may contain relationships between this object and other objects.

Each object has a unique object identifier OID. This identifier is generated by the database system and it is neither changeable (immutable) nor can be reused or assigned to another object when the object is deleted. OIDs can be used to reference other objects. This is similar to the use of foreign keys and referential integrity in relational model.

An object may have an arbitrary complex structure in order to contain all of the necessary information that describes the object. For example a car is an object that is composed of other instances of type object such as engine, gear, and tires and some

other simple structures such as colour, shape, and model.

2.2.2 OODBs History

Object oriented concepts are formally introduced with the emergence of object oriented programing languages in the late 1960s. Simula-67 was the example for the first developed object oriented language designed by Dahl et al. [26] at the Norwegian Computing Centre. Smalltalk, developed by Alan Kay et al. [43] at the Learning Research Group at Xerox's Palo Alto Research Centre in the 1970s is considered to be the first real and pure object oriented programming language as the language structure is designed to be object oriented language; i.e. it incorporate concepts such as inheritance and message passing. This language forces the developers to use the object oriented paradigm.

In the early 1980s, many object oriented database projects were started in research and university labs. Some of these projects include IRIS at Hewlett-Packard, ODE at Bell Labs, Encore-Ob/Server at Brown University, and ORION at Microelectronics and Computer Technology Corporation (MMC). Among all, ORION [44, 45] is one of the main research projects that had started by Won Kim at MCC from which the current object oriented database Versant [71] traces its history from this project.

Starting from 1985, the term of object-oriented database system was used [5]. A number of commercial OODBs products had been developed after the mid of 1980s and during 1990s. This includes Gemtone [34] from Servo-Logic (Changed to Gemtone Systems), Grapheal from GBase, Versant [71] from Versant Corporation (was Object Sciences Corp), Objectivity/DB from Objectivity Inc. [55], and O2 from Ontologic (changed to ONTOS Inc.).

Atkinson et al. [1]. In their popular and influential paper "The Object-Oriented Database System Manifesto" defines an object oriented database management system as: it should be an object-oriented system and it should be a database management system (DMBS). They had grouped the characteristics into three categories:

- Mandatory

  An object-oriented database system must satisfy two criteria: it should be a DBMS and it should be an object-oriented system. DBMS translates into five features: persistence, secondary storage management, concurrency, recovery and an *ad hoc* query facility, and object-oriented system translates into eight features: complex objects, object identity, encapsulation, types or classes, inheritance, overriding combined with late binding, and extensibility and computational completeness. Any systems that satisfy these characteristics deserve the OODBS label.

- Optional

  Those features are not mandatory to make the system an object-oriented database system, but clearly they improve it. Those features are: Multiple inheritance,Type checking and type inferencing, Distribution, Design transactions, and Versions.

- Open

  These are the characteristics where OODBS designers and implementers can select from a number of equally acceptable solutions. No consensus has yet been reached and it is not known which of the alternatives are better. Those

characteristics are: programming paradigm, representation system, type system, and uniformity.

Although there were many object-oriented databases present and available in the market, there were no agreed-upon standard model and query languages for these databases. Many benefits can be gained out of the presence of a standard OODBS model such as portability, interoperability, and the possibility to compare different commercial products. Portability means that an application can be migrated from one product to another with minimal changes and cost. Interoperability means that the same application can access different DBMS packages in different databases such as OODMS and RDBMS packages. Comparing different products help in showing which features of the standards are supported, and this help in making proper decision when purchasing a product. The lack of standards for object-oriented database model could be one of the reasons that lead to no wide spread take up of object-oriented databases. In contrast, the standard SQL was one of the main causes for the wide-spreading of RDBMS model.

In 1991, Rick Cattle and other five major OODBS vendors form the Object Data Management Group (ODMG). The major aim of this group is seta standard object-oriented database model and specification that allow developers to write portable applications. In 1993, ODMG published their first release ODMG-93 of the standard specification for object oriented database management systems and for object-relational mappings. This version has the initial standards that include Object Definition Language (ODL) which is compatible with object management group Interface Description Language (IDL), Object Query Language (SQL-like declarative language), and C++ and Smalltalk Bindings.

The second version of ODMG standards was published in 1997. This version has many enhancements and improvements to OMDG-93 version. It includes four main components: the ODMG object model, the object definition language (ODL), the object query language (OQL), and object-oriented programming languages bindings for C++, SMALLTALK, and JAVA languages. Language bindings are the definition on how to write a portable code that manages and manipulates persistent objects. The ODMG members that represent most of the ODMS industry are supporting this specification. So these specifications become the de facto standard for OODB industry [15].

In 2001, ODMG published their last standard release ODMG 3.0. This release includes enhancements to Java bindings, improvements to object model, verification for the implementation of the standards in industry, and the specification for persistence of object-oriented programming language objects in databases [15].

The initiative started with ODMG-93 for creating a standard object query language (OQL) has been abandoned in 2001. Instead, ODMG submit ODMG Java Binding to the Java Community Process as a start for the Java Data Objects (JDO) Specification, then after that, ODMG disbanded.

The emergence of open source for ODBMS had started in 2004 with the launch of db4o database system from db4objects, Inc. In 2005, William Cook et al. [21, 22] proposed to use the object oriented programming languages to express queries using the language itself; e.g. use Java or C# to query objects data. This was the start of using Native Queries. The main benefit of native queries is ability of writing complex and precise queries is quick and easy way. In late 2005, db4o is one of the first to implement Native Queries to access object oriented data. Also, Microsoft

announced Language Integrated Query (LINQ) that allow the integration of the query capabilities with its programming languages C# and VB.NET 9.

In early 2006, the Object Management Group (OMG) started developing a new specifications based on the ODMG 3.0. They formulate the Object Database Technology Working Group (ODBTWG) that had started creating standards to incorporate advanced features in OODBS such as replication, data management (e.g., spatial indexing) and data formats (e.g., XML).

2.2.3 Object Oriented Concepts

Object orientation has a set of major characteristics. Some of these characteristics include: the object model, encapsulation, inheritance, polymorphism, classes, and persistence. Next are some details of the main object oriented concepts.

2.2.3.1 Object Model

Object Model is a database model that can determine the characteristics of the object. These characteristics include how an object can be named, how it can be identified, how the object can be related to other objects, and how the object can be stored, retrieved and manipulated. This model handles objects natively, i.e. the object can easily fit into the model regardless of its complex structure.

2.2.3.2 Class

Class is the logical construct that defines the characteristics of an object. Once the class is defined, new data types of type object is already created, so class can be considered as a template for creating objects of that class. The variables created of type class are called instance variables. For instance, in Java, a class could be

depicted as in this small code:

```
class Car {
        double    Model;
        string    Color;
        string    Type;
}
```

A new instant variable of type “Car” can be defined such as in the next line of code.

```
Car mycar = new Car();
```

2.2.3.3 Encapsulation

Encapsulation is the mechanism that put together the object operation (method and interface) with the state (data) and protects them from any external visibility. The internal structure of the encapsulated object is not visible (hidden) to the external objects and the object is only accessed by an external object is through predefined operations of the encapsulated object. This feature protects data from any misuse.

Encapsulation allow modification of the internal object operations and state without causing any disturbance for the external objects that are invoking these object operations. In other words, the external object access the encapsulated object the same way by using same operations and interfaces (name and argument) regardless of any internal complex changes that may occurs into operation and state of the object. A good example of encapsulation is a car that includes automatic transmission, gas pedal, tires and so forth. The automatic transmission is an object that encapsulates a complex structure of the gear and its relation to engine. The interface for accessing this operation is the gear-shift lever, so the driver is only using this interface and has nothing to do with the internal structure of the transmission. All automotive companies have different internal structure of the

automatic transmission. Any changes they may make in the internal structure of the transmission do not affect the way the driver is using the gear. [27, 63].

2.2.3.4 Inheritance

Inheritance is one of the cornerstones of object-oriented concepts. Simply it can be described as a process that allows an object to possess the properties and attributes of another object. This is very significant because it support the concept of hierarchical classification. If some objects are sharing the same attributes, then an object can be defined to include these attributes and then all objects can inherit this new object attributes. For example, the objects (classes) EMPLOYEE, STUDENT, and SECRETARY have same attributes such as name, ID, department, salary and so forth. A general object (class) named PERSON can be defined to include these shared attributes (name, ID, department, salary). Then PERSON object can be inherited by objects EMPLOYEE, STUDENT, and SECRETARY. Hierarchical classification can be illustrated by this example. The Orinoco Crocodile is part of crocodile classification, which in turn is part of the reptile class. Reptiles are part of the animal classification. Objects in real-world are related to each other in a hierarchical way such as animals, reptiles, and crocodiles. Animals have common behavior such as eat, sleep, move, and breathe. Reptiles have behaviors (creep, hide) that may not necessarily be present in all animals. Crocodiles have behaviors (swim, preys) that may not necessarily present in all reptiles. So the attributes of an animal class can be inherited in reptile class and the attributes of animal and reptile classes can be inherited into crocodile class.

2.2.3.5 Polymorphism

Polymorphism, also called overloading, is a generic term that means many forms. This feature makes it possible to process objects, methods, and variables in different forms depending on their data types. It makes it possible to use one generic interface for a group of related activities such as methods [63].

Java allows us to define more than one method with the same name but with different parameters declarations. This is called method overloading, which is the Java implementation of polymorphism. Next is a Java segment that shows different methods called using one interface.

```
// Example for method overloading
class poly {

   // Overload polytest for one integer parameter
   void polytest(int x) {
      System.out.println("x: " + x);
   }

   // Overload polytest for two integer parameters
   void polytest(int x, int y) {
      System.out.println("x: " + x + "   y: " + y);
   }
}
class Polyoverload {
   public static void main(String args[]) {
     poly my_par = new poly();
     // call the 2 versions of polytest()
     My_par.polytest(15);
     My_par.polytest(15,20);
   }
}
```

When executing this segment, the result will be as:

```
x: 15
x: 15    y: 20
```

This Java segment shows that the method “polytest” is defined in the class “poly” with the same name for two times; first with one integer parameter argument and the second with two integer parameter arguments.

2.2.3.6 Transient and Persistent Objects

Objects in object oriented programing languages exist only during program execution. They are allocated to the main memory like variables and once the program terminates, the objects are destroyed. This type of objects is called transient objects. In contract, object oriented databases extend the existence of the object and store it permanently to a durable storage. This object persists after the program terminates and can be retrieved, and shared by other programs. These types of objects are called persistent objects.

2.2.4 Object Oriented Database Management Systems

An object oriented database management system (OODBMS) is a database management system that supports the object data model. This model allows creating, storing, and managing objects in the database. The object model specifies the semantics that can be defined explicitly to an object database management system. For instance, the semantics of the object model could determine the characteristics of objects, how objects can be related to each other, how objects can be named, and how they can be identified, ..etc. [15]. Applying the former definition and characteristics of object-oriented databases on a created database using set of Java classes, the result will clearly show that this created database is classified as an object-oriented database.

There are two ways to manage objects; first is to use object database management systems (ODBMS) that store objects directly, and second is to use the approach of Object-to-Database Mappings (ODMs) that map and store the object in relational or other types of database presentation. The two types are called object data management systems (ODMSs) [15].

Object oriented database model is designed so that the object oriented programming languages can easily and smoothly handle and integrate the data because they share the same semantics; i.e. the data of objects can be read and stored from/to the database directly without any mapping.

Due to the popularity of relational model and the need to present objects (complex data), RDBMS manufacturers added extra features and characteristics to support the object model, so object-relational databases emerged. OODBMS was expected to replace relational databases, but due the popularity of relational model (significant amount of applications running under RDBMS), a very high cost of migration, and the emergence of object relational databases; all this contributed to the lack of wide-spread use of object oriented databases.

2.2.5 Benefits of Using Object Oriented Databases

OODBS is used when it is required to have a business need for high performance on complex data presentation [6]. Complex structure can be stored in the database without any translation or mapping as the database model supports storing complex objects. Performance can be gained because when reading the complex structure from disk, there is no need to translate or map the data as the development tools such as Java or C++ can natively read any arbitrary complex format. Less code is required to write applications because the development tools such as Java or C++ won't have to translate into an intermediate language (sub-language|) such as SQL via JDBC, or ODBC. Also using OODBS will avoid the object-relational impedance mismatch problem. Impedance mismatch occurs when there is a clash between two incompatible models; i.e. object model structure is totally different than relational model, so when trying to store an object into object-relational database, object will

be scattered over many relations so as fit in the model. This causes a loss of links and ties between the object in real-world and its database representation. Figure 2.6 shows the impedance mismatch case [58].

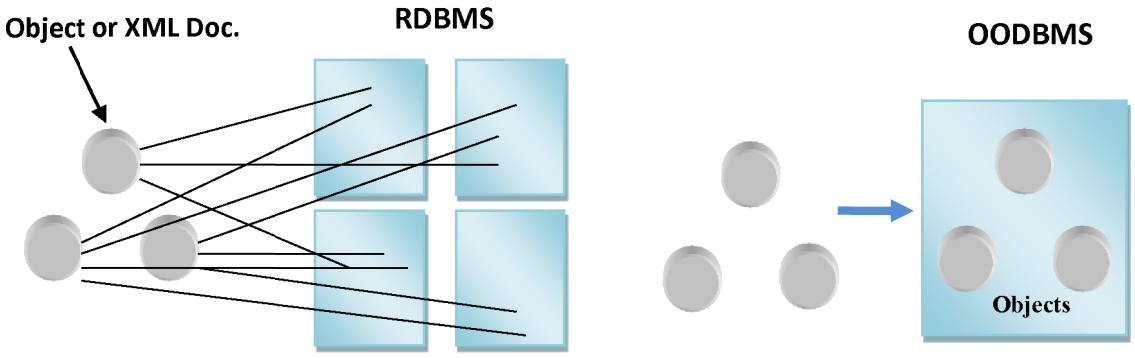


Figure 2.6: Impedance Mismatch when Storing Object or XML Document in RDBMS

2.3 Related Work

In this section we discuss mapping XML to traditional databases in general. We also discuss some of the work conducted on mapping XML to object-oriented, object-relational, relational, and some other ways of mapping such as EER (Extended Entity Relationship model) to XML, and using UML(Unified Modelling Language) to design XML

2.3.1 Mapping XML to Traditional Databases

As XML becomes widely accepted, the need for an efficient storage and management of XML documents triggered the attention of researchers. There are two dominating approaches for storing XML documents [29]. Some researchers consider storing XML documents in databases that can have an internal model (structure)

designed specially to accommodate XML documents. Those databases are called native XML databases [38, 49, 56, 69]. However, most of the e-business applications are stored and managed in relational, object-relational or object-oriented databases that have very powerful set of management services that can handle and manage the data in an efficient manner (including ad-hoc queries, transactions, security, referential integrity, concurrency control, scalability, etc.). Further, benefiting from the highly optimized relational query processors, researchers considered storing, retrieving and managing XML documents in relational [11, 14, 31, 30, 28, 64, 66, 73, 77], object relational [14,19, 57, 61, 68]and object-oriented databases [2, 20, 41, 54, 51, 52]; these are called XML Enabled databases.

2.3.2 Mapping XML to Object-Oriented Databases

Little work has been performed on mapping between XML Schema and object-oriented databases schemas.

The work conducted by Tae-Sun Chung et al. [20] has proposed a technique for extracting object-oriented database schemas from XML DTDs using inheritance. From the DTD declaration of an element, they create subelements that inherit the common attributes of the parent element. Those subelements are mapped as child classes of the parent element. Inlining technique of relational databases have been used. Inlining is to store as many descendants of an element as possible into a single class so as to resolve the fragmentation problem. Figure 2.7 is an example for the "univ" XML fragment.

```
<univ>
  <student>
     <id>244</id>
     <name>
         <first>Bob</first>
         <last>Jone</last>
     </name>
     <nat>British</nat>
  </student>
  <student>
    <id>245</id>
    <name>Robert Delpy</name>
    <dob>20-june-1960</dob>
  </student>
</univ>
```

Figure 2.7: XML Segment Inlining Example

This fragment can be presented into an object-oriented schema using a simple inlining technique. Using the inlining approach, all or most of "univ" descendants are stored in one class of the object-oriented schema. Part of object-oriented schema could be depicted as:

univ (univ.id, univ.name.first, univ.name.last, univ.nat, univ.name, univ.dob),

and the univ descendent elements data are stored in one class as

```
univ = {< 244, "Bob", "Jone", "British", null, null>
        <245,null, null, null, "Robert Jone", "20-june-1960">
       }
```

The work presented in [41] focuses on the ability to wrap an XML Schema definition in an object-oriented virtual database mediator system to help solving the integration problems between XML documents and other applications that are not using XML. They develop a tool called *Amos II XML Schema import tool* (AXSI)for importing XML Schema definitions and corresponding XML document into a mediator system called *Active Mediators for Information Integration*(Amos II) [60]. A given set of translation rules from XML Schema to object schema definitions govern the

translations that take place in the wrapper. A developed wrapper is an interface that translates a data source's data model to a common data model known by the mediator. Object-oriented query language of the mediator called AmosQLis used to query the imported data.

Ahmad, et al. [2] developed a system called *"Transformation of data between XML and object databases"* (TransODB). This system is to resolve the problems facing CERN (European Organization for Nuclear Research) in replicating and transferring data between different data repositories in a heterogeneous operating system environment. *TransODB* is composed of two modules. First is *TransODB Database to XML* module that extracts the object-oriented schema and corresponding object-oriented data from the object-oriented database and convert them into an XML Schema and an XML document. The second is *TransODB XML to Database Conversion* module, which is composed of two main sub modules, *DB Schema Builder* and *Object Builder*. *DB Schema Builder* rebuilds the object-oriented schema from the extracted XML Schema generated by first module *TransODB Database to XML*, and *Object Builder* recreate the objects data from the extracted XML document. The architecture is not well explained and the internal mechanism is not discussed.

Toth and Valenta [70] investigated possibilities of reuse already known techniques from object and object-oriented processing in XML-native database systems. This addresses the mapping of the contents of an existing XML into object-oriented database.

Our work in [54, 51, 53] discuss the two-way mapping between object-oriented database and XML. Details are discussed in chapters 3 and 4. The work in [52]

explains the implementation for converting object-oriented database into XML, and the work in [40] describes the mapping between ODL and XML. Details can be found in chapter 5 and 6 respectively.

2.3.3 Mapping XML to Object-Relational Databases

Object-Relational databases have more powerful features over relational databases; this attracted researchers to use those databases for storing and managing XML documents.

Runapongsa et al. [61] mapped XML documents to tables in an Object-Relational Database Management System (ORDBMS) using XML DTD schema. An important part of this mapping is assigning a fragment of an XML document to a new XML data type.

XPERANTO [19] (XML Publishing of Entities, Relationships, ANd Typed Objects) project from IBM Almaden Research Centre is a middleware layer that supports publishing object-relational database as XML data. It provides a virtual XML view over an object-relational database and supports XML queries against this view. It allows users to query and (re)structure the contents of the database as XML data, without worrying about the underlying SQL tables and without having to learn SQL query language. XPERANTO translates requested XML queries into SQL, submits SQL queries to the underlying database system, receives SQL execution (by RDBMS) results, and then tags the results for constructing XML documents.

R. Bourret [14] discussed the object-relational mapping (called it object-based mapping) between XML DTD schema and object-relational schema. This model the XML document as a tree of objects based on the data in the document. XML DTD

schema is mapped to an object schema, and then the object schema is mapped to the relational database schema. Classes in object schema are mapped to tables in relational schema, and complexType elements are identified in the mapped tables by foreign keys.

Oracle 11g XML DB [56] provides a way to store and retrieve data-centric structured XML Schema by mapping it to object-relational schema. Also for storing the data-centric unstructured XML Schema, a hybrid of object-relational and Character Large OBject [CLOB] is a good option [56]. B-tree indexes are used for querying data-centric structured XML documents, while XML and Full Text Indexes are used for querying the data-centric unstructured XML documents.

2.3.4 Mapping XML to Relational Databases

Since XML emerged, there has been a significant amount of research work on using the relational database as a mean for storing and managing XML documents.

Bohannon et al. [11] provide a cost-based approach that models the target application with an XML Schema, XML data statistics, and an XQuery workload. It explores a space of possible XML-to-relational mappings and automatically finds the best and efficient relational configuration for a target XML application.

Yushikawa et al. [77] developed an approach for storage and retrieval of XML documents on top of any off-the-shelf relational databases. No extension is required for the relational database. It decomposed XML documents into nodes based on document tree structure and stored them in relational tables according to the node

type, with path information from the root to each node even without any information about the DTDs.

Florescu et al. [30] developed a way to store XML documents named “the edge approach”. Edges of an XML documents tree are stored in a relational database as relational tuples. XML documents are represented as an ordered and labelled directed graph. Each XML element is represented by a node in the graph, and the node is labelled with an object identifier (OID).

Schmidt et al. [64] proposed a data model for storing and retrieving XML documents based on binary fragmentation of the document tree. XML document is decomposed into small units. This caused all associations such as parent-child relationship, attributes and the sibling order to be described, stored, and queried.

Shanmugasundaram et *al.* [66] converts XML Schema DTD to relational schema and then the XML documents to relational tuples, translates semi-structured queries over XML documents to SQL queries over tables, and converts back the results to XML. They used Basic, Shared, and Hybrid inlining techniques to resolve the fragmentation of XML documents by storing as many descendants of an element as possible into a single relation.

XML Extender [24] serves as a repository for XML documents as well as their Document Type Definitions (DTDs), and also generates XML documents from existing data stored in relational databases. It is used to define the mapping of relational tables and columns to DTD. XSLT and XPath syntax are used to specify the transformation and the location path.

SilkRoute [33] is described as a tool for viewing and querying relational data in XML. It serves as middle-ware between a relational database and an application accessing that data over the Internet. In SilkRoute, XML views of relational databases are defined using a relational to XML transformation language called RXL. XML-QL queries are issued against views. The query composer combines queries and views together, and the combined RXL queries are then translated into corresponding SQL queries. In order to use SilkRoute, it is necessary to learn the new language RXL.

VXE-R [46] is an engine for transforming a relational schema into equivalent XML Schema. They issue XML queries against the XML Schema. The engine is composed of three components. A translator of relational schema into XML Schema, a query translator for the XQuery queries against the XML Schema into SQL queries against the underlying relational database, and the last is the generator of the SQL query result into an XML document.

2.3.5 Other XML Mapping

Some other XML mappings are not done directly to databases. They use intermediate models such as EER and UML, then they map this model to a database.

Mani et al [50] proposed the conversion of EER-to-XML. The idea of this conversion is to generate XGrammar from a given XML model, then convert XGrammar to EER model, or vice versa.

Fong et al. [28] applies the Indirect Schema Translation Method that translates relational schema into an Extended Entity Relationship model (EER). EER model is mapped into XML Schema Definition Language (XSD) Graph, and then the XSD

graph is mapped into XML Schema. In [32], Fong et al. derive the EER model from relational database and store it in an intermediate repository. Then they map EER entities to DTD elements and relationships to DTD Hrefs attribute. Finally they propose to construct the XML instance (document) by mapping the data from relational database using the generated XML DTD Schema and SAX APIs and the XSLT processor.

The work done by Booch et al. [10] describes a graphical notation in UML (Unified Modelling Language) for designing XML schemas. UML is a standard object-oriented design language. They map all elements and data types in XML Schema to classes annotated with stereotypes that reflect the semantics of the related XML Schema concept.

Wang et al. [72] developed an approach to convert legacy relational databases into XML databases through reverse engineering. They had addressed this issue by first applying the reverse engineering approach by extracting the ER (Entity Relationship) model from a legacy relational database, then convert the ER to XML Schema. The proposed approach is capable of reflecting the relational schema flexibility into XML Schema by considering the mapping of binary and nary relationships. Also Wang et al. [73] had developed a system, named COCALEREX (Conversion of Catalog-based and Legacy Relational databases to XML), which handles the reengineering of relational databases into XML. It can let users view XML of the underlying relational data.

2.4 Conclusion

In this chapter, Section 2.1 includes a review of XML and relevant XML technologies. Different examples are discussed to clarify XML document structure, XML Schema languages, XML databases, XML query languages, and XML technologies such as DOM, XPATH, SAX and others. Section 2.2 discusses the object-oriented databases and object-oriented concepts. The history of OODB is explored and the definition of the basic components such as objects, object model, and the classes are discussed. In addition, the object oriented concepts such as encapsulation, inheritance, object persistence, and polymorphism are explained. Besides, the benefit of using OODBs is discussed. Section 2.3 discusses in details different work carried out by researchers, techniques used for mapping between XML and object-oriented, object-relational, and relational databases management systems. The work discussed in Section 2.3.2 is about mapping XML into object oriented, and in Section 2.3.3 is for mapping XML into object relational databases, and in Section 2.3.4 is about mapping XML into relational databases, while Section 2.3.5 discusses the mapping of XML into other ways such as EER, UML and others.

Storing an XML document into relational database requires fragmenting the document into small components of data to fit in relational structure. Also, referential integrity constraints are to be defined for referencing different parts of the document. When retrieving the XML document, it is required to collect different components of that stored document from different relations. The difference between the XML

model and the relational model is called impedance mismatch.

Mapping XML to object relational database is performed in two steps. First, XML is mapped into objects, and second the objects are mapped into relations. When retrieving a stored XML document, it requires collecting information from different tables for composing the requested document. Impedance mismatch is here inevitable as well.

The work performed in this thesis for mapping between XML and object-oriented databases is more attractive and more natural process because there is clear overlap between the XML Schema and the object-oriented Schema paradigm (refer to Section 3.1), so the impedance mismatch is avoided. XML Schema complex types are mapped to classes and elements are mapped to attributes. Also in this work, XML Schema is used while most the previous discussed work used DTDs. XML Schema is more powerful, flexible and accepted widely more than other schema languages such as DTD. XML Schema supports many rich build-in data types and has the flexibility to compose new complex data types. This makes it a more appropriate tool for creating different data types during the mapping process. The schema meta-data is by itself written in XML syntax, so the same tools used to process XML documents can be used to read, parse, and manage the XML Schema. In addition, it supports namespaces that helps in avoiding the name conflict of elements and attributes of different XML documents within an application. Detailed specification for the XML Schema language can be found in [75].

As the mapping from XML to object-oriented databases is concerned, the work described in [20] generates an object-oriented database schema from DTDs, stores it into the object-oriented database and processes XML queries; it mainly concentrates

on representing the semi-structural part of XML data by inheritance. However, the work detailed in [52] differentiates between inheritance and nesting, which is a more natural approach for handling object-oriented databases. In addition, XML Schema is used rather than DTD Schema.

The work presented in [2] does not include details about the mechanism for the extraction of XML Schema and corresponding documents from an object-oriented database neither the creation of the object schema and corresponding data from the generated XML Schema.

The work in [41] is a one way mapping of XML Schema into a virtual object-oriented database mediator. It creates Java classes for the mapped XML Schema in the Amos II system, and then use an object-oriented query language called AmosQL against the virtual object-oriented database. This work differs from the work presented in this thesis as it does not have the facility to generate an XML Schema and corresponding document from an object-oriented database data. Also a new query language should be learned.

In contrast, the novelty of our work comparing to others is as follows:

- XML Schema is mapped into object-oriented database schema. It is done by mapping the XML Schema into an intermediate object graph (OG) and then the OG is mapped into object-oriented database schema. This work differentiates between nesting and inheritance and handles them accordingly.
- XML Schema and corresponding document are constructed from an object-oriented database (a reverse process of the former process).

- It uses object-oriented database as the underlying database.
- This work uses XML Schema as it is more flexible comparing to other XML Schemas such as DTD.
- As the mapping between OODB and XML is natural, the impedance mismatch has been avoided.

Chapter 3

Transforming Object-Oriented Database into XML

3.1 Introduction

In this chapter we will discuss in details an approach of mapping a generic OODB schema and corresponding data into XML schema and corresponding XML document using a directed object graph (OG) as a mean for describing this process. Both nested and flat XML schemas are handled. Further, inheritance and nesting attributes of classes are recognized and handled.

When starting this research, we were facing different challenges that can be enumerated as: a) a lack of research on mapping between object-oriented model and XML Schema. Some work is available about mapping between object relational and DTD schema. b) it is required to define an object model components that are shared in most of object oriented databases. c) XML Schema has a complex structure, so it is required to define the core components that should be used in the mapping process. c) to investigate for a smooth and efficient way for incorporating the object graph into the mapping process. d) As the process will handle the nesting and flat schema structures, one of the challenges is to find a way to incorporate and differentiate between the inheritance and nesting.

Extensible Markup Language (XML) [76] has become the dominant means for representing, exchanging and accessing data over the Internet. As many applications are dependent on XML, there is an interest in storing XML data in databases. There is a considerable amount of research that has been performed into managing and

storing XML documents into relational and object-relational databases [11, 14, 19, 33, 61, 66, 73]. However little attention and work have been performed on storing and managing XML documents into object-oriented databases [2, 20, 41]. The mapping from XML into an object-oriented database is more attractive than the relational or object-relational alternatives as there is more overlap between XML Schema and the object-oriented paradigm. Consequently, it is more efficient to store and manage XML documents using object-oriented databases. Bourret [14] make a comparison between relational and object-relational and XML models. The next example show the similarity of structure between object-relational database and XML comparing to relational database, Figure 3.1 shows an example of an XML document fragment that represents student grades. Figure 3.2 shows an object-oriented structure presentation for the XML fragment in Figure 3.1. StdGrades complexType element in Figure 3.1 is represented by the StdGrades class in Figure 3.2, where the Courses variable is pointing to the Course class. In contrast, Figure 3.3 presents the relational structure presentation for the same XML fragment. This shows that the XML structure (Figure 3.1) is more similar to object-oriented structure (Figure 3.2) than the relational structure (Figure 3.3).

```
<StdGrades>
      <StdID>1234</StdID>
      <Name>Sheala Norm</Name>
      <Course>
         <YrSem>08FL</YrSem>
         <Code>COMP3333</Code>
         <Grade>B+</Grade>
         <Credits>3</Credits>
      </Course>
      <Course>
         <YrSem>09SP</YrSem>
         <Code>COMP4322</Code>
         <Grade>A</Grade>
         <Credits>4</Credits>
      </Course>
</StdGrades
```

Can easily mapped to class

Figure 3.1: XML Fragment for StdGrades Complex Type

```
class StdGrades {
        StdID = 25332;
        Name  = "Sheala Norm";
        Courses = {pointers to Course classes};
  }
```

```
class Course {
   YrSem = "09SP";
   Code = "COMP4322";
   Grade = "A";
   Credits = 4;
}
```

```
class Course {
  YrSem = "08FL";
  Code = "COMP3333";
  Grade = "B+";
  Credits = 3;
}
```

Figure 3.2: Object-Oriented Presentation for XML Fragment in Figure 3.1

StdGrades

StdID	Name
25332	Sheala Norm
...	.....
...	.....

Courses

StdID	YrSem	Code	Grade	Credits
25332	08FL	COMP3333	B+	3
25332	09SP	COMP4322	A	4
.....	....	.....	...	...

Figure 3.3: Relational Presentation for XML Fragment in Figure 3.1

Since the business data currently stored and maintained in object-oriented database management systems is increasing steadily, this encourages to propose a novel approach for the transformation of an existing object-oriented database schema into XML [54] and to store an XML document into object-oriented database [51]. The major motivation to carry out this study is the fact that it is necessary to facilitate platform independent exchange of the content of object-oriented databases. There are more common features between the object-oriented model and XML and thus the mapping from object-oriented databases into XML should be more natural and smooth. To achieve the mapping, an object graph (OG) is derived based on the characteristics of the object-oriented schema; it simply summarizes and includes all nesting and inheritance links, which are the basics of the object-oriented model. Then, the inheritance is simulated in terms of nesting to get a simulated object graph. This way, everything in a simulated object graph can be directly represented in XML

format. Finally, we handle the mapping of the data from the object-oriented database into corresponding XML document(s).

3.2 Related Work

In the literature and to the best of our knowledge, there are few approaches for transforming object-oriented data into XML Schema. Little work performed on mapping XML Schema with object-oriented databases schemas is discussed in Section 2.3.2. The work in [54] is discussed in this chapter while the work performed in [51] is discussed in details in chapter 4. Chapter 5 include the work performed in [52].

Figure 3.4 represents the current available ways for publishing XML documents from an object-oriented database. First, it has to transform the object-oriented database schema and related data into relational database schema and corresponding data. Second, it uses any available tools for publishing the mapped relational data as an XML format. The two step process should preserve as much information as possible during the transformation, but unfortunately this is not guaranteed.

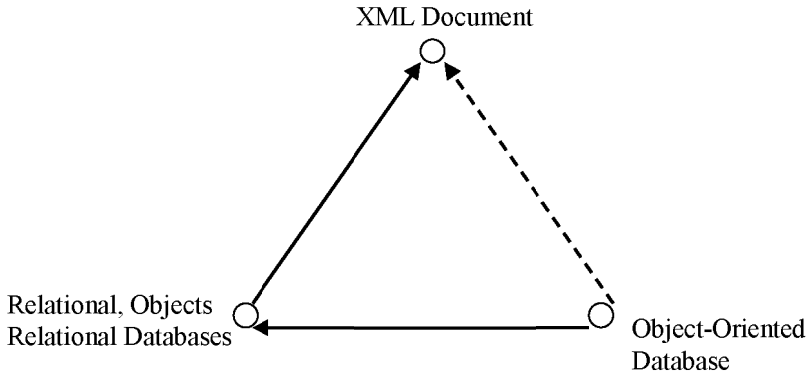


Figure 3.4: Object-Oriented to XML Transformation

What we tried to do in this research is to have direct transformation from object-oriented database to XML document. Work in this chapter is represented in the dotted side of the triangle in Figure 3.4.

3.3 Object-Oriented to XML Transformation Process

The main objective of this research [54] is to develop a system that takes a given object-oriented database as input, and produces a corresponding XML document. These are the main steps that will be followed to achieve this goal:

- Investigate the characteristics of the object-oriented database and derive a summary for the object-oriented schema.
- Derive an object graph (OG) from the investigated object-oriented schema.
- Investigate flat and nested XML Schemas.
- Transform the OG into flat or nested XML Schema.
- Generate XML document.

The next Section 3.3.1 will discuss in detail the mentioned steps above.

3.3.1 Object-Oriented Database Characteristics

3.3.1.1 The Basic Terminology and Definitions

An Object-Oriented database schema is a description of database objects. The Object Oriented Database Management System (OODBMS) schema defines what objects are stored within the database. Within an OODBMS, the class construct is normally the main component used to define the database schema. Object-oriented modeling is

based on the concept of a class. A class defines the data values stored by, and the functionality associated with, an object of that class [48]. An object is often referred to as an instance of a class. Refer to Section 2.2 for more details.

This work is mainly interested in class characteristics as present in Definition 3.1 and illustrated in Example 3.1, given next.

Definition 3.1 (Class)

A class can be defined as a tuple ($C_p(c)$, $C_b(c)$, $L_{attributes}(c)$, $L_{behavior}(c)$, $L_{instances(}c)$, OIDG), where c is a class identifier, $C_p(c)$ is a list of direct superclasses of class c, $C_b(c)$ is a set of direct subclasses of class c, $L_{attributes}(c)$ is the set of additional attributes locally defined in class c, $L_{behavior}(c)$ is the set of additional methods added to the definition of class c, $L_{instances}(c)$ is the set of object identifiers of objects added locally to class c, and OIDG is object identifier generator that holds the identifier to be granted to the next object to be added to $L_{instances}(c)$.

Inheritance makes it possible to utilize the attributes, instances, and methods defined in the superclasses of the class. The precedence for super classes is followed as from left to right and from bottom to up. For example, if class A has two superclasses B and C, then the left side class B is superseding class C. Also if class B has a superclass D and class D has a superclass E, then class D is superseding class E and so on.

Every attribute in a class has a domain. Attribute domains can be either primitive like integer, string or non-primitive domain that are built from primitives.

As illustrated in class Definition 3.1 (Class), class can be composed of those components:

1. All attributes defined locally in the class plus all attributes inherited from the superclasses of the class.
2. All Instances defined locally in the class plus all instances inherited from the super classes of the class.
3. All behaviors defined locally in the class plus all behaviors inherited from the superclasses of the class.
4. All subclasses instances that may understand any of the class behavior.

Example 3.1 (Classes)

Next is an object-oriented database schema named "UNIVERSITY". It is composed of classes Person, Country, Student, Staff, ResearchAssistant, Course, Department, and Secretary. Detailed characteristics of each class are shown below.

Person:

C_p(Person) =[] C_b(Person) ={Student, Staff, Secretary}

$L_{attributes}$(Person) = {SSN:integer; name:string; age:integer; sex:character; spouse:Person; nation:Country}

$L_{behavior}$(Person) = {SSN(); SSN(i); name(); name(t); age(); age(i); sex(); sex(i); spouse(); spouse(p); nation(); nation(c)}

Country: C_p(Country) =[] C_b(Country) ={}

$L_{attributes}$(Country) = {Name:string; area:integer; population:integer}

$L_{behavior}$(Country) = {Name();Name(t); area(); area(i); population(); population(i)}

Student:

C_p(Student) =[Person]C_b(Student) ={ResearchAssistant}

$L_{attributes}$ (Student) = {StudentID:integer; gpa:real; student in:Department; Takes:{(course:Course; grade:string)}}

$L_{behavior}$ (Student) = {StudentID(); StudentID(i); gpa(); gpa(i); student_in(); student_in(d);Takes();Takes(t)}

Staff:

C_p(Staff) =[Person] C_b(Staff) ={ResearchAssistant}

$L_{attributes}$ (Staff) = {StaffID:integer; salary:integer; works_in:Department}

$L_{behavior}$ (Staff) = {StaffID(); StaffID(i); salary(); salary(i);works_in(); works_in(d)}

ResearchAssistant:

C_p (ResearchAssistant) =[Student,Staff]

C_b (ResearchAssistant) ={}

$L_{attributes}$ (ResearchAssistant) = {}

$L_{behavior}$ (ResearchAssistant) = {}

Course:

C_p (Course) =[] C_b(Course) ={}

$L_{attributes}$ (Course) = {Code:integer; title:string; credits:integer;Prerequisite:{Course}}

$L_{behavior}$ (Course) = {Code();Code(i); title(); title(t);credits(); credits(i); Prerequisite(); Prerequisite(c)}

Department:

C_p (Department) =[] C_b(Department) ={}

$L_{attributes}$ (Department) = {Name:string; head:Staff}

$L_{behavior}$ (Department) = {Name();Name(t); head(); head(t)}

Secretary:

C_p (Secretary) = [Person] C_b(Secretary) = {}

$L_{attributes}$ (Secretary) = {words_mimute:integer; works in:Department}

$L_{behavior}$(Secretary) = {words_mimute();words_mimute(i); works in(); works in(d)}

3.3.1.2 Object-Oriented Schema Information

In the object-oriented schema shown in Example 3.1, the analysis is based on the domain information summarized in Table 3.1 ObjectAttributes (a) and (b). Table 3.1 ObjectAttributes includes information about all attributes in the object-oriented

schema. For each attribute, it shows its name, class, and the domain. Attributes with primitive domains are placed in ObjectAttributes (a) while attributes with non-primitive domains are placed ObjectAttributes (b). Each domain of "tuple type" is assigned a short name that consists of the letter “T” suffixed with a consecutive non-decreasing number, starting with 1. For instance, as shown in the fourth row in Table 3.1 (b), the short name T1 has been assigned to the domain of the attribute Takes from $L_{instances}$(Student). This way, it becomes not very hard to identify attributes that appear within a tuple domain as illustrated in the last row of Table 3.1 ObjectAttributes (a) and (b).

Class Name	Attribute Name	Domain
Person	ssn	integer
Person	name	string
Person	age	integer
Person	sex	integer
Country	name	string
Country	area	integer
Country	population	integer
Student	studentID	integer
Student	gpa	real
Staff	satffID	integer
Staff	salary	integer
Course	code	integer
Course	title	string
Course	credits	ineger
Department	name	string
Secretery	word_minute	integer
T1	grade	string

(a)

Class Name	Attribute Name	Domain
Person	spouse	Person
Person	nation	Country
Student	student_in	Department
Student	Takes	T1
Staff	work_in	Department
Course	prerequisite	Course
Department	head	Staff
Secretery	work_in	Department
T1	course	course

(b)

Table 3.1: ObjectAttributes

3.3.2 The Object Graph (OG)

An Object Graph (OG) is a structure that present the non-primitive attributes as nodes and the directed links between them. Nodes are the Vertices and links are the Edges of the graph (V,E). We use the information present in Table 3.1 ObjectAttributes (b) and the inheritance information defined in Example 3.1 to construct the OG. This includes all possible relationships between the classes present in the given object-oriented schema. Nodes in the OG are the schema classes and representatives of tuple type domains. Two nodes are connected by a link to show the inheritance or a nesting relationship between them. Nodes and links are represented by small rectangles and directed arrows, respectively. Inheritance link is assigned score 0, for example; Person class is a superclass of Student class, so a link between Person and Student has the score 0. Nesting links are assigned the score 1. In Person class there is a non-primitive attribute (nation) of type Country, so Country class is nested inside Person class and therefore a link of score 1 is created between Person and Country. A link is assigned the score 2 if it is connecting a node that represents a tuple domain with the class that is referenced to. To illustrate this, refer to attribute Takes in $L_{attributes}$(Student) in Example 3.1 and to the corresponding link connecting the two nodes T1 and Student in Figure 3.8. More formal details related to OG are included in Definition 3.2, given next.

Definition 3.2 (Object Graph)

Every object-oriented schema has a corresponding OG graph(V,E) such that,

1. For every class c in the object-oriented schema there is a corresponding node c in V,

2. For all classes c_1 and c_2, such that $c_2 \epsilon C_p(c_1)$, an edge(c_1, c_2, 0) is added to E

3. For every class c

For every attribute $a \epsilon L_{attributes}(c)$, such that a has a non-primitive domain,

If domain of a involves a class, say c', then an edge (c, c', 1) is added to E

Else if domain of a involves a tuple T_i,(I ≥1) then

A node T_i is added to V and an edge($_{Ti}$, c, 2) is added to E

For every class c'' that appears as a domain in tuple T_i, an edge(T_i,, c'', 1) is added to E.

Definition 3.2 can be illustrated as follows:

- Step 1 stated that "*For every class c in the object-oriented schema there is a corresponding node c in V* ". This means that for every class in the OODB schema, a node represents a class is created in the OG, so a node is created for class Person and another node is created for the class Country and so forth. Figure 3.5 depicts this step.

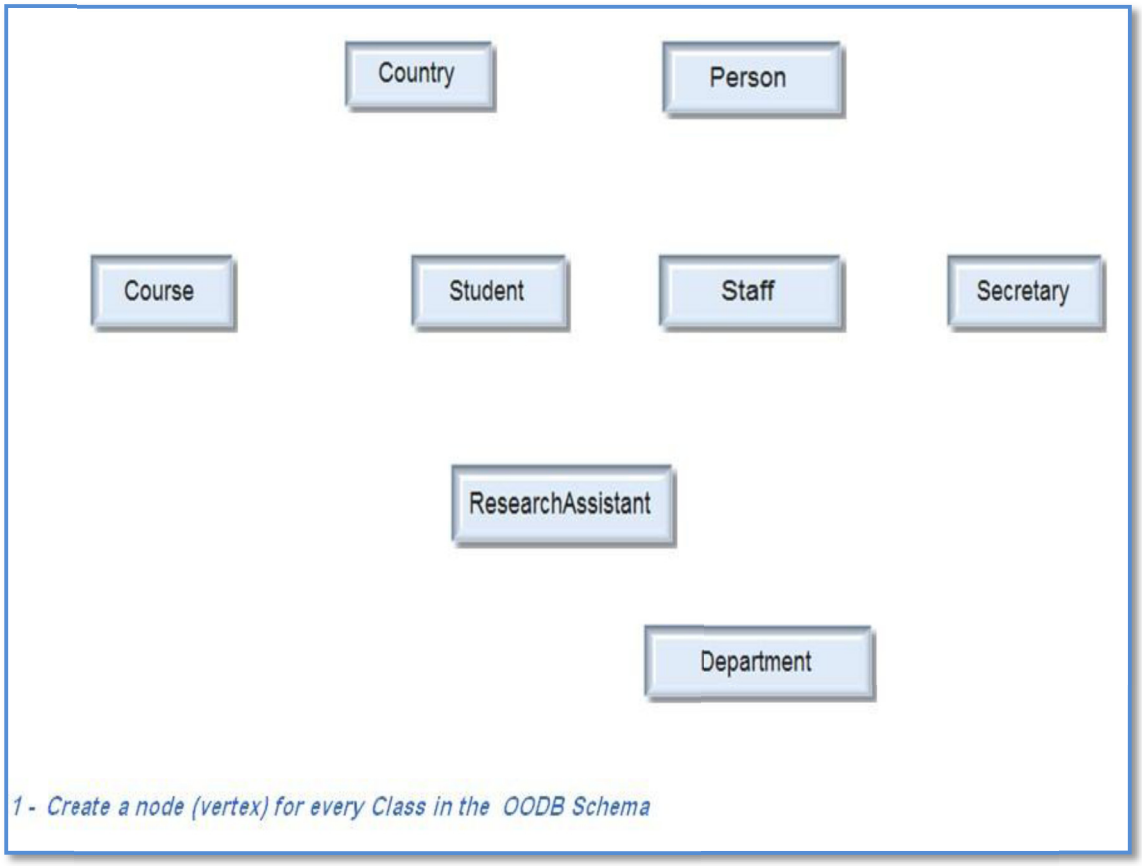


Figure 3.5: Object Oriented Schema Classes Represented as Nodes in OG

- Step 2 stated that "*For all classes c_1 and c_2, such that $c_2 \in C_p(c_1)$, an edge(c_1, c_2, 0) is added to E*". This means that for each class in OO schema, a directed link (edge) is created with score 0 from the class to all its superclasses. For example class Person is a superclass for class Staff, so a directed link connects Staff node to Person node with the head of arrow toward the superclass. Figure 3.6 illustrates this.

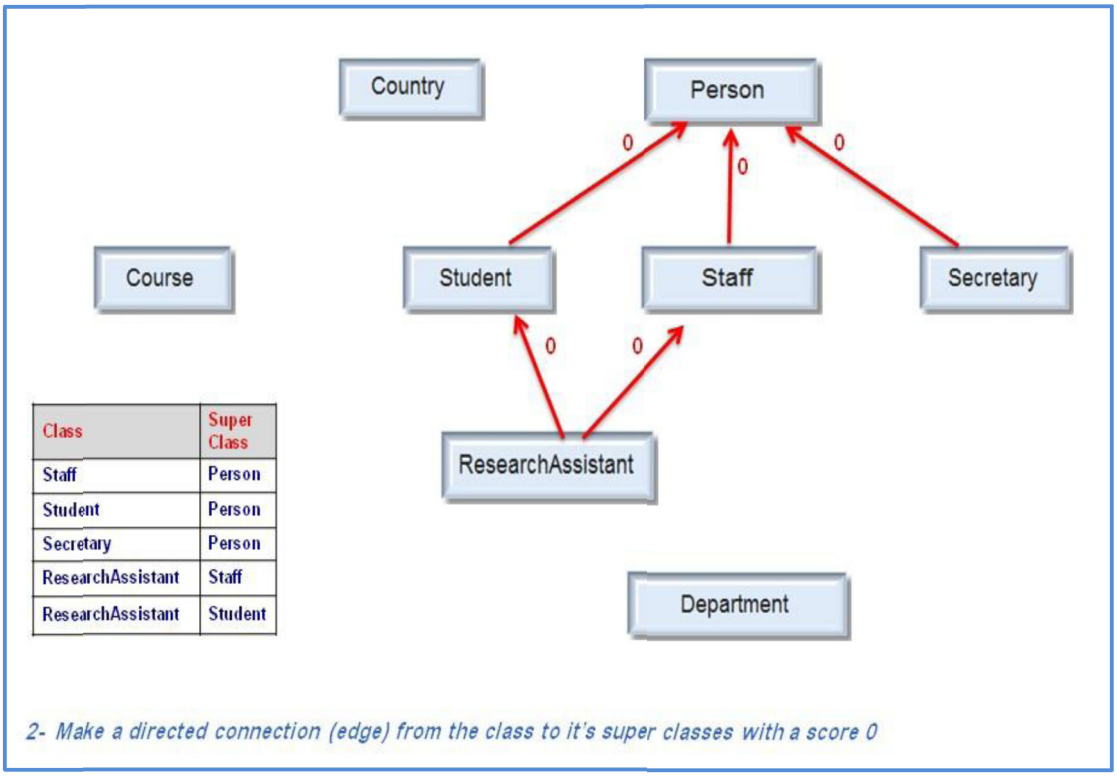


Figure 3.6: Direct Connections from the Class to its Superclasses

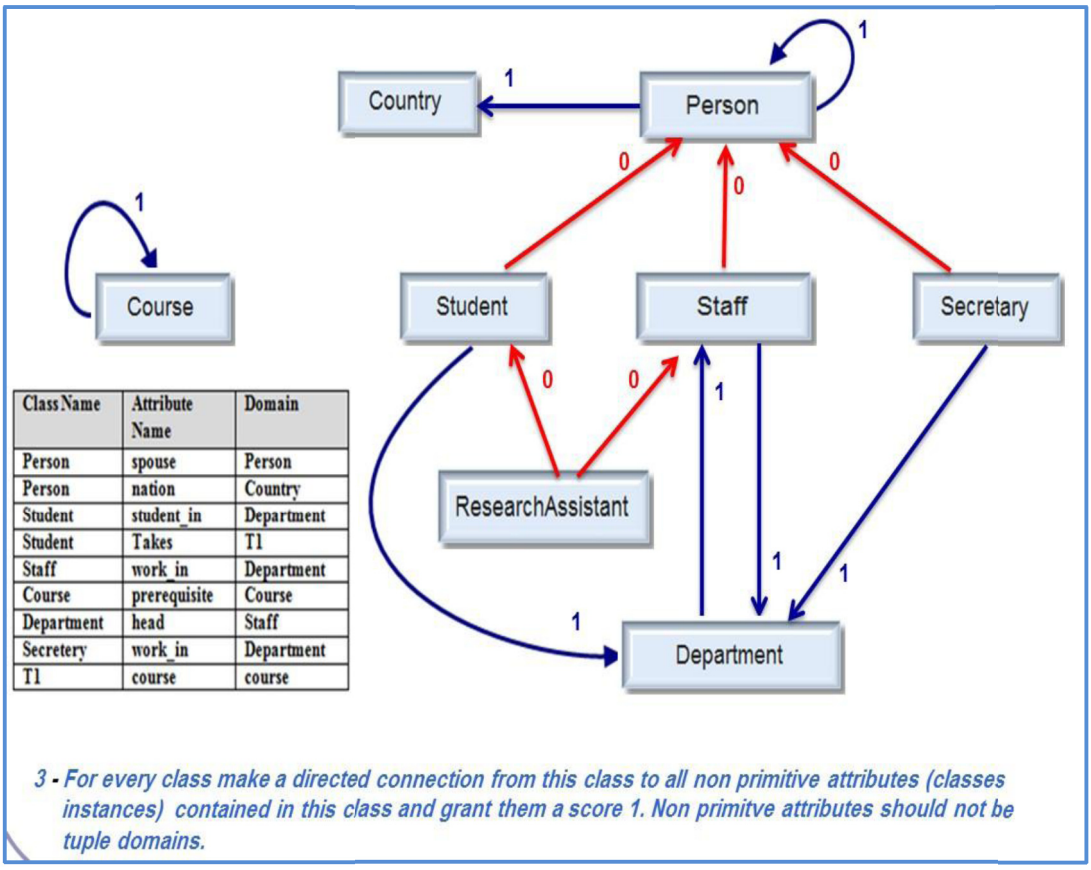
- Step 3 stated that:

  "*For every class c*

  For every attribute a ϵ $L_{attributes}$(c), such that a has a non-primitive domain,

  If domain of a involves a class, say c', then an edge (c, c', 1) is added to E".

This means that for every class, make a directed link from the class to its non-primitive attributes (classes instances) and grant them a score 1. Non-primitive attributes of the class should not be tuple domains. For instance, Department is a non-primitive attribute of class Secretary, so a directed link is created from Secretary to Department with the head of arrow directing toward Department. Figure 3.7 illustrates this.



Class Name	Attribute Name	Domain
Person	spouse	Person
Person	nation	Country
Student	student_in	Department
Student	Takes	T1
Staff	work_in	Department
Course	prerequisite	Course
Department	head	Staff
Secretery	work_in	Department
T1	course	course

3 - For every class make a directed connection from this class to all non primitive attributes (classes instances) contained in this class and grant them a score 1. Non primitve attributes should not be tuple domains.

Figure 3.7: Direct Connections from the Class to all its Non-Primitive Attributes

- Step 4 stated that:

  "if domain of a involves a tuple Ti,(i ≥1) then

  A node Ti is added to V and an edge(Ti, c, 2) is added to E

  For every class c'' that appears as a domain in tuple Ti, an edge(Ti,, c'',1) is added to E."

This is interpreted as; for every class that has a tuple domain attribute:

a. Create a node in the OG with a name Ti (i is sequence starting from 1).

b. Make a directed link from Ti to the class that contains it with a score 2.

c. Make a directed link from Ti to all non-primitive domains contained into this tuple domain with a score 1.

After the completion of step 4, the ultimate Figure 3.8 is the OG that is derived from the information presented in Table 3.1 ObjectAttributes (b) and the inheritance information in the C_p lists in Example 3.1.

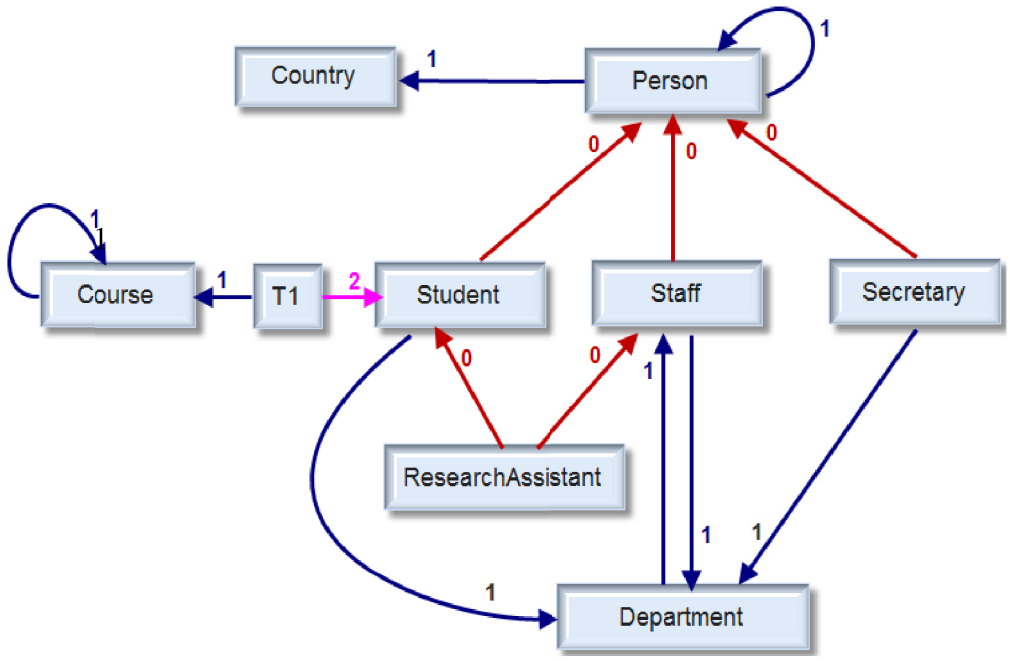


Figure 3.8: Object Graph for Object-Oriented Schema in Example 3.1

3.3.3 Flat and Nested XML Schema Types

In XML Schema, there are two ways to represent a relationship between two parts of an XML document:

- Nested XML; The complexType elements nested in another complexType elements.
- Flat XML; This type of structure is called flat XML Schema and flat XML document. It specifies “key" and “keyref" constraints in the structure of the XML Schema. Those constraints behave in the same as keys and referential integrity constraints in relational model.

The next sections explicate the nested and flat XML Schemas and corresponding document,

3.3.3.1 Nested XML Schema and Document Structure

Nested complexType definitions in an XML Schema define relationships between two elements. If we specify nested complex types to create a relationship between two parts of an XML document, this will create a nested XML document structure. For example, the following XML Schema fragment shows that the “Student” element is nested under the “SUPERVISOR” element. All students having the same supervisor are grouped as a unit. Although a nested XML Schema reflects better the natural structure and linkage between elements, is easy to read, and may reduce search time, it would occupy more space because of redundant data. Figure 3.9 is an example of nested XML Schema for SUPERVISOR_Class and Student_Class elements.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
xmlns:oodb="http://scim.brad.ac.uk/xml">
<xsd:complexType name="SUPERVISOR_Class">
      <xsd:sequence>
            <xsd:element name="SUPERVISOR_Object"
            type="oodb:SUPERVISOR_Object" maxOccurs="unbounded”/>
      </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SUPERVISOR_Object">
      <xsd:sequence>
            <xsd:element name="StaffID"type="xsd:int”/>
            <xsd:element name="name"type="xsd:string”/>
            <xsd:element name="advisee"
      type="oodb:Student_Class”/>
      </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Student_Class">
      <xsd:sequence>
      <xsd:element name="Student_Object" type="oodb:Student_Object"
            maxOccurs="unbounded”/>
      </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Student_Object">
      <xsd:sequence>
            <xsd:element name="StdID" type="xsd:int”/>
            <xsd:element name="name" type="xsd:string”/>
            <xsd:element name="Course" type="xsd:string”/>
            <xsd:element name="Grade” type="xsd:string”/>
            <xsd:element name="SSN” type="xsd:string”/>
      </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Figure 3.9: Nested XML Schema for SUPERVISOR and STUDENT Classes

Figure 3.10 represents a nested XML document fragment based on the XML Schema in Figure 3.9 example.

```
<oodb:SUPERVISOR_Class>
   <oodb:SUPERVISOR_Object>
      <oodb:SSN>1</oodb:SSN>
      <oodb:name>JOHN</oodb:name>
      <oodb:Student_Class>
         <oodb:Student_Object>
            <oodb:StaffID>2444</oodb:StdID>
            <oodb:name>Sonia Bird</oodb:name>
            <oodb:Course>COMP3000</oodb:Course>
            <oodb:Grade>B</oodb:Grade>
            <oodb:AdvSSN>1</oodb:AdvSSN>
         </oodb:Student_Object>
         <oodb:Student_Object>
            <oodb:StdID>2323</oodb:StdID>
            <oodb:name>David Hunter/oodb:name>
            <oodb:Course>COMP1200</oodb:Course>
            <oodb:Grade>A</oodb:Grade>
            <oodb:AdvSSN>1</oodb:AdvSSN>
         </oodb:Student_Object>
         <oodb:Student_Object>
            <oodb:StdID>1122</oodb:StdID>
            <oodb:name>Sayed Noor</oodb:name>
            <oodb:Course>MATH3211</oodb:Course>
            <oodb:Grade>C+</oodb:Grade>
            <oodb:AdvSSN>1</oodb:AdvSSN>
         </oodb:Student_Object>
      </oodb:Student_Class>
   </oodb:SUPERVISOR_Object>

   <oodb:SUPERVISOR_Object>
. . . . . .
. . . . . .
   </oodb:SUPERVISOR_Object>
</oodb:SUPERVISOR_Class>
```

Figure 3.10: A Fragment for Nested XML Document

3.3.3.2 Flat XML Schema and Document Structure

Flat type definitions in an XML Schema use “key” and “keyref” constructs to create a relationship between two parts of the XML document. Flat XML documents take less space than nested documents, but the search time on a flat XML document is

more than search time of the query raised on nesting XML document because of the join-like operations. To illustrate this let us look into this example. Suppose we have a flat XML document that include two complex types, the first one represents Student information and the second represents student GRADES. If it is required to generate a document that includes a student with his grades, then the search engine either will make a full document scan or use the keys and keyrefs constraints which exist to join the Student and GRADES parts of the document so as to retrieve the information. In contrast to nesting, all student GRADES data are clustered under the same Student data, so search is faster because there is no need for a full document scan and for using constraints joins to get the data.

The next XML fragment illustrates the presentation of a flat XML Schema. Figure 3.11 is an example of Flat XML Schema for Person and Country Classes defined in Example 3.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
xmlns:oodb="http://scim.brad.ac.uk/xml">

<xsd:complexType name="Person_Class">
      <xsd:sequence>
            <xsd:element name="Person_Object"
                  type="oodb:Person_Object" maxOccurs="unbounded"/>
      </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Person_Object">
      <xsd:sequence>
            <xsd:element name="SSN"type="xsd:int”/>
            <xsd:element name="name"type="xsd:string"/>
            <xsd:element name="age"type="xsd:int”/>
            <xsd:element name="sex"type="xsd:string"/>
            <xsd:element name="nation" type="xsd:string"/>
      </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Country_Class">
      <xsd:sequence>
      <xsd:element name="Country_Object" type="oodb:Country_Object"
            maxOccurs="unbounded"/>
      </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Country_Object">
      <xsd:sequence>
            <xsd:element name="name" type="xsd:string”/>
            <xsd:element name="area" type="xsd:int”/>
            <xsd:element name="population type="xsd:int”/>
      </xsd:sequence>
</xsd:complexType>

<!-- Define Primary Keys and Keyrefs -- >

<xsd:key name="Person_PrimaryKey">
 <xsd:selector xpath="oodb:Person_Class/oodb:Person_Object"/>
 <xsd:field xpath="oodb:SSN"/>
</xsd:key>
<xsd:key name="Country_PrimaryKey">
  <xsd:selector path="oodb:Country_Class/oodb:Country_Object"/>
  <xsd:field xpath="oodb:name"/>
</xsd:key>
<xsd:keyref name="Person.nation" refer="oodb:Country_PrimaryKey">
  <xsd:selector xpath="oodb:Person_Class/oodb:Person_Object"/>
  <xsd:field xpath="nation"/>
</xsd:keyref>
</xsd:schema>
```

Figure 3.11: Flat XML Schema for Person and Country Classes

Figure 3.12 represents an example for a flat XML document fragment defined based on the flat XML Schema example in Figure 3.11.

```
<oodb:Person_Class>
   <oodb:Person_Object>
      <oodb:SSN>13</oodb:SSN>
      <oodb:name>YOUNUS</oodb:name>
      <oodb:age>22</oodb:age>
      <oodb:sex>M</oodb:sex>
      <oodb:nation>AUSTRALIA</oodb:nation>
   </oodb:Person_Object>
   <oodb:Person_Object>
      <oodb:SSN>99</oodb:SSN>
      <oodb:name>SARA</oodb:name>
      <oodb:age>21</oodb:age>
      <oodb:sex>F</oodb:sex>
      <oodb:nation>CANADA</oodb:nation>
   </oodb:Person_Object>
. . . . . .
</oodb:Person_Class>
<oodb:Country_Class>
   <oodb:Country_Object>
      <oodb:name>AUSTRALIA</oodb:name>
      <oodb:area>7692024</oodb:area>
      <oodb:population>20000000</oodb:population>
   </oodb:Country_Object>
   <oodb:Country_Object>
      <oodb:name>CANADA</oodb:name>
      <oodb:area>9093507</oodb:age>
      <oodb:population>27000000</oodb:population>
   </oodb:Country_Object>
. . . . . .
</oodb:Country_Class>
```

Figure 3.12: A Fragment for Flat XML Document

3.4 Transforming Object Graph into XML Schema

As described in Section 3.3.2, an object-oriented schema non-primitive attributes and their corresponding links can be presented in an Object Graph (OG) structure. Now based on the graph, the two types of XML Schemas can be derived; the flat and the nested schemas. A pseudo-code for two algorithms is defined, the first takes the OG

structure as an input and generates the flat XML Schema, and the second generates the nested XML Schema. Details follow in the next sections.

3.4.1 Object Graph into Flat XML Schema Transformation

Below is the pseudo-code for an algorithm named OG2FXML that derives a flat XML Schema from the input Object Graph.

Algorithm 3.1 OG2FXML (Object Graph to Flat XML Conversion)

Input: The Object Graph

Output: The corresponding flat XML Schema

1. Transform each node in the object graph (we call it class hereafter) into a "complexType" in the XML Schema.
2. Map each attribute in a class transformed in Step (1) into a subelement within the corresponding "complexType".
3. Create a root element as the object-oriented database schema name and insert each class identified in Step (1) as a subelement with the corresponding "complexType".
4. Define the primary key for each class identified in Step (1) by using "key" element.
5. Map in the object graph each link between classes identified in Step (1) by using "keyref" element.

END Algorithm3.1

To illustrate how the **OG2FXML** algorithm works, we present more details with supporting examples.

- Each node in the OG is represented in XML by a "complexType" element, where this complexType element includes one empty element. Empty elements in XML Schema cannot have content but they can have attributes. The tag of the empty

element is ended by "/" before the final right-angle bracket of the tag. The next schema fragment shows that Person node in the OG is mapped into Person_Class "complexType". Person_Class "complexType" has only one empty element "Person_Object" which has only attributes and no content. Person_Object declaration has the "Person_Object" type definition. This is useful to define complexTypes elements that can be used as a type for other declared elements.

```
<xsd:complexType name="Person_Class">
     <xsd:sequence>
          <xsd:element name="Person_Object"
                type="Person_Object" maxOccurs="unbounded"/>
     </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Person_Object">
     <xsd:sequence>
     . . . . . .

     </xsd:sequence>
</xsd:complexType>
```

The occurrence constraints "minOccurs" and "maxOccurs" are used with value "unbounded" for the Person_Object element defined in Person_Class as shown in the XML fragment above.

- The empty element Person_Object defined in the XML Schema includes subelements that are mapped with the attributes of the class. In this example, subelements of the empty element Person_Object are mapped with attributes of class Person. A schema fragment illustrating this is shown below.

```
<xsd:complexType name="Person_Class ">
   <xsd:sequence>
         <xsd:element name="Person_Object"
               type="oodb:Person_Object" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Person Object">
   <xsd:sequence>
         <xsd:element name="SSN"type="xsd:int”/>
         <xsd:element name="name"type="xsd:string"/>
         <xsd:element name="age"type="xsd:int”/>
         .
         .
         . . . . . .

   </xsd:sequence>
</xsd:complexType>
```

- The “sequence” constructor (called also indicator) in XML Schema is very important because it control the sequence order of information in the XML document. It enforces the appearance of subelements in same order as they were declared within a complexType element. For instance, the “sequence” used in the above XML fragment enforce the subelements (SSN, Name, Age,..) to appear in this order. This process tries to preserve the sequence within classes by creating the instances and attributes of the class in the same order of the XML Schema elements sequence. When retrieving data from a class, the data is extracted in the same sequence it appears in that class structure. As work deals with data-centric documents, the order of instances in different classes is usually not very significant.
- Each class in the OG is mapped into the XML Schema. First it is required to create a root element that represents the entire given object-oriented database. The root element is created as a “complexType” in XML Schema and is given the same name as the object-oriented database schema. Then each class is

inserted as a subelement of the root element. Figure 3.13 is an illustrated example of an XML Schema that can be generated from the algorithm 3.1. The schema defines the root element named UNIVERSITY and contains the complexType elements for the eight classes Person, Country, Student, Staff, ResearchAssistant, Course, Department, and Secretary that are composing the object-oriented database Example 3.1.

```
<xsd:element name="UNIVERSITY">
     <xsd:complexType>
          <xsd:sequence>
               <xsd:element name="Person_Class"
                         type="oodb:Person_Class"/>
               <xsd:element name="Country_Class"
                         type="oodb:Country_Class"/>
               <xsd:element name="Student_Class"
                         type="oodb:Student_Class"/>
               <xsd:element name="Staff_Class"
                         type="oodb:Staff_Class"/>
               <xsd:element name="Research_Assistant_Class"
                         type="oodb:Research_Assistant_Class"/>
               <xsd:element name="Course_Class"
                         type="oodb:Course_Class"/>
               <xsd:element name="Department_Class"
                         type="oodb:Department_Class"/>
               <xsd:element name="Secretary_Class"
                         type="oodb:Secretary_Class"/>
          </xsd:sequence>
     </xsd:complexType>
. . . . . .
</xsd:element>
```

// Now we define all above elements

```
<xsd:complexType name="Person_Class">
      <xsd:sequence>
            <xsd:element name="Person_Object"
                  type="oodb:Person_Object" maxOccurs="unbounded"/>
      </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Person_Object">
      <xsd:sequence>
            <xsd:element name="SSN" type="xsd:int”/>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="age" type="xsd:int"/>
            <xsd:element name="sex"type="xsd:string"/>
            <xsd:element name="spouse" type="xsd:string"/>
            <xsd:element name="nation" type="xsd:string"/>
      </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Country_Class">
      <xsd:sequence>
      <xsd:element name="Country_Object" type="oodb:Country_Object"
            maxOccurs="unbounded"/>
      </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Country_Object">
      <xsd:sequence>
            <xsd:element name="name" type="xsd:string”/>
            <xsd:element name="area" type="xsd:int”/>
            <xsd:element name="population type="xsd:int”/>
      </xsd:sequence>
</xsd:complexType>
. . . . . .
. . . . . .
```

Figure 3.13: Flat UNIVERSITY XML Schema

- The elements “key” and “keyref” are used to enforce the uniqueness and referential constraints. They are among the key features introduced in the XML Schema. Further, we can use “key” and “keyref” to specify the uniqueness scope and to create keys. This is illustrated in next example in Figure 3.14.

```
<xsd:complexType name="Person_Class">
   <xsd:sequence>
         <xsd:element name="Person_Object"
               type="oodb:Person_Object" maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Person_Object">
   <xsd:sequence>
         <xsd:element name="SSN"type="xsd:int”/>
         <xsd:element name="name"type="xsd:string"/>
         <xsd:element name="age"type="xsd:int”/>
         <xsd:element name="sex"type="xsd:string"/>
         <xsd:element name="spouse" type="xsd:string"/>
         <xsd:element name="nation" type="xsd:string"/>
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Country_Class">
   <xsd:sequence>
   <xsd:element name="Country_Object" type="oodb:Country_Object"
         maxOccurs="unbounded"/>
   </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Country_Object">
   <xsd:sequence>
         <xsd:element name="name" type="xsd:string”/>
         <xsd:element name="area" type="xsd:int”/>
         <xsd:element name="population type="xsd:int”/>
   </xsd:sequence>
</xsd:complexType>
<xsd:key name="Person_PrimaryKey">
   <xsd:selector xpath="oodb:Person_Class/oodb:Person_Object"/>
   <xsd:field xpath="oodb:SSN"/>
</xsd:key>
<xsd:key name="Country_PrimaryKey">
   <xsd:selector path="oodb:Country_Class/oodb:Country_Object"/>
   <xsd:field xpath="oodb:name"/>
</xsd:key>
<xsd:keyref name="Person.nation" refer="oodb:Country_PrimaryKey">
   <xsd:selector xpath="oodb:Person_Class/oodb:Person_Object"/>
   <xsd:field xpath="oodb:nation"/>
</xsd:keyref>
```

Figure 3.14: Example for Key and Keyref Constraints

In Figure 3.14 example, “nation” attribute in Person_Class complexType is effectively used as a foreign key pointing to Country_Class complexType, so

”keyref” is used to specify the foreign key relationship between Country_Class and Person_Class. In Person_Class, “nation” attribute is of type Country_Class, so to implement the “key” and “keyref” it is required to map the “name” attribute of Country_Class to the “nation” attribute of Person_Class.

After this illustration, it is clearly shown that the algorithm can generate the flat XML Schema. The next section will discuss the process of generating the nested XML Schema.

3.4.2 Object Graph into Nested XML Schema Transformation

In XML Schema, nested complexType elements are used to define the relation between two elements. One advantage of the nested XML structure is to store all related information in one fragment of an XML document. This reduces the time for data retrieval when users query the XML document. Algorithm 3.2 OG2NXML does the transformation from the object graph to a nested XML structure.

Algorithm OG2NXML depends on the nesting sequence specified in the object graph and generates an output of nested XML Schema. Pseudo-code is depicted as:

Algorithm 3.2 OG2NXML (object Graph to Nested XML Conversion)

Input: The object graph
Output: The corresponding nested XML Schema

1. For classes connected by a link labeled with 1 or 2 in the object graph, we nest the element that corresponds to the class at the head of the arrow inside the element that correspond to the class at the tail of the arrow.
2. For classes connected by a link labeled with 0 do
   Extend the element that corresponds to the subclass to include the content of the element that corresponds to the superclass.

End Algorithm 3.2

To illustrate the nesting process, consider the UNIVERSITY database in Example 3.1; it is taken as input by OG2NXML which generates as output the XML Schema

in a nested structure. In the Object Graph Figure 3.8, an arrow is connecting Person class (node) with Country node where the head of arrow is pointing to Country with a score of connection 1. According to the algorithm 3.2 OG2NXML, Country class should be nested inside Person class. Also, an arrow is connecting Staff class with Person class where the head of arrow is pointing to Person with a 0 score of connection. Person class is a superclass of Staff class, so Staff class inherits the Person class and a new element named Staff_SP1 with type Person is added as a nested element into Staff class in the nested XML Schema presentation. Figure 3.15 is an example of the generated nested XML Schema segment. It illustrates the nesting between Person and Country and the nesting between Staff and Person.

```
<xsd:complexType name="Person_Class">
      <xsd:sequence>
            <xsd:element name="Person_Object"
                  type="oodb:Person_Object" maxOccurs="unbounded"/>
      </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Person_Object">
      <xsd:sequence>
            <xsd:element name="SSN"type="xsd:int”/>
            <xsd:element name="name"type="xsd:string"/>
            <xsd:element name="age"type="xsd:int"/>
            <xsd:element name="sex"type="xsd:string"/>
            <xsd:element name="spouse" type="oodb:Person_Class"/>
            <xsd:element name="nation" type="oodb:Country_Class"/>
      </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Country_Class">
      <xsd:sequence>
      <xsd:element name="Country_Object" type="oodb:Country_Object"
            maxOccurs="unbounded"/>
      </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Country_Object">
      <xsd:sequence>
            <xsd:element name="name" type="xsd:string”/>
            <xsd:element name="area" type="xsd:int”/>
            <xsd:element name="population type="xsd:int”/>
      </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Staff_Class">
      <xsd:sequence>
            <xsd:element name="Staff_Object"type="oodb:Staff_Object"
                maxOccurs="unbounded"/>
      </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Staff_Object">
      <xsd:sequence>
            <xsd:element name="StaffID"type="xsd:int”/>
            <xsd:element name="salary"type="xsd:real"/>
            <xsd:element name= "rank"type="xsd:string"/>
            <xsd:element name="work_in"type="oodb:Department_Class"/>
            <xsd:element name="Staff_SP1" type="oodb:Person_Class"/>
      </xsd:sequence>
</xsd:complexType>

. . . . . .
. . . . . .
```

Figure 3.15: Example for UNIVERSITY Nested Schema Fragment

3.5 Generating XML Document

After the XML Schema is generated using the algorithms OG2FXML and OG2NXML discussed above, the next step is to generate XML document(s) from the object-oriented database Example 3.1. Algorithm 3.3 GenXMLDoc checks top-down through the list of selected objects and generates an element for each object.

Algorithm 3.3 GenXMLDoc (Generating XML Document)

Input: XML Schema and object-oriented database

Output: The corresponding XML Document

Create XML document and set its namespace declaration
Create a root element of the XML document with the same name as the root name of the XML Schema
For each class R in the object-oriented database do
    If R is selected and does not contain any nested classes
        Create R Class element for R
        Let queryString = "select * from R"
        ResultSet = execute(queryString)
        For each object T in ResultSet do
            Create R_Object element for object T
            Create an element for each attribute in R and insert it into R_Object element
    else if R is selected and contains a nested class R_c then
        Create R Class element for R and R_c Class for R_c
        Let queryString = "select selectedAttrs from R, R_c"
        ResultSet = execute(queryString)
        For each object T in ResultSet do
            Create R_Object element for the tuple of R, and R_c_Object element for the object of R_c
            Create an element for each selected attribute in R and insert it to R_Object element, and do same for R_c

EndAlgorithm 3.3

This is a generic algorithm for creating both flat and nested XML documents from an object-oriented database. The idea of the algorithm is to create a complexType element for each class and for each non-primitive domain instance within the class.

The first two lines in GenXMLDoc algorithm creates the XML declaration, namespaces, and the root element of the document. This could be depicted as:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
 xmlns:oodb="http://scim.brad.ac.uk/xml">
<xsd:element name="UNIVERSITY">
    <xsd:complexType>

. . . . . .
     </xsd:complexType>
</xsd:element>
</xsd:schema>
```

The rest of the algorithm describes the process of generating the body of XML document. If the class has primitive domains only, then insert all elements representing those primitive domains into class mapped complexType. The next fragment of the algorithm depicts this.

```
For each class R in the object-oriented database do
    If R is selected and does not contain any nested classes
        Create R Class element for R
        Let queryString = "select * from R"
        ResultSet = execute(queryString)
        For each object T in ResultSet do
            Create R_Object element for object T
            Create an element for each attribute in R and insert it into R_Object
              element
```

If the class has non-primitive domains then insert the corresponding elements of primitive and non-primitive domains of the class into class mapped complexType and insert all elements of the non-primitive domains into their corresponding complex types. The next algorithm segment represents what we have already discussed.

```
    else if R is selected and contains a nested class Rc then
        Create R Class element for R and Rc Class for Rc
        Let queryString = "select selectedAttrs from R, Rc"
        ResultSet = execute(queryString)
```

For each object T in ResultSet do
Create R_Object element for the tuple of R, and R_c_Object element for the object of R_c
Create an element for each selected attribute in R and insert it to R_Object element, and do same for R_c

Creating flat or nested XML document can decide the structure of complex types in the document. Figure 3.16 could be a generated nested XML document fragment from this algorithm.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
xmlns:oodb="http://scim.brad.ac.uk/xml">
<oodb:Person_Class>
   <oodb:Person_Object>
      <oodb:SSN>13</oodb:SSN>
      <oodb:name>YOUNUS</oodb:name>
      <oodb:age>22</oodb:age>
      <oodb:sex>M</oodb:sex>
      <oodb:Country_Class>
           <oodb:Country_Object>
                <oodb:name>AUSTRALIA</oodb:name>
                <oodb:area>7692024</oodb:area>
                <oodb:population>20000000</oodb:population>
           </oodb:Country_Object>
      </oodb:Country_Class>
   </oodb:Person_Object>
   <oodb:Person_Object>
      <oodb:SSN>99</oodb:SSN>
      <oodb:name>SARA</oodb:name>
      <oodb:age>21</oodb:age>
      <oodb:sex>F</oodb:sex>
      <oodb:Country_Class>
           <oodb:Country_Object>
                <oodb:name>CANADA</oodb:name>
                <oodb:area>9093507</oodb:area>
                <oodb:population>27000000</oodb:population>
           </oodb:Country_Object>
      </oodb:Country_Class>
   </oodb:Person_Object>
. . . . . .
</oodb:Person_Class>
</xsd:schema>
```

Figure 3.16: Nested XML Document Fragment

3.6 Conclusion

This chapter discussed the process of mapping an object-oriented database schema and corresponding data into an XML Schema and corresponding XML document. An example of object-oriented schema called “UNIVERSITY” and its related characteristics are described. A presentation of flat and nested XML Schemas and the differences between them is discussed in details. The construction of object graph from the object-oriented schema is discussed. Also, how to extract flat and nested XML Schemas and corresponding documents from the constructed object graph is explained in details. The next chapter will discuss the reverse process; i.e. transferring XML Schema and corresponding data into object-oriented database.

Chapter 4

Transforming XML into Object-Oriented Database Using XML Schema

4.1 Introduction

In this chapter, a reverse process of the work described in chapter 3 is performed. This work maps XML Schema into a generic object oriented database schema and store related XML document into the database. This process preserves the structure of XML document. A directed object graph is used to clarify and simplify the mapping process. Both flat and nested XML Schemas are stored and the superclasses and nesting attributes are handled.

XML emerged as one of the key universal formats for platform independent data exchange between different partners [76]. Since structured databases are widely used to store data, it is important to automate the process of storing XML documents in those databases; this will help in better analysis of the data using the efficient querying facilities of the existing structured databases. Of course, one would like to preserve as much information as possible during the transformation process.

Chapter 3 discussed the process of extracting XML Schema and corresponding XML document from an object-oriented database. This chapter will discuss the reverse process; i.e. storing XML documents into an object-oriented database. The acronym XOG (XML to Object Graph) for the Object Graph will be used to distinguish it from the OG acronym used in chapter 3.

Since many applications store and maintain their data in object-oriented database management systems, an approach is proposed for the transformation of an existing XML Schema into an object-oriented database schema [51] and to transfer the data in the corresponding XML document accordingly. To achieve the mapping, an object graph (XOG) based on the characteristics of the XML Schema is derived; it simply summarizes and includes all complexType elements and the links between them which are the basics of the XML Schema model. With user involvement, suggestions can be made to divide the nested complexType elements into two groups; the elements that are actually nested and the elements that are inherited. User involvement is explained in Section 4.3.1.3. XOG is composed of nodes that represent the complexType elements and the links between these nodes that are derived from nesting and from key and keyref constructs if exist. In this way, everything in a simulated XOG can be directly represented in object-oriented database schema. Finally, data from the XML document is mapped into corresponding XML object-oriented database.

4.2 Related Work

While XML to relational transformation received considerable attention; however, XML-to object-oriented conversion is at least equally important because the latter conversion tends to preserve most of the model characteristics during the conversion process e.g., [2, 20, 70]. More details are found in Section 2.3.2.

4.3 XML to Object-Oriented Transformation Process

A major motivation to carry out this study is the fact that there are more common features between XML and object-oriented databases; thus it is more attractive to

store XML Schema and data, and more data is preserved; refer to Section 3.1 for more details. This is actually backward engineering; the forward engineering part described in [54] and discussed in chapter 3, extracts XML from object-oriented database. This backward engineering process takes a given XML Schema as input and produces a corresponding object-oriented schema. So our main objective is to develop a technique that takes a given XML Schema and corresponding XML document as input and store them into an object-oriented database. This process can be summarized as follows:

- Investigate the characteristics of the XML Schema and derive a summary for it.
- Derive an object graph (XOG) from the investigated XML Schema. This includes inheritance and nesting links.
- Map the derived object graph (XOG) into object-oriented database and build the related schema and classes. The process is capable of taking as input both nested and flat XML Schemas. However, as the mapping is into object-oriented schema, nested XML Schema is preferred and more emphasized.
- Store the corresponding XML document.

The following is a detailed description for the XML to object-oriented transformation process summarized above.

4.3.1 XML Schema Characteristics

This section investigates the characteristics of a given XML Schema and shows how to derive the corresponding object graph (XOG).

4.3.1.1 The Basic Terminology and Definitions

XML Schema can be described as a set of rules and constraints to which an XML document must confine in order to be considered a well-formed and valid document. This work [51] will use XML Schema complexType elements, primitive type (simple) elements and the sequence indicator. Cardinality minOccurs and maxOccurs constraints will also be used. For more details about cardinality refer to Section 2.1.3.2.

Definition 4.1 (ComplexType) A complexType element is defined as a tuple, $C_{complextypes}$ (ct), $C_{primitivetypes}$(ct), C_{keys}(ct), $C_{keyrefs}$(ct), where ct is the complexType identifier, $C_{complextypes}$(ct) is the complexType elements of complexType ct, $C_{primitivetypes}$ (ct) is the set of primitive type elements of ct, C_{keys}(ct) is the set of keys defined for ct, $C_{keyrefs}$(ct) is the set of key references defined for ct.

To demonstrate the complexType concept introduced in Definition 4.1, consider Example 4.1 which starts by a description of some complex types followed by the corresponding XML Schema definition.

Example 4.1 (XML Schema)

In the next set of complex types, each complexType has a set of attributes with their domains and could have a primary key.

Person Complex Type:Key=SSN
Attributes = {ssn :integer; name:string; age :integer; sex :character;spouse :Person; nation :Country}

Country Complex Type:Key= Name
Attributes = {name:string; area :integer; population :integer}

Student Complex Type:Key= StudentID
Attributes = {StudentID:integer; gpa :real; student in :Department; Takes:{(course :Course; grade :string)}}

Staff Complex Type:Key= StaffID

Attributes = {StaffID:integer; salary :integer; works_in:Department}

ResearchAssistant Complex Type:
links to both student and staff; hence gets the key of either one.
Attributes ={student:Student,staff:Staff}

Course Complex Type:Key= Code
Attributes = {Code :integer; title :string; credits :integer; Prerequisite :{Course}}

Department Complex Type:Key= Name
Attributes = {name:string; head :Staff}

Secretary Complex Type:
links to person; hence gets the primary key of person.
Attributes = {person:Person; words_minute:integer; works_in:Department}

4.3.1.2 Nested and Flat XML Schemas

Figure 4.1 depicts a nested XML Schema. It describes the Person and Country complexType elements.

```
<xsd:complexType name=" Person_Object">
      <xsd:sequence>
            <xsd:element name="SSN" type="xsd:int"/>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="age" type="xsd:int"/>
            <xsd:element name="sex" type="xsd:string"/>
            <xsd:element name="spouse" type="Person"/>
            <xsd:element name="nation" type="Country"/>
      </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Country">
      <xsd:sequence>
      <xsd:element name="Country_Object"type="Country_Object"
            maxOccurs="unbounded"/>
      </xsd:sequence
</xsd:complexType>
<xsd:complexType name="Country_Object">
      <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="area" type="xsd:int"/>
            <xsd:element name="population type="xsd:int"/>
      </xsd:sequence>
</xsd:complexType>
```

Figure 4.1: Nested XML Schema Segment for Person and Country Complex Types

This shows that Country element is a child element of the Person element. The Person element has the subelement “nation”, which includes details of the “Country” element that represents the nationality of “Person” element.

The Person complexType element includes an empty Person_Object element, whereas the Person_Object element is defined as a complexType that include all attributes of Person element.

A flat XML Schema can be constructed in the same way as the way nested XML Schema is constructed with the difference that the subelement Country in the “complexType” element Person is defined as a string element type. The two parts of the constructed flat XML Schema are connected by "key" and "keyref" constraints if exist instead of nesting. Figure 4.2 depicts a fragment flat XML Schema. It includes Person_Class and Country_Class complex types. Also it defines the key and keyrefs for both classes. Refer to sections 3.3.3.1 and 3.3.3.2 for more details about flat and nested XML Schemas and documents.

```
<xsd:complexType name="Person_Class">
      <xsd:sequence>
            <xsd:element name="Person_Object"
                  type="oodb:Person_Object"
            maxOccurs="unbounded"/>
      </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Person_Object">
      <xsd:sequence>
            <xsd:element name="SSN" type="xsd:int”/>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="age" type="xsd:int”/>
            <xsd:element name="sex" type="xsd:string"/>
            <xsd:element name=”spouse” type “xsd:string”/>
            <xsd:element name="nation" type="xsd:string"/>
      </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Country_Class">
      <xsd:sequence>
      <xsd:element name="Country_Object"
      type="oodb:Country_Object"
            maxOccurs="unbounded"/>
      </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Country_Object">
      <xsd:sequence>
            <xsd:element name="name" type="xsd:string”/>
            <xsd:element name="area" type="xsd:int”/>
            <xsd:element name="population type="xsd:int”/>
      </xsd:sequence>
</xsd:complexType>

<!-- Define Primary Keys and Keyrefs -- >

<xsd:key name="Person_PrimaryKey">
 <xsd:selector xpath="oodb:Person_Class/oodb:Person_Object"/>
 <xsd:field xpath="oodb:SSN"/>
</xsd:key>
<xsd:key name="Country_PrimaryKey">
  <xsd:selector path="oodb:Country_Class/oodb:Country_Object"/>
  <xsd:field xpath="oodb:name"/>
</xsd:key>
<xsd:keyref name="Person.nation" refer="oodb:Country_PrimaryKey">
  <xsd:selector xpath="oodb:Person_Class/oodb:Person_Object"/>
  <xsd:field xpath="nation"/>
</xsd:keyref>
```

Figure 4.2: Flat XML Schema for Person and CountryComplextypes

4.3.1.3 XML Schema Information

Related to the nested XML Schema, the analysis is based on the domain information summarized in Table 4.1 *XMLAttributesNE* (complexType Name, element name, domain).

Complex Type Name	Element Name	Domain
Person	ssn	integer
Person	name	string
Person	age	integer
Person	sex	integer
Country	name	string
Country	area	integer
Country	population	integer
Student	studentID	integer
Student	gpa	real
Staff	staffID	integer
Staff	salary	integer
Course	code	integer
Course	title	string
Course	credits	integer
Department	name	string
Secretary	wordsperminute	integer
T1	grade	string

(a)

Complex Type Name	Element Name	Domain	Inher-itance Flag
Person	spouse	Person	1
Person	nation	Country	1
Student	Student_in	Department	1
Student	Takes	T1	2
Student	person	Person	0
Staff	work_in	Department	1
ResearchAssistant	student	Student	0
ResearchAssistant	staff	Staff	0
Course	prerequisite	Course	1
Secretary	work_in	Department	1
T1	course	course	1

(b)

Table 4.1: *XMLAttributesNE* (a) Alist of all elements attributes with primitive domain (b) A list of all elements attributes with non-primitive domains

To understand better the content and purpose of this table, Table 4.1 *XMLAttributesNE* includes information about all elements and attributes in the XML Schema given in Example 4.1. For each element, it is necessary to know its complexType name, element name, and the domain. Elements with primitive domains and elements with non-primitive domains are placed in separate occurrences of Table 4.1 *XMLAttributesNE*, namely *XMLElementsNE*(a) and *XMLElementsNE*(b), respectively.

Concerning the information in Table 4.1 *XMLAttributesNE*(b), user involvement is necessary to suggest the elements that are inherited (representing the superclasses in the object-oriented database), the elements that can represent the nested non-primitive domain attributes and the elements that can represent a tuple type domain attributes. To explicate the user involvement, consider the next XML Schema fragment in Figure 4.3 and the Staff class Definition in Figure 4.4.

```
<xsd:complexType name="StaffClass" >
      <xsd:sequence>
            <xsd:element name="StaffID"  type="xsd:int"/>
            <xsd:element name="salary"  type="xsd:int"/>
            <xsd:element name="works_in"  type="DepartmentClass"/>
            <xsd:element name="PersonSuperclass"   type="Person"/>
      </xsd:sequence>
```

Figure 4.3: XML Schema Fragment for Staff Class

```
public class Staff extends Person{

//    private String[] superclasses = {"Person"};
      private int StaffID;
      private int salary;
      private Department works_in;
}
```

Figure 4.4: Staff Class

Class Staff has "Person" as a super class and "work-in" as an instance of class Department. Instance "work-in" is mapped into an XML Schema element named work-in of type DepartmentClass which is in turn a complexType for the Department class. Also "Person" super class is mapped into an XML Schema element named PersonSuperclass of type PersonClass which in turn a complexType for the class Person. Both the instance work-in and the super class Person are mapped to a complexType element in the schema, so there is nothing that can tell about the category of the two elements; i.e. nothing to tell if the element actually separates

nesting or inheritance. Now in the example above, when mapping the same generated schema to an object-oriented database, there is no information about the complexType element PersonSuperclass that tells if this element can be mapped as a super class or as a class instance, and the same applies to work_in element as both element are nested elements in the schema. Even through if nesting and inheritance information are mapped and stored into the complexType elements, there is no guarantee that this information can be available with other XML Schemas For that, the user involvement is required to tell which complexType elements that could be mapped as instances for a class (nested). By default, the rest of nested elements can be handled as super classes (inherited).

In Table 4.1 *XMLAttributesNE*(b), an "Inheritance Flag" is assigned to each element. The score 0 is given for the candidate superclass elements (inherited element), score 1 is given for the nested non-primitive domain elements and score 2 is assigned for the tuple domain elements. Subclasses are not included in the attributes of the class because they have been included when considering the superclasses list of subclasses. To illustrate the “Inheritance Flag”, in Table 4.1 *XMLAttributesNE*(b) the "nation" element is given the value 1 because it is a nested non-primitive domain. The “Inheritance Flag” in row 3 is given the value 1 because student_in of type Department is a nested type, while it is given the value 0 in row 5 because Person is a candidate superclass for Student (inheritance). Also, row seven is give the value 0 as ResearchAssistant is a subclass of Student and Staff; that means Student and Staff are superclasses for ResearchAssistant. This way, it becomes clear to identify superclasses, subclasses and non-primitive domain attributes using Table 4.1 *XMLElementsNE*(a) and *XMLElementsNE*(b).

Related to the flat XML Schema, the analysis is based on the domain information summarized in Table 4.2 *XMLAttributesFL (complexType name, element name, domain, expected domain,inheritanceflag, keys information, key ref information).*

Complex Type Name	Element Name	Domain	Expected Domain	Inheritance Flag
Person	ssn	integer	Integer	9
Person	name	string	String	9
Person	age	integer	Integer	9
Person	sex	integer	Integer	9
Person	spouse	string	Person	1
Person	nation	string	Country	1
Country	name	string	String	9
Country	area	integer	Integer	9
Country	population	integer	Integer	9
Student	studentID	integer	Integer	9
Student	gpa	real	Real	9
Student	student_in	string	Department	1
Student	Takes	T1	T1	2
Student	person	integer	Person	0
Staff	satffID	integer	Integer	9
Staff	salary	integer	Integer	9
Staff	work_in	integer	Department	1
ResearchAssistant	student	integer	Student	0
ResearchAssistant	staff	integer	Staff	0
Course	code	integer	Integer	9
Course	title	string	String	9
Course	credits	ineger	Ineger	9
Course	prerequisite	string	Course	1
Department	name	string	String	9
Secretery	word_minute	integer	Integer	9
Secretery	work_in	string	Department	1
T1	Course	string	Course	1
T1	grade	string	String	9

***XMLElementsFL*(a)**

Key Name	Complex Type Name	Element Name
Person_pk	Person	ssn
Country_pk	Country	name
Student_pk	Student	studentid
Staff_pk	Staff	staffid
Course_pk	Course	code
Department_pk	Department	name

***XMLKeys*(b)**

Key Reference Name	Ref. Complex type	Ref. element	Refer to Element
Person.nation	Person	nation	country_pk
Student.Student_in	Student	Student_in	Department_pk
Secretery.work_in	Secretery	work_in	Department_pk

***XMLKeyRefs* (c)**

Table 4.2 *XMLAttributesFL*:
(a) A List of All Elements Attributes with Primitive and Non-Primitive Domains
(b) A List of All Keys for the Complex Type Elements
(c) A List of All Key References of the Complex Type Elements

Table 4.2 *XMLAttributesFL* represents the flat XML Schema. To understand better the content and purpose of this table, Table 4.2 *XMLAttributesFL* includes information about all elements, attributes, keys and keyrefs in the XML Schema given in Example 4.1 and Figure 4.2. For each element, it is necessary to know its complexType name, element name, domain, expected domain name, and the inheritance status (inherited or not). Also it is necessary to know the complexType elements "keys" and "keyrefs". Information about the elements is placed in Table 4.2 *XMLElementsFL(a)*, keys information is placed in Table 4.2 *XMLKeys(b)*, and key reference information is placed in Table 4.2 *XMLKeyRefs*(c).

In Table 4.2 *XMLElementsFL*(a), user involvement is required to suggest which element is inherited (a candidate superclass in the object-oriented database), the element that can represent the nested non-primitive domain attributes and the element that can represent the tuple type domain attributes. For each element, a value is assigned to “Inheritance Flag”. The score 0 is given for the inherited element (the candidate superclass elements), score 1 is given for the nested non-primitive domain elements, score 2 is assigned for the tuple domain elements, and score 9 is assigned to the primitive domain elements. Score 9 is given because primitive domains are different from all other non-primitive domain, so this gap is kept for any future new non-primitive domain types. User involvement and the information available in Table 4.2 *XMLKeys*(b) and *XMLKeyRefs*(c) can define the expected non-primitive domain for flat XML primitive domain elements. For instance, the "nation" in complexType Person is a primitive type element of type “string” and expected to have the score 9 for the "Inheritance Flag" but it is given the value 1 instead. This is because by analysing and linking the information in Table 4.2 *XMLKeys*(b) and in

XMLKeyRefs(c), it is shown that there is a reference link between the “nation” element of complexType Person and the “country_pk” of Country complexType, so the expected domain is Country and not “string”. Row 1 is given the score 9 because it is a primitive domain element. In Student “complexType” element, "Person" element of primitive type is given the score 0 because user involvement can decide that this is an inherited element, and thus it could be mapped as a superclass for Student.

As a result, it becomes not so hard to construct Table 4.1 - which represents the nested schema - from information in Table 4.2. Explicitly, primitive domains in Table 4.2 *XMLElementsFL(a)* can be mapped into Table 4.1 *XMLElementsNE*(a), and expected non-primitive domains of Table 4.2 *XMLElementsFL(a)* can be mapped into Table 4.1 *XMLElementsNE*(b). This way, it becomes clear to identify superclasses, subclasses and non-primitive domain attributes using Table 4.1 *XMLElementsNE*(a) and Table 4.1 *XMLElementsNE*(b). This is useful because one algorithm can be used to handle both flat and nested XML Schemas as the anticipated input for the algorithm is Table 4.1 *XMLAttributesNE.*

To sum up, the information needed for mapping XML Schema into the object-oriented database is summarized in Table 4.1 *XMLElementsNE*. This table is derived directly from a nested XML Schema. However, for flat XML Schema, the process involves preprocessing step to derive the information in Table 4.2 *XMLElementsFL* which in turn is used to construct Table 4.1 *XMLElementsNE*. This opens the door for a new relational to object-oriented database conversion by converting a relational database directly into a flat XML Schema and then mapping the latter into object-oriented schema.

4.3.2 The Object Graph (XOG)

In this section, information presented in Table 4.1 *XMLElementsNE* and complexType information defined in Section 4.1 is used to draw the Object Graph that includes nodes that represent complexType elements and all possible relationships between them. Two nodes are connected by a link to show the inheritance or a nesting relationship between them. Nodes are represented by small rectangles and links are represented by directed arrows. Inheritance links are assigned the scores 0 and nesting links are assigned the scores 1. A link is assigned the score 2 if it is connecting a node that represents a tuple (element) domain and the entity in which it is referenced. To illustrate this, refer to the attribute Takes in Student in Example 4.1 and to the corresponding link connecting the two nodes T1and Student in Figure 4.5.

Definition 3.2 in Section 3.3.2, defines the Object Graph (OG) for the object-oriented schema while the next definition 4.2 is for defining the Object Graph from XML Schema (XOG). More details related to object graph are included in Definition 4.2, given next.

Definition 4.2 Every XML Schema has a corresponding object graph (V,E) such that,

1. For every complextype ct in the XML Schema there is a corresponding node ct in V,

2. For every complextype ct

For every element e $\in C_{complextypes}(ct)$*, such that e has a non-primitive domain,*

If domain of e invovles a compextype e' then an edge (ct, e', InheritanceFlag) is added to E, where InheritanceFlag is defined in table XMLElements (b)

Else if domain of e involves an element T_i*, (i≥1) then a node* T_i *is added to V and an edge* $(T_i, ct, 2)$ *id added to E.*

For every complextype ct″ that appears as a domain in element T_i*, an edge* $(T_i, c'', 1)$ *is added to E.*

As Example 4.1 is concerned, shown in Figure 4.5 is the object graph derived from the information present in Table 4.1 (b) and the inheritance information provided by an expert based on the content of Table 4.1.

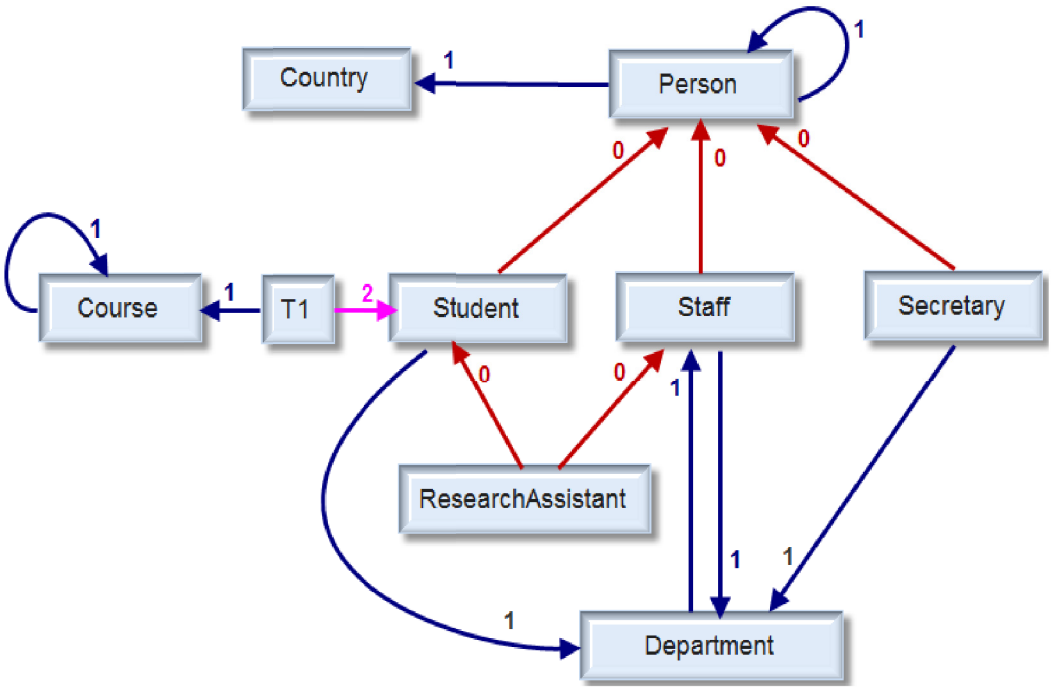


Figure 4.5: Object Graph for the XML Schema in Example 4.1

4.3.3 Transforming Object Graph into Object-Oriented Schema

In this section, an algorithm will be presented for transforming the object graph to object-oriented Schema (XOG2OODB).

Algorithm 4.1 XOG2OODB (Object Graph to Object-Oriented Schema Conversion)

Input: The Object Graph

Output: The corresponding object-oriented Schema

1. Transfer each node in the XOG (it is called complexType hereafter) into a class in the object-oriented schema. Exclude nodes like T_i, ($i \geq 1$).
2. Map each subelement of primitive type in Table 4.1 *XMLElementsNE*(a) into a primitive attributes in the corresponding class. Exclude subelements like T_i, ($i \geq 1$).
3. Map each subelement of non-primitive domain with score 1 defined in Table 4.1 *XMLElementsNE*(b) into the non-primitive attributes in the corresponding class. Exclude subelements like T_i, ($i \geq 1$).
4. Map each subelement of non-primitive domain with score 2 defined in Table 4.1 *XMLElementsNE*(b) as a tuple non-primitive attributes in the corresponding class. Add to this tuple non-primitive attributes all elements of complexType name equivalent to its domain.
5. Add each subelement of non-primitive domain with score 0 defined in Table 4.1 *XMLElementsNE* (b) into the superclasses list of the corresponding class.

EndAlgorithm 4.1

To understand the steps of Algorithm 4.1, more details with supporting examples will be presented.

Each complexType E in the XOG is translated into a class of the same name E in the object-oriented schema. In each "complexType" E, there is only one empty element, which includes several subelements. Those primitive and non-primitive subelements and their domains are mapped into class attributes with same domains. Also, superclasses of the class are added to its superclasses list. Information related to three example classes is given next; only attributes, superclasses and subclasses are shown; functions are excluded because they are trivial. Each attribute satisfies encapsulation by having two corresponding functions, one to set its value and one to return its value.

Person class can be depicted as
$Person_{attributes}$ = {ssn :integer; name:string; age :integer; sex :character; spouse :Person; nation Country}
$Person_{superclasses}$ = [ ]
$Person_{subclasses}$={Student, Staff, Secretary}

Country class can be depicted as
$Country_{attributes}$ = {name:string; area :integer; population :integer}
$Country_{superclasses}$ = [ ]
$Country_{subclasses}$ = [ ]

Student class can be depicted as
$Student_{attributes}$ = {StudentID:integer; gpa :real; student in :Department; Takes:{(course :Course; grade :string)}}
$Student_{superclasses}$ = [Person]
$Student_{subclasses}$={ResearchAssistant}

4.3.4 Transforming XML Document into Object-Oriented Database

In this section, an algorithm will be presented for transforming the XML document to object-oriented database (XMLDoc2OODB).

Algorithm 4.2 XMLDoc2OODB (Storing XML Document into OODB)

Input: XML Schema, Corresponding XML Document, Generated object-oriented database

Output: Storing XML Document into OODB

```
If XML Schema is flat then
    for each complexType ct in flat XML Schema do
        let C = mapped class of ct
        Create T elements for ct
        for each complexType ci within ct do {
            let queryString = "select * from T"
            ResultSet = execute (queryString)
                for each element E in ResultSet do {
                    C(E) = ci(E)
                }
            Store class C
        }
else if Schema is nested then
    for each second hierarchy level complextype ct in nested XML Schema do
        Create T elements for ct and Tn elements for nested complextypes in ct
        For each complexType ci in ct do {
            let C = mapped class of ci, Cn = mapped class of Tn
            let queryString = "select * from T,Tn"
            ResultSet = execute (queryString)
                for each element E in ResultSet do {
                    C(E) = T(E)
                    Cn (E) = Tn(E)
                }
            Store class C   // related classed are automatically stored
        }
```

EndAlgorithm 4.2

To understand the steps of Algorithm 4.2, more details will be presented. Each complexType E in the flat XML Schema is mapped with the same name of the class created by algorithm 4.1. In each "complexType" E, there are complexType

subelements E2 that include the all data instances of the class. Each complexType element E2 is recursively stored into mapped class of the object-oriented database.

In nested XML Schema, each complexType E is also mapped with the same name of the class created by algorithm 4.1. In each “complexType” E, there are complexType subelements E2 that include the data of the class and the data of inheritance (superclasses) or the data of nested classes. Each complexType element E2 is recursively stored into mapped class of the object-oriented database. All related nested or inherited classes are stored during the store of the main class that includes their data.

4.4 Conclusion

This chapter discussed the process of mapping an XML Schema into object-oriented database schema and storing the corresponding XML document into the database. This is a reverse process of what has been discussed in chapter 3. An example of an XML Schema was proposed and related characteristics were defined. The defined schema is populated into a nested XML Schema structure and presented by set of tables. The same approach is followed for the flat XML Schema that allocates the characteristics into another set of table structure. The flat XML Schema tables structure are mapped to match the nested XML Schema tables structure. Then, this unified structure is used for constructing the object graph. The object graph and the unified schema tables are used for creating the object-oriented schema and storing the corresponding XML document. The user involvement discussed in this chapter can be minimized using different scenarios explained in details in Section 7.2.

Chapter 5

Implementation for converting between Object-Oriented Database and XML

5.1 Introduction

The objective of the implementation is to test the approaches explained in chapters 3 and 4. Also, a proposed framework for combining the two way mapping discussed in chapter 3 and chapter 4 is explained. In the implementation process, an XML Schema and XML document are constructed from an object-oriented database. After inspecting different options, two approaches were followed. The first approach is using one of the known object oriented databases db4o, while the second approach is using Java and Java classes as a Customised object-oriented database. OODB db4o supports both object oriented programming languages Java and C Sharp (C#). The latter (Java) approach could be better because the implementation will not be limited to a specific object-oriented database schema; i.e. this can define a generic object-oriented database schema. Java is an object-oriented development language that supports most of the object-oriented languages features such as encapsulation, polymorphism, inheritance, and others. In other words, it is the "lingua franca" of object-oriented databases. The availability of Java as a free tool for personal use is another consideration. Moreover, developers and users have limited control over the structure of open source object-oriented databases comparing to Java that developers have full control on the generated source of a database schema. The next Section 5.2 explains the implementation of the second approach; Customised Java database using

Java classes. Section 5.3 explains the implementation using the db4o object oriented database.

5.2 Customised Java Classes Implementation

This section explains the implementation of Java and Java classes as a Customised OODB.

5.2.1 Defining the Object-Oriented Database

In Example 3.1, an object-oriented database schema named "UNIVERSITY" is defined. It is composed of classes Person, Country, Student, Staff, ResearchAssistant, Course, Department, and Secretary. Also detailed characteristics of each class are described. In the implementation, UNIVERSITY schema is defined as Java classes. Fiigure 5.1 is an example of a Java class code that defines the Staff class.

```
// class Staff
// primitive attribute int StaffID
// primitive attribute int salary
// nonprimitive attribute Department works_in
// superclasses Person
// End Class Definition

public class Staff extends Person {
      private int StaffID;
      private int salary;
      private Department works_in;

}
```

Fgure 5.1: Java Code for Staff Class

The data dictionary information is extracted using Java reflection mechanism. Reflection in Java provides the ability to obtain information about the characteristics of the class such as constructors, methods, attributes, and modifiers. The commented

statements on top of the Java code starting with " // class Staff " and ends with "// End Class Definition" summarize the data dictionary information. It mentions the primitive attributes, non-primitive attributes, superclasses, keys and so on. The statement "// primitive attribute in StaffID" summarizes the instance StaffID as a primitive (simple) type instance, while the statement"// non-primitive attribute Department works_in" shows a non-primitive (complex) instance work_in of type Department. All classes in the UNIVERSITY schema are defined in similar way and have been located in a given directory named OODB.

5.2.2 Extracting the Object-Oriented Database Schema

Set of Java programs are developed to extract the object-oriented database schema data dictionary information and then to construct the XML Schema. Also they are used to extract the data and generate the XML document. These are the major steps followed to achieve this goal:

- Read all Java classes of type "**.java**" from the object-oriented database directory OODB.
- Use Java reflection feature to extract primitive attributes, non-primitive attributes, superclasses and so on. Superclasses are considered as non-primitive attributes of their subclasses.
- Construct working area arrays to populate the extracted data dictionary information.

5.2.3 Creating XML Schema

This process composes the XML Schema out of the constructed data arrays. First, It generates the XML declaration statement and the schema namespace statement shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

Declaration and namespaces statements should appear at the beginning of the XML document. Each class is mapped into a “complexType” element in the XML Schema. A naming convention is followed for those “complexType” elements. It takes the class name and concatenates the word “Class” to it. For example the Country class will compose a “complexType” element named as "CountryClass". The XML "sequence" construct is created to contain all subelements with the same sequence as they appear in the object-oriented database class. All primitive instances are mapped into simpleType elements and all non-primitive instances and the superclasses of the class are mapped as complexType elements within the class complexType. Figure 5.2 is a generated segment of an XMLSchema for the class Country.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:complexType name="CountryClass" >
            <xsd:sequence>
                  <xsd:element name="name"  type="xsd:string"/>
                  <xsd:element name="area"  type="xsd:int"/>
                  <xsd:element name="population"  type="xsd:int"/>
            </xsd:sequence>
      </xsd:complexType>
</xsd:schema>
```

Figure 5.2: Generated XML Schema Segment

Appendix A, include the UNIVERSITY object-oriented database schema with the data dictionary information. UNIVERSITY nested XML Schema generated from the UNIVERSITY object-oriented database example is presented in Appendix B, while

UNIVERSITY flat XML Schema is found in Appendix C. Appendix D includes flat XML document generated from object-oriented database.

5.2.4 Constructing the XML Document

XML document is extracted from the object-oriented database using the generated XML Schema. Algorithm 3.3 GenXMLDoc can generate both flat and nested XML document. More details about the Algorithm 3.3 GenXMLDoc can be found in Section 3.3.5. Figure 5.3 represents a flat XML document fragment generated from this algorithm, while Appendix D includes a flat XML document generated from UNIVERSITY object-oriented database.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
 xmlns:oodb="http://scim.brad.ac.uk/xml">
<oodb:Person_Class>
   <oodb:Person_Object>
      <oodb:SSN>233</oodb:SSN>
      <oodb:name>Mick</oodb:name>
      <oodb:age>45</oodb:age>
      <oodb:sex>M</oodb:sex>
      <oodb:nation>United Kingdom</oodb:nation>
   </oodb:Person_Object>
. . . . . .
</oodb:Person_Class>

<oodb:Country_Class>
   <oodb:Country_Object>
      <oodb:name>United Kingdom</oodb:name>
      <oodb:area>244820</oodb:area>
      <oodb:population>60000000</oodb:population>
   </oodb:Country_Object>
. . . . . .
</oodb:Country_Class>
</xsd:schema>
```

Figure 5.3: Generated Flat XML Document Fragment

5.3 Implementation Using OODB db4o

In Section 5.2, a Customised Java classes object-oriented database is implemented. Another way of implementation is performed by using one of the well-known open source object-oriented databases currently available in the market; db4o database. This object-oriented database is the first database that had implemented the native queries proposed in work described in [21, 22]. Native queries are written in object oriented programming languages themselves such as Java and C# rather than query languages. The next example Java segment shows a native query for retrieving information of Person.

```
public static void retrievePersonNQ(ObjectContainer db) {
    List<Person> result=db.query(new Predicate<Person>() {
  public boolean match(Person person) {
        return true;
  }
    });
    listResult(result);
}
```

Also, query by example can be implemented with this database. The next Java program segment retrieves all Person classes using query by example.

```
public static void retrieveAllPersons(ObjectContainer db) {
        ObjectSet result = db.queryByExample(Person.class);
        listPerson(result);
}
```

Both, native query and query by example can retrieve information for classes with specific information. The next Java segment illustrates retrieving Person class with the attributes age is greater than 20 and name is "Mary Tomson".

```
public static void retrievePersonNQ(ObjectContainer db) {
    List<Person> result=db.query(new Predicate<Person>() {
  public boolean match(Person person) {
    return person.getName().equals("Mary Tomson")
                        && person.getAge > 20;
  }
    });
    listResult(result);
}
```

A set of Java programs are written and implemented using db4o database classes and interfaces. The process can be summarized as follows:

- UNIVERSITY object oriented data example is created inside db4o and the data is stored into the database. The next Java segment in Figure 5.4 shows the creation of the database, the creation of Person and Country classes, and the insertion of data into the database.

```
import com.db4o.Db4oEmbedded;
import com.db4o.ObjectContainer;
import com.db4o.ObjectSet;
import com.db4o.config.EmbeddedConfiguration;
import com.db4o.query.Predicate;
import com.db4o.query.Query;
import java.io.File;
import java.util.List;
public class CreateDB {
   public static void main(String[] args) {
      new File("D:/newimp/oodb.db4o").delete();
      ObjectContainer db = Db4oEmbedded.openFile
   ("D:/oodb.db4o");
             try {
              storePersons(db);
             } finally {
              db.close();
               }
   }
   public static void storePersons(ObjectContainer db) {
      Person person1 = new Person(115,"Tushi Imamura",50,"M");
      Person spouse1 = new Person(50,"Suzuki  Yoshikawa",40,"F");
            Country JAP = new Country ("Japan",400000,120000000);
            spouse1.setNation(JAP);
            person1.setNation(JAP);
            person1.setSpouse(spouse1);
            db.store(person1);
      }
```

Figure 5.4: Java Segment for Creating Person and Country Classes Using db4o Database

- Data is retrieved and verified for accuracy. The next Java segment in Figure 5.5 shows a dump for County class using query by example and standard methods.

```
import com.db4o.Db4oEmbedded;
import com.db4o.ObjectContainer;
import com.db4o.ObjectSet;
import com.db4o.config.EmbeddedConfiguration;
import com.db4o.query.Predicate;
import com.db4o.query.Query;
import java.util.*;
import java.io.File;
import java.util.List;

public class RetDB {
   public static void main(String[] args) {
         ObjectContainer db =
Db4oEmbedded.openFile("oodb.db4o");
           try {
           retrieveAllNatsQBE(db);
//         retrieveAllNats(db);

         } finally {
         db.close();
           }
   }
   public static void retrieveAllNatsQBE(ObjectContainer
db) {
         Country nat = new Country(null,0,0);
         ObjectSet result = db.queryByExample(nat);
         listResult(result);
   }
   public static void retrieveAllNats(ObjectContainer db) {
         ObjectSet results =
db.queryByExample(Country.class);
         listResult(results);
   }
    public static void listResult(ObjectSet result) {
          while (result.hasNext()) {
              System.out.println(result.next());
          }
          System.out.println("---------------");
    }
}
```

Figure 5.5: Java Segment for Dumping Country Class Data Using Different Queries methods

- University object oriented schema is extracted from the db4o database.
- A nested XML document is generated from this database. This document is shown in Appendix E.

This implementation using db4o OODB used the same input data as the previous implementation discussed in Section 5.2. The generated outputs (XML Schema and corresponding XML document) of this implementation are the same as the outputs generated from the former implementation.

5.4 Presenting COODaX

5.4.1 Introduction

In [52], a flexible approach is introduced for representing the object-oriented database into XML format named COODaX (**C**onverting **O**bject-**O**riented **Da**tabases to **X**ML). This approach is based on work presented in [54, 51] that have been thoroughly explained in chapters 3 and 4 and depends on previous findings on reverse engineering of legacy databases explored in [3]. The main purpose of COODaX system is to extract an XML Schema and corresponding document from an object-oriented database.

Two basic steps are identified in the process of transforming object-oriented databases into XML. The first step is reconstructing the object-graph from the given object-oriented database. The generated object-graph summarizes the structure of the object-oriented schema by exploring the inheritance and nesting links of the given object-oriented database. Second, the obtained object-graph is transformed into XML Schema in a process known as forward engineering. In the first step, the process requires knowing the metadata. Also, the opportunity to bypass the metadata (which

may not be fully available) and to apply reverse engineering to derive information about inheritance and nesting is introduced. In other words, COODaX either utilizes the metadata when it is available, or applies reverse engineering to extract information missing from the metadata. COODaX enable users to view the underlying object-oriented data as either flat or nested XML structure Also, users can specify the nesting sequence. Users can directly view the result of each phase during the process.

5.4.2 COODaX Architecture

COODaX is composed of four main modules:

1) **EOGOR** – **E**xtracting **O**bject **G**raph from **O**bject Oriented Database by **R**everse Engineering Module. This module in COODaX extracts all possible schema information (meta-data) by analysing the content of the given object-oriented database, and then generates the OG. Details can be found in Section 5.4.3.

2) **EOGMOD**- **E**xtracting **O**bject **G**raph from **M**eta-data of **O**bject-oriented **D**atabase Module. It has a function similar to EOGOR, but it generates the OG model from an existing database catalogue; it employs EOGOR in case some catalogue information is missing

3) **OG2X**- **OG to X**ML Module.

   It takes the OG generated from EOGOR or EOGMOD as input, and applies the OG to XML transformation algorithm to obtain the XML Schema. It can also generate the XML documents.

4) **X2OG- XML Schema to OG** Module.

It takes the XML Schema file as input, generates the OG, and then runs the XML Schema to object-oriented schema transformation algorithm to generate the corresponding object-oriented schema; it also stores the result in object-oriented database.

The next sections of 5.4 explain in details the COODaX components.

5.4.3 Extracting the Object Graph

The OG contains one node (vertex) per class and two types of edges; inheritance and nesting. There are two ways to extract data from the object-oriented database; either by content-based or by catalogue-based. **EOGOR** module is used to extract all possible information by analysing the content of the object-oriented database, usually legacy database, while **EOGMOD** extracts data from the catalogue of the database. Any missing information from the catalog could be extracted using **EOGOR**. Constructing OG from the catalogue-based analysis is explained in *Definition 3.2*. Extracting OG by analysing database contents can be depicted in the following steps:

1. For each class in the object-oriented database schema: find the domains of its attributes.
2. For each class c: identify candidate direct inheritance links between c and every class c_i such that the list of domains for attributes of c is a subset of the list of domains for attributes of c_i (denoted c<<ci), and there is no class c_j, where $c<<c_j<<c_i$.

3. For each class c, identify every class c_k, such that $c_k$is the domain for an attribute in c.
4. Use the result from Steps 2 and 3 to construct the initial OG. We can apply the same rules explained in *Definition 3.2*.

5.4.4 Transforming Object Graph Model to XML Schema

COODaX can properly handle all types of relationships in the conversion from object-oriented database into XML. The OG2X module is responsible for transforming the object-graph of the given object-oriented database into the corresponding XML Schema. The schema itself is an XML document, and so can be processed by the same tools that read the XML documents it describes. The XML Schema supports rich built-in types and allows building complex types based on built-in types.

We also consider mapping all different types of the constraints particular to the class hierarchy, including: object-identifiers (denoted PKs), object-references from within other objects (denoted FKs), null/not-null, unique, etc, to the XML Schema. Basically, the null/not-null constraint can be easily represented by properly setting "minOccurs" of the XML element transformed from the object-oriented attribute. The unique constraint can also be represented by the unique mechanism in the XML Schema in a straight forward manner.

The OG2X module by default generates a flat structure of the XML Schema. However, users may specify a nested structure in a way to improve the performance of querying XML documents. The conversion into flat XML Schema is explained in

details in Algorithm 3.1 OG2FXMLand the conversion into nested XML Schema is thoroughly explained in Algorithm 3.2 OG2NXML.

5.4.5 Generating XML Document

After the XML Schema is obtained, COODaX can generate XML document(s) from the considered object-oriented database. It uses the XML Schema to decide which type of XML document will be generated. If the XML Schema is nested, then a nested XML document is generated and if the schema is flat, then a flat document is generated. This process is explained in details in Algorithm 3.3 GenXMLDoc.

5.4.6 Transforming XML Schema into Object-Oriented Schema

We use the same approach discussed in Section 4.3.1 to construct the OG from an XML Schema. We also use Algorithm 4.1 OG2OODB that transforms the object graph to object-oriented Schema.

5.5 Conclusion

In this chapter Java classes are used to represent an object-oriented database. Flat and nested XML Schemas are generated from object-oriented UNIVERSITY database example and the flat XML document is generated from the database. Another way of implementation is presented by using the open source object-oriented db4o. Nested Schema and nested document is generated from this implementation. The set of Java classes used for the implementation of the generic object oriented approach are used for the implementation of db4o approach. With simple extension and incorporation of db4o database classes, the implementation of the same examples used in the

generic object oriented database are successfully performed. This clearly proves that classes for the generic approach can be used as a base for any implementation with other OODBMS.

COODaX framework is presented. It has a GUI interface to extract and store flat and nested XML Schemas and corresponding documents from/to object-oriented databases. This system extracts the meta-data from the database catalogue, and when database catalogue or part of it is not available, it investigates and extract the missing meta-data from the data itself.

Chapter 6

Mapping between Object Definition Language (ODL) and XML Schema

6.1 Introduction

Chapter 3 and chapter 4 discuss in details the mapping between a generic OODB model and XML, including both structure specification and database content. An intermediate graph is used to map between OODB and XML Schema. This chapter discusses another approach of mapping; the mapping between Object Definition Language (ODL) and XML. One of the major components for ODMG 3.0 specification [15] is the object specification language Object Definition Language. It is used to define the semantic constructs of the object database schema, operations, and specifications of object types that conform to the ODMG object model. As ODL is programming languages independent, it supports the portable object schemas across ODMG-compliant object data management systems (ODMSs) and facilitates the migration of data. ODL is used in the same way as the data definition language (DDL) of the traditional databases such as relational database. If java is the "lingua franca" of object-oriented languages, we also consider ODL as the "lingua franca" of object-oriented specification.

In this work [40], a comprehensive approach is presented for the mapping between the object database language (ODL) and XML, including both structure specification and database content. This study concentrates on deriving a set of transformation rules for two way mapping between ODL and XML. For the mapping from ODL to XML, the set of rules described in [39] are expanded and developed. The fact that the

rules only cover a subset of ODL, as well as the fact that the rules provided a solid foundation for expansion, is the main motivation for continuing the study. After analysing and evaluating the correctness and completeness of the rules, some improvements and extensions are proposed to have a complete set of transformation rules. By modifying the existing rule set, a wider variety of ODL to XML mapping is able to be handled. Also some ODL scenarios are discussed that the original rule set cannot handle. The presented complete rule set is capable of handling a larger subset of ODL (including dictionaries, global and local scope enumerations, and most importantly, inheritance). No research is encountered for XML to ODL mapping. Hence, a complete set of rules capable of handling the mapping process is developed. Finally, a set of rules for handling the two-way mapping of data is proposed. Implementation for two-way mapping between ODL and XML is performed to ensure effectiveness and correctness. This work could be a solid base for future investigation of transformation between ODL and XML.

6.2 A Comparison between Object Graph to XML Mapping and ODL to XML Mapping

The main work described in chapters 1, 2, 3, 4, and 5 is explaining in detail the mapping approach between OODB and XML using object graph. The approach of mapping between ODL and XML has common similarities with the former approach. The main similarities can be summarized as:

- OODB to XML mapping is depending on the OODB schema information and in case catalogue information is not available, the database content is analysed and the missing schema information is built; refer to section 5.4.3.

The same analogy, ODL to XML is using a similar deductive approach for exploring the schema.

- OODB to XML is using an intermediate graph as a means for mapping and the same OG is used with implementation of ODL to XML mapping.
- Both approaches handle the mapping of schemas as well as the transformation of related data.
- OODB to XML is using a generic schema where it can theoretically apply to any object oriented data. While object oriented database management systems have differences in their object model, they agree on some basics of ODL. So the approach of ODL to XML could also work with any object oriented database.
- There is a clear analogy when mapping classes to complex types and defining the keys and keyrefs.

The comparison between OODB and XML mapping using OG or XOG and ODL and XML mapping can be summarized into Table 6.1.

Major Mapping Attributes	OG / XOG Mapping	ODL and XML Mapping
Object Graph is Used in Core of Mapping	Yes	NO
Set of Algorithms are used to Construct OG and to Generate Flat and Nested XML Schemas	Yes	NO
Generic OODB is used	Subset of Structure	Wider Range of Structure
Complex XML Schema Structure is Used in the Mapping Process	Subset	Subset
Corresponding Documents to Schema are Handled	Yes	Yes
Mapping Dictionaries and Lists	NO	YES

Table 6.1: Comparison between (OODB and XML) and (ODL and XML) Mapping

6.3 Previous Work

The work in [65] presents a set of rules to help in the transformation from ODL to XML Schema. It makes the argument that the ODMG standard [15] has reached a sufficient level of maturity, and therefore, it is a valid and attractive alternative for the storage of XML data. The work describes the fundamental cornerstone concept: in order to store XML documents in conventional databases (relational, object-relational, and object oriented), a certain amount of translation work is unavoidable.

The work in [39] presents an approach for the transformation of ODL Schemas into XML Schemas. The approach starts with an incomplete set of rules described in the literature to assist in the transformation process. This work presumes that the global adoption of XML will push forward the acceptance of object-oriented databases as the most appropriate repository for maintaining XML data without sacrificing the semantics of the XML model as when storing XML into relational database. Thus, they believe that more rule based work should be done regarding bi-directional translation between XML documents and object-oriented databases. It is then asserted that ODL is the language of choice for specifying the object-oriented database schema because it is the proposed language standard from the ODMG. This will result in high portability between different object-oriented databases. The authors address the issue that the rules are not complete, and that more work must be done. First, new rules are needed to handle more complex object-oriented modeling constructs. Second, a new set of rules are needed for conversion in the reverse direction (XML Schemas to ODL). Third, some more rules will be needed for data migration. This work is thoroughly handling all these three mentioned aspects. The

next sections discuss in detail this mapping approach

6.4 Rules for Conversion from ODL structure to XML Schema

ODL is a well-structured language that allows for specifying the definition of an object-oriented schema, including classes, inheritance and attributes. Given an ODL specification, the goal is to produce a corresponding XML Schema. The XML Schema has elements and attributes in addition to sequences, keys/keyrefs, cardinality, etc. Rules are defined to provide a smooth transformation of an ODL schema into XML Schema by considering and maintaining as much as possible the characteristics of each model. The process starts from the root, then moves on to the classes and finally considers the inheritance and the attributes. Further, complex attributes are considered in more details by investigating and mapping the details of the underlying domain. The sample ODL schema shown in Figure 6.1 will be used as a running example for illustrating the conversion rules. The resulting XML Schema will be shown at the end after all the rules are introduced. The following are ODL to XML Schema (O2X) conversion rules.

```
class author;
class person;

class book (extent books key ISBN)
{
  attribute int ISBN;
  attribute list<int> ratings;
  attribute dictionary<int, int> chapterPageNumbers;
  relationship author writtenby inverse author::write;
};

class author extends person (extent authors key ID)
{
  attribute int ID;
  relationship set<book> write inverse book::writtenby;
  enum sex { MALE, FEMALE };
  attribute sex gender;
};

class person (extent people keys firstName,lastName)
{
  attribute string firstName;
  attribute string lastName;
};
```

Figure 6.1: Sample ODL Schema

The following rules for mapping ODL to XML and for mapping XML to ODL are extracted from multiple researches as addressed in the conclusion of this chapter.

Rule 1

O2X_R1. ***Base XML document structure*****:** As discussed in algorithm 3.3 section 3.5, XML document must have a root element. XML specifies that all elements within a document must be enclosed within a single root element which is equivalent to the root of the object-oriented hierarchy. This element is presenting the database name. An anonymous complexType element named choice is associated with the root element to define the cardinality or the maximum occurrence attribute specified as unbounded (XML constructor maxOccurs). This choice element will be referred to as

the root element. This is where the content of the database will be stored. The XML schema declaration of the root element (whose name may be arbitrary; we will use the name database here) will be referred as the element declaration, while the schema element will be referred to as the schema root. The additional rules will insert child elements in all of these places. Elements representing classes also have anonymous content models; they are inserted in the root element. The sequence element defining the content model of the class will be referred to as the class root.

Rule 2

O2X_R2. ***Top-level classes*****:** Each top-level class in the ODL schema is converted into an element in XML schema with the same name. These elements are included in the special choice element created by applying rule *O2X R1*. Within each element, the sequence constructor is created as a container to include the content of the class element. For each class in the ODL schema, a corresponding, like-named container is created, in which its attributes will be stored based on later rules. With the simplifying assumption stated next, only the key attributes are included to define class types, as an object in a class is entirely defined by the values of its key attributes. Finally, class types are implemented in the same manner as relationships (see rules *O2X R5* and *O2X R6*), also to simplify the implementation, n the ability to visually differentiate between attributes and relationships is lost.

One additional note to make is how to declare keyrefs for attributes whose type is a list of class type: the name of the keyref would then be given as classType.attribute.item.ref, where classType is the name of the declaring class and attribute is the name of the attribute. The selector XPath is then given the value ”./classType/attribute/item”. In short, the name of a keyref closely mirrors that of its

selector XPath expression.

Simplifying Assumption: ODL schemas allow for a key to be any set of properties that is defined for the class. This means that in addition to primitive type attributes, keys may also be complex types, classes, structures, unions, or even relationships. However, the implementation of any type beyond the primitive ones increases the complexity of the transformation process substantially. In this work, the simplifying assumption is adopted. This assumption allows to make progress in the implementation, while avoiding some tricky technical details beyond the scope of this research. The simplifying assumption can be stated as follows: All classes must have at least one key attribute, and every key must be of primitive or enumerated type.

Rule 3

O2X_R3. ***Attributes with primitive domains:*** This rule states that each basic datatype attribute is mapped into a like-named element within the container described in rule R2. In other words, the attributes specified in ODL with basic data types(string, short, date, float, etc.) are translated into atomic elements included in the corresponding special sequence element that has already been produced by rule *O2X R2*, and their data types should, if possible, be the same.

The handling of attributes of primitive type is accomplished as follows: an element is created with the same name as the corresponding attribute name, and the value of the type attribute for the new element is specified as the XSD analogue of its ODL type. Refer to Table 6.2 for list of type mapping [75].

ODL	XSD
Boolean	xsd:Boolean
Double	xsd:double
float	xsd:float
int	xsd:int
long	xsd:long
short xsd:short	short xsd:short
string xsd:string	string xsd:string
unsigned long	xsd:unsignedLong
unsigned short	xsd:unsignedShort

Table 6.2: Primitive Type Mapping

Rule 4

O2X_R4. ***Key attributes:***. The rule for converting key/keyref attributes is as follows: For each class, an element representing the key is added to the element declaration. Suppose the name of class type is defined as classType, so name of the key will be classType.key (Example person key name will looks as person.key). The value of the xpath attribute of the selector child element is given as "./classType".For each key attribute in the class, a field element is added as a child to the key element, with its xpath attribute given the name of the key attribute as its value. In Figure 6.1, two single-part keys, "ISBN", and "ID" are shown; the class person makes use of a compound key, using two strings, firstName and lastName.

Rule 5

O2X_R5. ***Multiple values specified by collection type (one-to-many relationship):*** Each relationship that contains any of the three keywords set, list or bag, is converted into an element with the same name. This element is included in the special sequence element already produced by rule *O2X R2*. As the cardinality of the relationship is greater than one, the attribute maxOccurs is set to unbounded. Further, this element should include:

- An anonymous complex data type with a special element complexContent.
- Special restricted element that contains an attribute with the name of the key attribute of the referred class.

Special keyref element has also to be declared as a subelement of the element that has been produced by rule *O2X R1*. This special element should also have a name attribute with its value specified as: class_name_which_the_relationship_belongs.relationship_name.ref. This special element should also include a refer attribute with the value: key_element_name_of_referred_class. The xpath attribute of the selector (sub)element belonging to this special element should contain a Xpath expression like: .//class_name_which_the_relationship_belongs_to/relationnship_name. The xpath attribute of the (sub)element field of this special element should contain a Xpath expression like: @key_attribute_name_of_the_referred_class. The example below explains these definitions.

```
<xsd:keyrefname="author.write.ref" refer="book.key">
        <xsd:selectorxpath="./author/write"/>
        <xsd:fieldxpath="@ISBN"/>
</xsd:keyref>
```

Rule 6

O2X R6. ***One-to-One relationships:*** One-to-One relationship means that the keywords set, list, or bag are not involved into relationship. An element with the same relationship name is created in sequence element defined in *O2X R2*.This element does not have maxOccurs attributes as the default value for this constructor is 1. A new like-named element is created within the class root for the relationship. This element includes an anonymous complex datatype. For each key attribute in the

related class, an attribute with the name of the related key attribute is added to the complexType, with its type matching that of the type of the related key attribute. Special keyref element has also to be declared as a subelement of the element that has been produced by rule *O2X R1*. In the declaration of the special keyref element, the xpath attribute of the selector element should contain a Xpath expression like: class_name_which_the_relationship_belongs_to. The xpath attribute of the field element should contain a Xpath expression like: **//relationship name**. If the name of the relationship is "relationship", the name of the declaring class is classType, and the name of the related class is relatedType, then the name of the keyref is classType.relationship.ref and the value of its refer attribute is relatedType.key. The xpath attribute of the selector element child is given as "./classType". For each key attribute in the class, a field element is added as a child to the key element. The xpath attribute therein is given the value "@attribute", where attribute is the name of the key attribute. In Figure 6.1, an example of both one-to-one and one-to-many relationships, namely "writtenby" of class "book", and "write" of class "author", respectively.

Rule 7

O2X_ R7. ***Lists:*** Each attribute of a class in ODL schema that contains the keyword "list" in its definition is converted into an element with the same name; the new element is included in the special sequence element that has resulted from rule O2X R2. This element should include an anonymous simple data type.

If an attribute is of a list type, could be specified as list, set, bag, sequence, or array, a new element is created with an anonymous complexType as its content model. This

complexType will have an unbounded number of child elements named item. This element type is then inserted into the class root. In Figure 6.1, the attribute “ratings” of the class “book” is of list type.

Rule 8

O2X_R8. ***Complex data type attributes:*** Each attribute with a complex data type specified in ODL is converted into an element with the same name. This element should be included in the special sequence element that has resulted from rule O2X R2. This special element should include an anonymous complexType with a special sequence element in which the elements corresponding to the conversion of the attributes integrated in the class (that defines the complexType of the attribute being converted)are declared.

Rule 9

O2X_R9. ***Enumerations:*** Each attribute of a class in ODL that contains the keyword enum in its definition is converted into an element in the special sequence element that has resulted from rule O2X R2. This element should include an anonymous simple data type and this simple data type should include a special restriction element with a base attribute. The restriction element should also include, for each one of the values of the enum data type, a special enumeration element with the same value in its value attribute.

Enumerated types may be declared on the same level of classes, within the class, or anonymously within the attribute. For each enumerated type, a simpleType restricting xsd:string is inserted into the schema root. Given the enumerated type enumType:

- If enumType is globally declared, the name enumType.enum is given

- If enumType is locally declared in class classType, the name classType.enumType.enum is given.
- If enumType is declared anonymously in attribute of class classType, the name classType.attribute.enum is given.

In Figure 6.1, enum “sex”, which is the type of the attribute “gender”, is an example of locally declared enumerated types in the class author.

Rule 10

O2X_R10. ***Dictionary Types:*** The object-oriented dictionary constructor allows a user to query a list of associations, searching for an entry that matches the search criteria. ODL defines a dictionary as an Association [15]. Association is a simple list of structure that defines a key and a value. Dictionary type transformation creates an element with an attribute as its name and an anonymous complexType as its content model. This anonymous complexType - named sequence - will have an unbounded number of children of elements named item. The content model sequence has two child elements named key and value.

The content models of the key and value elements depend on the key and value types of the dictionary; this is handled in a manner similar to that of lists. If the key and value types involve classes, keyrefs will also be added accordingly. In Figure 6.1, a dictionary attribute named chapterPageNumbers is defined as a dictionary type example that associates chapters with page numbers.

Rule 11

O2X_R11. ***Inheritance:*** Inheritance is one a main concepts of object-oriented model, so transformation process would not be complete without supporting this feature. Because the content models of classes are anonymously declared, XML Schema cannot be defined to include the equivalence of the extended classes. This

can be handled in a symmetrical manner of the approach discussed in Section 3.4.1 and in Section 3.4.2. Inheritance can be handled as the following:

If a class extends another class, then all inherited properties of the superclass are included in the subclass. As XML does not support explicit inheritance, it is interpreted as nesting. This can be handled in two ways based on the user preference. First, a nested XML structure may be produced where the element that corresponds to a subclass is extended to include all the inherited definitions of the superclass. Second, a flat XML structure using key/keyref can be produced as follows. The selector element in the superclass key element has its xpath attribute appended with ”—./subclass”, where subclass is the name of the subclass. Similarly, the selector element in any keyref element referencing the superclass keymust have its xpath attribute appended in the same way. This process is repeated for every superclass in the class hierarchy.

To illustrate the functionality of inheritance, in figure 6.1, the “person” class is inherited in the “author” class.

Rule 12

O2X_R12. ***Structures and Unions:*** Structured and union types are implemented relatively in straightforward manner. Classes that do not have an extent, have an extent but no key, or violating the simplifying assumption stated in O2X_R2 are to be considered in a future extension of this work.

Structure is mapped into complexTypes that will be added to the schema root, with a naming scheme similar to that of enumerated types, but with the “.struct” extension. These complexTypes have a sequence element that contains the structure members. The above rules apply to related members.

As structures, union is also implemented as complexTypes. The union contains a discriminator member that is sharing the name of the union type. This discriminator must be of primitive or enumerated type, which will be the first sequence child. The second child is a choice element that will contain each member of the union. Again, union type elements are added to the schema root with a naming scheme similar to that of enumerated types, but with the ".union" extension. Similar to structures and enumerated types, attributes of union type can then be coded in the same way as primitive types.

There may be no full implementation due to the cyclic dependency problem. The cyclic dependency problem is caused when a structure has a member that may declare a member of the original structure type. In this case, the transformation process may not terminate. In fact, the simplifying assumption in O2X_R2 was introduced as a response to tackle the cyclic dependency.

Rule 13

O2X_R13. ***Times, Intervals, and Time-stamps:*** ODL has a way of codifying dates, times, intervals, and timestamps, using the structured types date, time, interval, and timestamp, as well as their corresponding class type counterparts Date, Time, Interval, and Timestamp [6]. These types can be treated specially by converting them into the XSD types xsd:date, xsd:time, xsd:duration, and xsd:datetime, respectively. However, a proper translation between these two types must be defined, owing to the possible fine-print differences between them.

Rule 14

O2X_R14. ***Null Objects:*** ODL does not define that the property can take the value of null. However, the implementation of null objects is discussed. Elements declaring

classes must set the nillable attribute to true. Elements representing attributes of list and dictionary type attributes must also set the nillable attribute of their item, key, or value children, In addition, elements representing relationships and attributes of class type must set the use attribute of each attribute child to optional.

Note that there is no way in XML Schema to enforce attributes to be required, optional, or prohibited: For example, if the lastName variable is set to null, then the XML Schema can be translated as: <xsd:element name="lastName" type="xsd:string" nillable="true"/>, while the XML document can be set as: `<lastNamexsi:nil="true"></lastName>`.

Rule 15

O2X_R15. ***Interfaces:*** Though interfaces are normally ignored because they are defining only the behavior of an object (which would be lost in the XML transformation), interfaces may also declare enumerated types, structures, and unions, which can then be used by any class implementing any of the mentioned types. Further exploration of this is needed.

As a result of applying the rules described above to the sample ODL described in figure 6.1, Appendix F includes the generated XML Schema. This process is repeated against many ODL example inputs and each of these has generated the corresponding XML Schema where the results are validated by an XML Schema validator.

6.5 Rules for Converting XML Schema into ODL

XML schema core components are elements and attributes; elements may be nested or presented in a flat structure that simulates the nesting using key/keyref constructs. The rules presented in this section will produce an ODL structure from the input flat or nested XML Schema. As ODL allows for inheritance, which is not part of XML by definition, there are two options; either to produce a one level class hierarchy where all classes are under the root or to produce a multi-level class hierarchy by applying an optimization rule on the one level hierarchy to minimize overlap and redundancy between the classes. Conversion of flat XML by using key/keyref constructs will produce a class hierarchy that incorporates nesting in a more natural way. Next is a summary of rules required for the conversion of XML Schema into the corresponding ODL.

Rule 1

X2O_R1. ***Create the main classes:*** Locate the XML Schema first level of element. For each first level element create a corresponding main class with the same name as the element.

Rule 2

X2O_R2. ***Create the nested classes:*** XMLSchema allows the definition of flat and nesting. Each element E that is mapped into a class c in X2O_R1 will be recursively checked for all nesting elements that are represented by nesting or a flat structure that is using key/keyrefs. For each element E for which a class c has been created, apply the following two sub rules.

X2O_2.1. ***Nested classes from nested XML schema***: If element E has a subelement E_s such that E_s has complex structure (includes some subelements) then create a new class c_s corresponding to E_s.

X2O_R2.2. ***Nested classes from flat XML schema:*** If element E has keyref specification which references another element E_k then add to the definition of class c a new non-primitive attribute with the name A_{Ek} and specify the domain of A_{Ek} as the class that corresponds to element E_k; create the latter class if it does not exist.

Rule 3

X2O_R3. ***Define primitive attributes for simple elements:*** For each simple element found in any of the already processed elements E, create in the corresponding class c a primitive attribute with the same name and domain as the simple element.

Rule 4

X2O_R4. ***Define primitive attributes for attributes in elements:*** For each attribute found in any of the already processed elements E, create in the corresponding class c a primitive attribute with the same name and domain as the attribute in E.

Rule 5

X2O_R5. ***Define the keys in classes:*** If element E has a key that is composed of one or more attributes/elements then create a key for the corresponding class c; the keyfor class c must include all the attributes that correspond to the components of the key of E.

Rule 6

X2O_R6. ***Define non-primitive attributes in classes:*** Both XML and ODL allow for complex domains to be specified. Below are rules of mapping for some

complexType domains.

X2O_R6.1. ***Relationships:*** For every element representing a relationship type inside element E, create a corresponding attribute in class c that corresponds to E. The mapped relationship type attribute should be set equivalent to the definition of the corresponding element in E (to an existing class) and the linking attribute should be determined.

X2O_R6.2. ***Collections:*** Both simple and complex domains may be allowed to have collections (set, list, bag or array) as the value; this may be found explicitly specified in XML using "maxOccurs". Also relationships may be specified to connect to a collection of values. For all these cases, determine the collection type from the definition of the corresponding element and reflect that into the definition of the attribute or relationship in the given class.

X2O_R6.3. ***Enumeration:*** Locate all specifications of enumerated type in the XML schema. Each of these enumerated types is defined with an element E; accordingly, locate the corresponding class c and define in class c the same enumerated type found in E.

X2O_R6.4. ***Dictionary:*** For elements specified to have the dictionary type in the XML schema, find the components of the dictionary type and define it accordingly for the corresponding attributes in the ODL schema.

Rule 7

X2O_R7. ***Decide on attributes that may have null value:*** For every attribute A in the ODL schema, if the element that corresponds to A has its "minOccurs" specified as "0" then attribute A is allowed to have the value "null" and this should be reflected into the definition of attribute A in ODL.

Rule 8

X2O_R8. ***Define interfaces in classes:*** Encapsulation is one of the paradigm's object-oriented paradigm main features, so for each attribute A defined in ODL, define two methods get() and set(X); these methods will have very simple code; the former will display the value of attribute A in the receiving object and the latter will replace in the receiving object the existing value of A with X.

Enforcing Inheritance into the Hierarchy

XML to ODL generates a one level hierarchy, so the inheritance is not explicitly defined. This means that both superclasses and subclasses are not defined. However, the inheritance information is implicitly present in the hierarchy and simulated by duplication of information in various classes. Hence, the target is to enforce reusability instead of duplication; this is possible by deriving a list of superclasses for each given class c to increase inheritance instead of duplication.

Non-inherited characteristics of class c include both locally defined attributes and locally defined behavior, denoted $L_{attributes(c)}$ and $L_{behavior(c)}$, respectively. The inherited characteristics of class c include both all attributes and behavior in the superclasses of class C, denoted $(W_{attributes(c)} - L_{attributes(c)})$ and $(W_{behavior(c)} - L_{behavior(c)})$, respectively. To maximize inherited and hence minimize locally defined characteristics, it is required to adjust the list of superclasses of class c to include classes that maximize inherited and minimize locally defined characteristics.

Rules are defined to add some existing classes to the list of superclasses of class c, denoted Cp(c), or by pushing class c into the list of superclasses of some existing classes. This will be straightforward to find the overlap between classes.

Rule 9

X2O_R9. ***Find potential superclasses:*** For each pair of classes c1 and c2 produced as the result of applying rules X2O_R1 to X2O_R8, if $L_{attributes}(c1) \cap L_{attributes}(c2)$ is not empty then create a new class c1_2 to include all attributes in $L_{attributes}(c1) \cap L_{attributes}(c2)$, Further, $L_{behavior}(c1\_2)$ is set to include $L_{behavior}(c1) \cap L_{behavior}(c2)$ as every class has in its local behaviour only pairs of methods, one pair per attribute to set/modify and get/return the value of the attribute.

6.6 Rules for Data Conversion

The XML document to object database data mapping process involves two main rules. First, one object (root element) is generated for each root element (object) and attributes (elements) with primitive domains are mapped directly. Second, all complex domains are mapped after their containers are ready to hold them. The object database to XML document mapping is performed in one pass to produce instantiations for elements with values for their primitive and non-primitive components specified. Next are the rules for mapping XML documents into object data and the object data into instantiation of elements (XML document).

X2O_D_R1. ***Creating objects from documents:*** In XML to ODL mapping rules, class c is created for every element E of complexType. For each instantiation I_E of element E, create a corresponding object O_c in the local instances of class c, denoted

$L_{instances}(c)$; object o_c will have a new unique identifier and the values of its primitive attributes will be taken from the corresponding instantiation I_E.

X2O_D_R2. ***Deciding on values of non-primitive attributes for objects:*** This rule uses the object identifiers created by rule X2O_D_R1 to specify the values of non-primitive attributes. For every object o_c which has been created by rule X2O_D_R1, if class c of object o_c has non-primitive attributes then for each non-primitive attribute A in class c, locate object(s) o_I that corresponds to the instantiation of the element that corresponds to attribute A and set the value of attribute A in object o_c to object(s) o_I. The latter value may be single or collection of values; for the collection of values case, the values o_I are arranged to fit the specification of the collection type whether set, list, bag or array.

O2X_D_R1. ***Creating documents from objects:*** For each element E, locates its corresponding class c. Some subelements from E may have their definition exists in superclass(es) of c because of the inheritance characteristic specified for class c but missing in XML. If this is the case, then start with elements E that correspond to classes c with no subclasses; produce one instantiation of E for each object o_c in c; each instantiation will utilize from o_c in c the values that are part of o_c due to the inheritance relationship between class c and its superclasses. For each non-leaf class c_L which has a corresponding element E_L, if c_L has some objects (in $L_{instances}(c_L)$) then for each object in $L_{instances}(c_L)$ create a corresponding instantiation for element E_L.

6.7 Implementation Details

After the description of transformation rules between ODL and XML for both schemas and data, a reference implementation is written using the Java language. This reference implementation contains a lexer and parser to accept input, as well as a transformer to do the actual XML and ODL transformations. The parser creates a set of objects representing ODL classes, attributes, relationships, and enumerated types; it is also capable of producing XML schemas and XML documents; it raises an error if the simplifying assumption is violated. The parser parses all parts of ODL and XML schemas that corresponds to rules specified in Section 6.3 and Section 6.4. The transformation process is described in ODLTransformer and XMLTransformer classes. As each class process is a reverse for the other class process, only ODL to XML transformation process specified in the ODLTransformer class is described.

After lexing and parsing the input ODL file, the ODL object representing the ODL schema, an object of type ODLSchema, is passed into the ODLTransformer, which proceeds to build a DOM tree. This DOM tree can then be further manipulated using XSL Transformations. In this case, it is chosen to output the DOM tree as-is, which then is our output XML Schema file.

ODLTransformer depends on a set of ItemBuilders to create the content models of each class type. There are separate ItemBuilders for primitive types, enumerated types, class types (which are also used in relationships), list types, and dictionary types. These ItemBuilders will also create the necessary keyrefs, though ODLTransformer itself holds the responsibility of creating keys as well as simpleTypes representing enumerated types.

6.8 Conclusion

The need to convert between the object-oriented database model and the XML database model has been covered in this work. It is eventually more attractive than the XML to relational conversion because the two models covered have more commonalities and hence the semantic gap almost disappears to the benefit of the transformation process both ways. The mapping from ODL to XML involves the conversion between an object schema, in the form of ODL, to an XML schema, using XML Schema, and then the conversion of the data itself following the schemas that have been created. The conversion of an object-oriented schema to an XML schema is a nontrivial process, requiring both human intervention and automated tools. Though we may make rules representing general guidelines on how to covert between ODL and XSD, they can only cover a limited subset of the ODL grammar. However, these rules cover a sufficiently large portion of ODL so that human intervention is kept to a minimum. Finally, the converted XML schema may also impose conditions on how the data is to be transformed as well, as with the case with date and time data types. The conversion from XML to ODL is more challenging as it is required to decide on the inheritance for ODL; it could successfully handle this vital property and requirement by invoking an optimization process that minimizes the overlap between classes, redundancy in other words. It is argued that the rules that we have defined should cover the vast majority of the needs of most object-oriented database users and administrators.

This work is a result of researchers collaboration listed in [40]. Our contribution to this research is covering the concepts of the mapping between OODB into XML. This includes: a) using a similar deductive approach to extract the meta-data from the

database if not exists, b) using an intermediate object graph in the mapping process, c) Using the approach of a generic schema so as to deal with any object oriented database. Also, the contribution covers major involvement into XML into ODL conversion rules (Rule 1 to Rule 6). Further, we have little involvement into rules (Rule 1 to Rule 4 and Rule 8) for conversion from ODL to XML.

Chapter 7

Conclusion and Future Work

7.1 Introduction

In Chapter 3 of this thesis, a process for extracting XML Schema and corresponding XML document from an object-oriented database is presented. This process maps a generic OODB into an object graph (OG), and then the OG is used to generate the required XML Schema and related document. A reverse process is also presented in Chapter 4 to store XML Schema and corresponding document into an object-oriented database. OG is used as a mean for the mapping process. Chapter 5 discusses the implementation of extracting XML schema and corresponding document from an object oriented database. Also, it discusses a framework mapper called COODaX. This mapper extracts the object-oriented schema meta-data from the object-oriented database. When meta-data does not exist, the system tries to construct it from the entire data. Chapter 6 explains another way of mapping between ODL and XML. This work structures the rules required to map different components of ODL and XML Schema.

7.2 Research Contribution and Benefits

As a very little work had been performed on mapping between XML and object-oriented database, this research fills a gap that had not been addressed. This work introduces a novel approach (theoretical back up) that have been performed on providing a sound basis for extracting XML Schema and corresponding XML

document from an object-oriented database (although some OODBMS provide XML export facility). As a very little work had been performed on mapping between XML and object-oriented database, this research fills a gap that had not been addressed. This work introduces a novel approach (theoretical back up) that have been performed on providing a sound basis for extracting XML Schema and corresponding XML document from an object-oriented database (although some OODBMS provide XML export facility). In this research, we tried answering the research questions that had been raised before starting this work. In the first start, we define a generic object oriented database that can be a subset of most of object oriented databases. The main component of this proposed database that may hold data and relations is the class, so a class explained in Definition 3.1 is used as a core component in the mapping process. The class primitive attributes, non-primitive attributes (instances), inherited attributes, super classes, and the object identifiers (OID) are the components that either extracted from the database or used for writing data into the database. Also, we successfully perform a research of two way mapping between a generic object oriented database and XML Schema. Corresponding data is also handled. The mapping process successfully uses an object graph (OG) as a means for the mapping. The mapping process is able to handle and generate both nesting and flat XML schemas and related documents and is able to differentiate between inheritance and nesting. This is represented by the work described in chapter 3 and chapter 4. The user involvement in mapping OODB into XML is very minimal if exists because the introduced framework COODaX can extract the meta-data either by reading the database catalogue or by analysing the content of the database. If the catalogue data is not sufficient then the content analysis could build the missing

information. The user involvement for mapping XML into OODB is required because as XML does not support inheritance, the is no way to differentiate between components that represent nesting and components that represent inheritance. The best way to reduce the user involvement is to map all XML complex components as OODB nested attributes. All data can then be scanned and analysed and a heuristic approaches can be applied to segregate between inheritance and nesting. Also, a GUI interface for both XML and generated OODB presentations can simplify the human intervention.

A two-way mapping between ODL and XML is also introduced. The mapping from XML to ODL is newly introduced, while the mapping from ODL to XML added some improvement to previous research performed. The mapping between XML and OODB (using OG) proves that this approach is successfully applied into XML and ODL mapping. Section 6.2 discusses in details the overlap between the two different mappings.

The main benefits gained from the outcomes of this study within the context of new emerging technologies are based on the mapping approach provided in this research to facilitate the availability of XML data.. The most notable attractions of XML is its ability to provide an ideal solution as platform independent. With the widespread of next generation technologies including web services, semantic web, web of things, cloud computing, etc., researchers and users will be working without any need to consider the underlying platform or to stick to a single platform and hence limit their productivity and the availability of their products whether services or platforms. Structured data whether relational or object-oriented could maximize the availability

and sharing of their content by using our approach to convert into XML and vice versa.

7.3 Theory Contribution

Thorough investigation is performed about the conducted research on all types of mapping between XML and different types of databases. The investigation clearly showed that little research is accomplished on mapping between XML and OODB.

This work introduces a two-way mapping between XML and OODB. Further, a framework called COODaX is wrapping these mapping processes, and different algorithms are created for handling different components. The mapping process from OODB into XML can be summarized as:

- Set of algorithms are created to perform this process.
- A directed object graph (OG) is constructed from the object oriented database schema.
- OODB schema meta-data is extracted either by reading the database catalog or by analysing the content of the database. If the database catalog is not present or missing some information, the content analysis could recover the missing meta-data..
- Inheritance and nesting are presented properly in the OG.
- Both nested and flat XML Schemas are constructed from the OG.
- The corresponding XML document is generated from the database.

The mapping from XML into OODB can be summarized as:

- Set of algorithms are created to perform this process.

- A directed object graph (OG) is constructed from the XML Schema.
- OODB Schema is constructed from the OG.
- The corresponding XML document is stored into the database.

Also, another way of mapping between ODL and XML is introduced. Different rules found in the literature for mapping from ODL to XML are improved, and the mapping rules from XML to ODL are constructed.

7.4 Practical Contributions

This research work was performed using two different modes of practical implementation. The first implementation is performed using a Customised object-oriented database composed of Java classes. The second implementation used db4o object-oriented database where a set of Java programs are written for achieving this task. The UNIVERSITY object-oriented database example introduced in Chapter 3 is used for the implementation. Hence, the practical contributions can be listed as the following:

- Object-oriented database schema information of UNIVERSITY example is extracted.
- Flat XML schemas for UNIVERSITY example is generated based on the extracted database schema.
- Nested XML schema is generated.
- Flat and nested corresponding XML documents are generated.

Generated outputs are verified against the algorithms defined in Chapters 3 and 4. The generated outputs are listed in the appendixes.

7.5 Lessons Learned from the Research Journey

I had started my research journey by investigating about the major components of the object-oriented database schema and the structure of XML Schema. As there was very limited research and published work about mapping between object-oriented and XML, I had started exploring the research conducted on relational or object-relational and XML mapping. This research was mostly about mapping between relational or object relational and DTD Schema but not using the XML Schema. Most of this research was concerned about how to map the two different model structures of XML and relational databases. As the two models have different structures, the mapping was trying to fit the tree-like XML structure into the linear relational structure; in other words, trying to resolve the impedance mismatch problem.

The challenge was that; we have to initiate the mapping between object-oriented and XML Schema without having any previous work. Concerns are raised up on trying to make smooth and proper mapping between the basic object-oriented database component (class) and the complex types of the XML Schema. This implies to differentiate between inherited, local, complex, and primitive attributes. Hence, we learn how to map the nesting caused by inheritance and the natural nesting of complex domain type. Also, we learn how to map flat and nested schemas.

7.6 Limitations and Directions for Future Research

Although the research conducted for this work is properly handling the mapping between object-oriented databases and XML, these are some limitations encountered on this work.

- The mapping between XML and OODB is taking a subset of the XML Schema and OODB schema structures. For instance limited number of primitive attributes such as integer, string, etc. is handled. Also as an example, mapping of lists, bag, set, and arrays are not handled.
- Attributes are presented as simple elements inside their type element. This option is more natural but this may cause losing the indication that the attribute element is an attribute and not a normal element. This causes that the initial structure of the XML document may not be exactly retrieved

There are some issues that can be addressed for the future work.

- As XML structure does not support inheritance and nesting, so during the mapping from XML to OODB, there is no way to tell if an XML attribute is mapped as nesting or inheritance database attribute. The way to tell in this research is user invovment. Therefore, a heuristic approach could be applied to minimize the user involvement in defining inheritance and nesting. The proposed idea is to make a full database scan. Out of this process and through the comparisons between classes attributes values, a process can infer and segregate between inheritance and nesting.
- The conducted research for this work can be extended and developed to have mapping for a wider range of structures of both XML and OODB schemas. This may include lists, arrays, and dictionaries.
- Mapping from XML to OODB is based on XML Schema. When schema does not exist, some available tools such as Microsoft Visual Studio XML Editor or Microsoft XSD Inference can be incorporated into the process to generate the schema before performing the mapping process.

- COODaX could be implemented with a GUI interface. The process of extracting OG from the database and the generated graph can be exposed into the system. Further, a native XML query can be incorporated into this framework. Also, the system could be adjusted to generate a subset of the schema when required.

7.7 Epilogue

This thesis has demonstrated a novel approach for mapping between XML and a generic object-oriented database using directed object graph. The experience gained from this work showed that the sematic gap between XML and object-oriented structures is minimal if exists. This reflects positively on smooth and natural mapping between XML and OODB compared to other databases structures such as relational databases. Nested and flat schemas are handled and the differentiation between inherited and non-primitive attributes is considered. Implementation is performed using two different object-oriented databases and the results produced are similar. Another way of mapping between ODL and XML is introduced as well and different rules are introduced.

References

[1] M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. "The Object-Oriented Database System Manifesto"**.***In Proceedings of the First International Conference on Deductive and Object-Oriented Databases,* pages 223-40, Kyoto, Japan, December 1989.

[2] U. Ahmad, M. Hassan, A. Ali, R. McClatchey, and I. Willers. "An Integrated Approach for Extraction of Objects from XML and Transformation to Heterogeneous Object Oriented Databases". InProceedings of the Fifth International Conference on Enterprise Information Systems (ICEIS), pp 445-449, 2003.

[3] Reda Alhajj. "Extracting the Extended Entity-Relationship Model from legacy Relational Database". Information Systems, Vol. 28, No. 6, pp.597-618, 2003.

[4] S. Abiteboul, D. Quass, J. McHugh, J. Widom, J. Wiener, and J. Widom. "The Lorel Query Language for Semistructured Data". International Journal. on Digital Libraries (IJDL), 1(1):68-88, Apr. 1997.

[5] T. Atwood, "An Object-Oriented DBMS for Design Support Applications". In Proceedings of the IEEE COMPINT 85, pp 299-307, September 1985.

[6] D. Barry. "Object Oriented Database Management Systems". [Online] available at:
http://www.service-architecture.com/object-oriented-databases/articles/odbms_faq.html

[7] A. Berglund, S. Boag, D. Chamberlin M. Fernandez, M. Kay, J. Robie, J. Simeon. "XML Path Language (XPath) 2.0". W3C Recommendation, January 2007. [Online] available at: http://www.w3.org/TR/xpath20/.

[8] J. Bosak, T. Bray, D. Connolly, E. Maler, G. Nicol, C. M. Sperberg-McQueen, L. Wood, J. Clark, "W3C XML Specification DTD", 1998. [Online] available at: http://www.w3.org/XML/1998/06/xmlspec-report-19980910.htm.

[9] S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robie, J. Simeon. "XQuery 1.0: An XML Query Language". W3C Recommendation, January 2007. [Online] available at: http://www.w3.org/TR/2007/REC-xquery-20070123/.

[10] G. Booch, M. Christerson, M. Fuchs, and J. Koistinen. "UML for XML Schema mapping specification", Rational Software Corp. and CommerceOne Inc., 1999. [Online] available at: http://xml.coverpages.org/fuchs-uml_xmlschema33.pdf.

[11] P. Bohannon, J. Freire, P. Roy, and J. Siméon. "From XML Schema to relations: a cost-based approach to XML storage". In Proceedings of the 18th IEEE International on Data Engineering, February 2002.

[12] T. Bray, D. Hollander, A. Layman, R. Tobin. "Namespaces in XML 1.0 (Second Edition) ". W3C Recommendation, August 2006. [Online] available at: http://www.w3.org/TR/2006/REC-xml-names-20060816.

[13] A. Bonifati and D. Lee. "Technical survey of XML schema and query languages". Technical report, July 2001. [Online] available at: http://www.csd.uoc.gr/~hy561/Data/Papers/surveyxmlschema-querylanguages.pdf .

[14] R. Bourret. Mapping DTDs to Databases, Mapping XML and Databases, 1999. [Online] available at: http://www.rpbourret.com/xml/DTDToDatabase.htm,

http://www.rpbourret.com/xml/XMLAndDatabases.htm.

[15] R.G.G. Cattel, D. Barry, M. Berler, J. Eastman, D. Jordan, C. Russell, O. Schadow, T. Stanienda, F. Velez. “The Object Database Standard: ODMG-93”. Morgan Kaufmann, 1994.

[16] D. Chamberlin, J. Clark, D. Florescu, J. Robie, J. Siméon, M. Stefanescu. “XQuery 1.0: An XML Query Language”. W3C Working Draft, June 2001. [Online] available at: http://www.w3.org/TR/2001/WD-xquery-20010607/.

[17] M. J. Carey, D. J. DeWitt, M. J. Franklin, N. E. Hall, M. L. McAuliffe, J. F. Naughton, D. T. Schuh, M. H. Solomon, C. K. Tan, O. G. Tsatalos, S. J. White, and M. J. Zwilling. “Shoring up Persistent Applications”. In Proceedings of Management of Data ACM SIGMOD Conference, ACM SIGMOD Record, Volume 23, issue 2, pp 383–394, 1994.

[18] J. Clark and S. J. DeRose (Eds). “XML Path Language (XPath) Version 1.0”. W3C Recommendation, April. 1999. [Online] available at: http:///www.w3.org/TR/xpath.

[19] M. Carey, D. Florescu, Z. Ives, Y. Lu, J. Schanmugasundaram, E. Shekita, and S. Subramanian. “XPERANTO: Publishing object-relational data as XML”. In Proceedings of the Third International Workshop on the Web and Databases (WebDB), pp 105-110, May 2000.

[20] Tae-Sun Chung, Sangwon Park, Sang-Young Han, and Hyoung-Joo Kim. “Extracting Object-Oriented Database Schemas from XML DTDS Using Inheritance”. In Proceedings of the Second International Conference on Electronic Commerce and Web Technologies (EC-WEB), pp 49-59, 2001.

[21] William. Cook and Siddhartha Rai. “Safe query objects: statically typed objects as remotely executable queries”. In Proceedings of the 27th International Conference on Software Engineering (ICSE), pages 97-106. ACM, 2005. [Online] available at: http://www.cs.utexas.edu/~wcook/papers/SafeQuery05/SafeQueryFinal.pdf.

[22] William Cook and Carl Rosenberger.“ Native Queries for Persistent Objects, A Design White Paper ”. February, 2006. [Online] available at: http://www.cs.utexas.edu/users/wcook/papers/NativeQueries/NativeQueries8-23-05.pdf.

[23] D. Chamberlin, J. Robie, and D. Florescu. “Quilt: An XML Query Language for Heterogeneous Data Sources”. In Third International Workshop on the Web and Databases (WebDB), May 2000.

[24] J. Cheng and J. Xu. XML and DB2. In Proceedings of the 16th IEEE International Conference on Data Engineering, pp 569-573, 2000.

[25] A. Deutsch , M. Fernandez , D. Florescu, A. Levy , D. Suciu “XML-QL: A Query Language for XML”. W3C Recommendation, August 1998. [Online] available at: http://www.w3.org/TR/NOTE-xml-ql/.

[26] Ole Johan Dahl and Kristen Nygaard.“Class and Subclass Declarations”. In IFIP TC 2 Working Conference on Simulation Languages, NCC Documents, March, 1967.

[27] R. Elmasri and S. B. Navathe. “Fundamental of Database Systems, Fourth Edition”. Addison-Wesley, 2003, ISBN: 0321122267.

[28] Joseph Fong, and San Kuen Cheung. “Translating relational schema into XML Schema definition with data semantic preservation and XSD graph”. Information and Software Technology, Volume 47, Issue 7, pp 437-462, May 2005.

[29] D. Florescu, G. Graefe, G. Moerkotte, H. Pirahesh, and H. Schning. "Panel: XML data management: Go Native or spruce up Relational Systems?". In Proceedings of the ACM SIGMOD International Conference on Management of Data", May 2001.

[30] D. Florescu and D. Kossman. "Storing and querying XML data using an RDBMS". In IEEE Data Engineering Bulletin, volume 22, pp 27-34, September 1999.

[31] M. Fernandez, A. Morishima, D. Suciu, and W. Tan. "Publishing relational data in XML: The SilkRoute approach". In IEEE Data Engineering Bulletin, volume 24, pp 12-19, 2001.

[32] J. Fong, F. Pang, and C. Bloor. "Converting Relational Database into XML Document". In Proceedings of the 12th International Workshop on Database and Expert Systems Applications, pp61-65, 2001.

[33] M. Fernandez, W. Tan, and D. Suciu. "SilkRoute: Trading between relational and XML". In Proceedings of the 9th International World Wide Web Conference (WWW), volume 33, pp 723-745, 2000.

[34] Gemestone Object Oriented Database. [Online] available at: http://www.gemtone.com.

[35] Charles F. Goldfarb. The SGML Handbook. OxfordUniversityPress, USA, 1991.

[36] Hyper Text Markup Language (HTML) 4.01, W3C Recommendation. [Online] available at : http://www.w3.org/TR/html4, December 1999.

[37] H. Jagadish et al. (The Timber Team, University of Michigan). “Timber”, 2000. [Online] available at: http://www.eecs.umich.edu/db/timber.

[38] H. V. Jagadish, S. Al-Khalifa, A. Chapman, L. Lakshmanan, A. Nierman, S. Paparizos, J. Patel, D. Srivastava, N. Wiwatwattana, Y. Wu, C. Yu. “Timber: A Native XML Database”. The VLDB Journal, Volume 11, Number 4, Springer, December, 2002. [Online] available at: www.eecs.umich.edu/db/timber/files/timber.pdf

[39] Jarada T.N., Chung K., Shimoon A., Karampelas P., Alhajj R. and Rokne J., “Mapping Rulesfor Converting from ODL to XML Schemas”. In Proceedings of International Conference onInformation Integration and Web-based Applications & Services, Paris, Nov. 2010.

[40] T. N. Jarada, A. M. Elsheikh, T. Naser, K. Chung, A. Shimoon, P. Karampelas, J. Rokne, M. Ridley and R. Alhajj, “Rules for Effective Mapping between two Data Environments: Object Database Language and XML”. In Recent Trends in Information Reuse and Integration. Springer-Verlag, 2011.

[41] Tony Johansson and Richard Heggbrenna. “Importing XML Schema into an Object-Oriented Database Mediator System”. In Uppsala Master's Theses in Computing Science no. 260 Examensarbete DV3 20 p, ISSN 1100-1836, 2003.

[42] Michael Kay. “XSL Transformations (XSLT) Version 2.0”. W3C Recommendation, January 2007. [Online] available at: http://www.w3.org/TR/2007/REC-xslt20-20070123/.

[43] Alan Kay. “The Early History of Smalltalk”, 1993. [Online] available at: http://www.smalltalk.org/downloads/papers/SmalltalkHistoryHOPL.pdf. http://www.smalltalk.org/people/alankay.html

[44] W. Kim, J. F. Garza, N. Ballou, and D. Woelk. “Architecture of the ORION Next-Generation Database System”. IEEE Transactions on Knowledge and Data Engineering, Volume 2, March 1990.

[45] Kim, Won. “Introduction to Object-Oriented Databases”. The MIT Press, 1990. ISBN 0-262-11124-1.

[46] C. Liu, M. W. Vincent, J. Liu, and M. Guo. “A Virtual XML Database Engine for Relational Database”, SpringerLink, pp 37-51, Volume 2824., 2003.

[47] I. Manolescu, D. Florescu, and D. Kossmann. “Answering XML queries over heterogeneous data sources”. In Proceedings of the 27th International Conference on VLDB, Morgan Kaufmann, pp 241-250, 2001.

[48] Gregory McFarland, Andres Rudmik, and David Lange. “Object-Oriented Database Management Systems Revisited”. Contract Number SP0700-98-4000, DoD Data & Analysis Center for Software (DACS), Dec. 1997.

[49] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. “Lore: A Database Management System for Semi-structured Data”. SIGMOD Record, 26(3): pp54–66, September 1997. [Online] available at: http://infolab.stanford.edu/lore/pubs/lore97.pdf.
http://www-db.stanford.edu/lore

[50] M. Mani, D. Lee, and R. R. Muntz “Semantic data modeling using XML schemas”. In the 20th International Conference on Conceptual Modeling (ER), pp 149-163, Springer, November 2001.

[51] Taher Naser, Reda AlHajj, and Mick J. Ridley. “Reengineering XML into Object-Oriented Database”. IEEE International Conference on Information Reuse and Integration (IRI-2008), July 2008.

[52] Taher Naser, Reda AlHajj, and Mick J. Ridley “Flexible approach for representing object-oriented databases in XML format ”. Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services. ACM SIGWEB, pp 430-433, 2008.

[53] Taher Naser, Reda AlHajj, and Mick J. Ridle. “Two-Way Mapping between Object-Oriented Databases and XML”. Informatica 33, pp 297–308, 2009.

[54] Taher Naser, Keivan Kianmehr, Reda AlHajj, and Mick J. Ridley. “Transforming Object-Oriented Database into XML”. In IEEE International Conference on Information Reuse and Integration (IRI-2007), August 2007.

[55] Objectivity Object Oriented Database. [Online] available at: http://www.objectivity.com.

[56] Oracle Database 11g XML DB Technical Overview, Started in Oracle Database 9i Release 2, 2002. [Online] available at: http://www.oracle.com/technology/tech/xml/xmldb/Current/xmldb_11g_twp.pdf.

[57] Oracle Database 11g, 2009. [Online] available at: http://www.oracle.com/technology/products/database/oracle11g/index.html.

[58] Jim Paterson, Stefan Edlich, Henrik Hörning, and Reidar Hörning “The Definitive Guide to db4o”, Apress, 2006.

[59] Dave Raggett, Arnaud Le Hors, Ian Jacobs. HTML 4.0 Specification, W3C Recommendation, 1999. [Online] available at: http://www.w3.org/TR/html4/.

[60] Tore Risch. “AMOS II Active Mediators for Information Integration”, 2000. Uppsala Database Laboratory, 2001. [Online] available at: http://user.it.uu.se/~udbl/amos/amoswhite.html.

[61] K. Runapongsa and J. M. Patel. “Storing and querying XML data in object-relational DBMSs”. 8thInternational Conference on Extending Database Technology (EDBT) XML-Based Data Management (XMLDB) Workshop, March 2002. [Online] available at: http://gear.kku.ac.th/~krunapon/research/pub/xorator.pdf

[62] SAX Project Team. "*Simple API for XML (*SAX 2.0)". January 2002. [Online] available at: www.saxproject.org.

[63] Herbert Schildt. “Java 2: The Complete Reference, Fifth Edition”. McGraw-Hill Osborne Media, 2002, ISBN: 0072224207.

[64] Albrecht Schmidt, Martin Kersten, Menzo Windhouwer, and Florian Waas. “Efficient Relational Storage and Retrieval of XML Documents”. The Third International Workshop WebDB 2000 on The World Wide Web and Databases, May 2000.

[65] de Sousa, A., Pereiera, J., Carvalho, J., “Mapping Rules to Convert from ODL to XML SCHEMA”. In Proceedings of 22nd International Conference of the Chilean Computer Science Society, IEEE Computer Society, 2002.

[66] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. “Relational Databases for Querying XML Documents: Limitations and Opportunities”. In Proceedings of the 25th International Conference on Very Large Data Bases, pp 302–314, September 1999.

[67] H. Schoning and J. Wasch. “Tamino - An Internet Database System”. In proceedings of the 7th International Conference on Extending Database Technology (EDBT), 2000. [Online] available at: ftp://ftp.cse.buffalo.edu/users/azhang/disc/disc01/cd1/out/papers/edbt/taminoaninternehaj.pdf.

[68] T. Shimura, M. Yoshikawa, and S. Uemura. “Storage and Retrieval of XML Documents Using Object-Relational Databases”. In the 10thInternational Conference on Database and Expert Systems Applications, pp 206–217, September 1999.

[69] “Tamino: The Native XML Management System”, 2003. [Online] available at: http://www.softwareag.com/tamino.

[70] D. Toth and M. Valenta, “Using Object And Object-Oriented Technologies for XML-native Database Systems”. In Proceedings of the Dateso Annual International Workshop on Databases, Texts, Specifications and Objects, 2006.

[71] Versatile Object Oriented Database. [Online] available at: http://www.versant.com/.

[72] C. Wang, A. Lo, R. Alhajj, and K. Barker. “Converting legacy relational database into XML database through reverse engineering”. In the 6th International Conference on Enterprise Information Systems (ICEIS), pp 216-221, April 2004.

[73] Chunyan Wang, Reda Alhajj, Ken Barker, and Svetlana N. Yanushkevich. "COCALEREX: An Engine for Converting Catalog-based and Legacy Relational Databases into XML", 2004. [Online] available at: http://www.visualgenomics.ca/~wangc/Chunyan_Thesis.pdf.

[74] L. Wood et al., "Document Object Model (DOM) Version 1. W3C Recommendation, October 1998.[Online] available at: http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/

[75] XML Schema Part 2: Data types Second Edition. W3C Recommandation, October 2004. [Online] available at: http://www.w3.org/TR/xmlschema-2/.

[76] Extensible Markup Language (XML) 1.0 Fifth Edition, W3C Recommendation, November 2008. [Online] available at: http://www.w3.org/TR/REC-xml/.

[77] M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura. "XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases". ACM Transactions on Internet Technology (TOIT), Volume 1, pp: 110 – 14, 2001.

Appendix A

UNIVERSITY Object-Oriented Database Schema

```
// class Country
// primary key name
// primitive attribute string name
// primitive attribute int area
// primitive attribute int population
// End Class Definition

public class Country {

      private String name;
      private int area;
      private int population;
}

// class Department
// primary key Name
// primitive attribute string name
// nonprimitive attribute Staff head
// End Class Definition

public class Department {

      private String Name;
      private Staff head;
}

// class Person
// primary key SSN
// primitive attribute int SSN
// primitive attribute string name
// primitive attribute int age
// primitive attribute char sex
// nonprimitive attribute Person spouse
// nonprimitive attribute Country nation
// End Class Definition

public class Person {

      private int SSN;
      private String name;
      private int age;
      private char sex;
      private Person spouse;
      private Country nation;

}

// class Course
// primary key Code
// primitive attribute int Code
// primitive attribute String title
// primitive attribute int credits
```

```
// nonprimitive attribute Prerequisite prerequisite
// End Class Definition


public class Course {

      private int Code;
      private String title;
      private int credits;
      private Prerequisite prerequisite;

}

// class Prerequisite
// primary key course
// nonprimitive attribute Course course
// End Class Definition

public class Prerequisite {

      private Course course;

}

// class Staff
// primary key StaffID
// primitive attribute int StaffID
// primitive attribute int salary
// nonprimitive attribute Department works_in
// superclasses Person
// End Class Definition

public class Staff extends Person {

      private String[] superclasses = {"Person"};
      private int StaffID;
      private int salary;
      private Department works_in;

}

// class Student
// primary key StudentID
// primitive attribute int StudentID
// primitive attribute float gpa
// nonprimitive attribute Department student_in
// nonprimitive attribute Takes takes1
// superclasses Person
// End Class Definition

public class Student extends Person {

      private String[] superclasses = {"Person"};
      private int StudentID;
      private float gpa;
      private Department student_in;
      private Takes takes;
      public class Takes {
```

```
            private Course course;
            private String grade;

      }
}

// class ResearchAssistant
// superclasses Student Staff
// End Class Definition

public class ResearchAssistant extends Student
{
      private String[] superclasses = {"Student", "Staff"};
}

// class Secretary
// primitive attribute int wordsPERminute
// nonprimitive attribute Department works_in
// superclasses Person
// End Class Definition

public class Secretary extends Person {

      private String[] superclasses = {"Person"};
      private int wordsPERminute;
      private Department works_in;

}

// class Takes
// primary key student
// nonprimitive attribute Student student
// nonprimitive attribute Course course
// primitive attribute String grade
// End Class Definition

public class Takes {

      private Student student
      private Course course;
      private String grade;
}
```

Appendix B

Generated Nested XML Schema from UNIVERSITY Object-Oriented Database Schema Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

      <xsd:complexType name="CountryClass" >
            <xsd:sequence>
                  <xsd:element name="name"  type="xsd:string"/>
                  <xsd:element name="area"  type="xsd:int"/>
                  <xsd:element name="population"  type="xsd:int"/>
            </xsd:sequence>
            <xsd:key name="CountryPK">
                  <xsd:selector xpath="CountryClass"/>
                  <xsd:field xpath="name"/>
            </xsd:key>
      </xsd:complexType>

      <xsd:complexType name="CourseClass" >
            <xsd:sequence>
                  <xsd:element name="Code"  type="xsd:int"/>
                  <xsd:element name="title"  type="xsd:String"/>
                  <xsd:element name="credits"  type="xsd:int"/>
                  <xsd:element name="prerequisite"  type="PrerequisiteClass"/>
            </xsd:sequence>
            <xsd:key name="CoursePK">
                  <xsd:selector xpath="CourseClass"/>
                  <xsd:field xpath="Code"/>
            </xsd:key>
      </xsd:complexType>

      <xsd:complexType name="DepartmentClass" >
            <xsd:sequence>
                  <xsd:element name="name"  type="xsd:string"/>
                  <xsd:element name="head"  type="StaffClass"/>
            </xsd:sequence>
            <xsd:key name="DepartmentPK">
                  <xsd:selector xpath="DepartmentClass"/>
                  <xsd:field xpath="Name"/>
            </xsd:key>
      </xsd:complexType>

      <xsd:complexType name="PersonClass" >
            <xsd:sequence>
                  <xsd:element name="SSN"  type="xsd:int"/>
                  <xsd:element name="name"  type="xsd:string"/>
                  <xsd:element name="age"  type="xsd:int"/>
                  <xsd:element name="sex"  type="xsd:char"/>
                  <xsd:element name="spouse"  type="PersonClass"/>
```

```
                <xsd:element name="nation"  type="CountryClass"/>
        </xsd:sequence>
        <xsd:key name="PersonPK">
                <xsd:selector xpath="PersonClass"/>
                <xsd:field xpath="SSN"/>
        </xsd:key>
</xsd:complexType>

<xsd:complexType name="PrerequisiteClass" >
        <xsd:sequence>
                <xsd:element name="course"  type="CourseClass"/>
        </xsd:sequence>
        <xsd:key name="PrerequisitePK">
                <xsd:selector xpath="PrerequisiteClass"/>
                <xsd:field xpath="course"/>
        </xsd:key>
</xsd:complexType>

<xsd:complexType name="ResearchAssistantClass" >
        <xsd:sequence>
                <xsd:element name="StudentSuperclass"   type="Student"/>
                <xsd:element name="StaffSuperclass"   type="Staff"/>
        </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="SecretaryClass" >
        <xsd:sequence>
                <xsd:element name="wordsPERminute"  type="xsd:int"/>
                <xsd:element name="works_in"  type="DepartmentClass"/>
                <xsd:element name="PersonSuperclass"   type="Person"/>
        </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="StaffClass" >
        <xsd:sequence>
                <xsd:element name="StaffID"  type="xsd:int"/>
                <xsd:element name="salary"  type="xsd:int"/>
                <xsd:element name="works_in"  type="DepartmentClass"/>
                <xsd:element name="PersonSuperclass"   type="Person"/>
        </xsd:sequence>
        <xsd:key name="StaffPK">
                <xsd:selector xpath="StaffClass"/>
                <xsd:field xpath="StaffID"/>
        </xsd:key>
</xsd:complexType>

<xsd:complexType name="StudentClass" >
        <xsd:sequence>
                <xsd:element name="StudentID"  type="xsd:int"/>
                <xsd:element name="gpa"  type="xsd:float"/>
                <xsd:element name="student_in"  type="DepartmentClass"/>
                <xsd:element name="PersonSuperclass"   type="Person"/>
        </xsd:sequence>
        <xsd:key name="StudentPK">
                <xsd:selector xpath="StudentClass"/>
                <xsd:field xpath="StudentID"/>
        </xsd:key>
</xsd:complexType>
```

```
      <xsd:complexType name="TakesClass" >
            <xsd:sequence>
                  <xsd:element name="student" type="StudentClass"/>
                  <xsd:element name="grade"  type="xsd:String"/>
                  <xsd:element name="course"  type="CourseClass"/>
            </xsd:sequence>
            <xsd:key name="TakesPK">
                  <xsd:selector xpath="TakesClass"/>
                  <xsd:field xpath="student"/>
            </xsd:key>
      </xsd:complexType>

      <xsd:element name="UnivSchema">
            <xsd:complexType>
                  <xsd:sequence>
                  <xsd:element name="Country" type="CountryClass"/>
                  <xsd:element name="Course" type="CourseClass"/>
                  <xsd:element name="Department" type="DepartmentClass"/>
                  <xsd:element name="Person" type="PersonClass"/>
                  <xsd:element name="Prerequisite" type="PrerequisiteClass"/>
                  <xsd:element
name="ResearchAssistant"type="ResearchAssistantClass"/>
                  <xsd:element name="Secretary" type="SecretaryClass"/>
                  <xsd:element name="Staff" type="StaffClass"/>
                  <xsd:element name="Student" type="StudentClass"/>
                  <xsd:element name="Takes" type="TakesClass"/>
                  </xsd:sequence>
            </xsd:complexType>
      </xsd:element>

</xsd:schema>
```

Appendix C

Generated Flat XML Schema from UNIVERSITY Object-Oriented Database Schema Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:oodb="http://scim.brad.ac.uk/xml">
    <xsd:complexType name="Country_Class" >
        <xsd:sequence>
            <xsd:element name="Country_Object" type="oodb:Country_Object"
                        maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Country_Object" >
        <xsd:sequence>
            <xsd:element name="name"  type="xsd:string"/>
            <xsd:element name="area"  type="xsd:int"/>
            <xsd:element name="population"  type="xsd:int"/>
        </xsd:sequence>
        <xsd:key name="CountryPK">
            <xsd:selector xpath="oodb:Country_Class/oodb:Country_Object"/>
            <xsd:field xpath="oodb:name"/>
        </xsd:key>
    </xsd:complexType>

    <xsd:complexType name="Course_Class" >
        <xsd:sequence>
            <xsd:element name="Course_Object" type="oodb:Course_Object"
                        maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Course_Object" >
        <xsd:sequence>
            <xsd:element name="Code"  type="xsd:int"/>
            <xsd:element name="title"  type="xsd:String"/>
            <xsd:element name="credits"  type="xsd:int"/>
             <xsd:element name="prerequisite"  type="xsd:string"/>
        </xsd:sequence>
        <xsd:key name="CoursePK">
            <xsd:selector xpath="oodb:Course_Class/oodb:Course_Object"/>
            <xsd:field xpath="oodb:Code"/>
        </xsd:key>
        <xsd:keyref name="prerequisiteFK"  refer="PrerequisitePK">
            <xsd:selector xpath="oodb:Course_Class/oodb:Course_Object"/>
            <xsd:field xpath="oodb:prerequisite"/>
        </xsd:keyref>
    </xsd:complexType>

    <xsd:complexType name="Department_Class" >
        <xsd:sequence>
            <xsd:element name="Department_Object"
            type="oodb:Department_Object" maxOccurs="unbounded"/>
        </xsd:sequence>
```

```
    </xsd:complexType>
    <xsd:complexType name="Department_Object" >
        <xsd:sequence>
            <xsd:element name="name"  type="xsd:string"/>
             <xsd:element name="head"  type="xsd:int"/>
        </xsd:sequence>
        <xsd:key name="DepartmentPK">
            <xsd:selector
                 xpath="oodb:Department_Class/oodb:Department_Object"/>
            <xsd:field xpath="oodb:Name"/>
        </xsd:key>
        <xsd:keyref name="headFK"  refer="StaffPK">
            <xsd:selector
                  xpath="oodb:Department_Class/oodb:Department_Object"/>
            <xsd:field xpath="oodb:head"/>
        </xsd:keyref>
    </xsd:complexType>


    <xsd:complexType name="Person_Class" >
        <xsd:sequence>
            <xsd:element name="Person_Object" type="oodb:Person_Object"
                         maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Person_Object" >
        <xsd:sequence>
            <xsd:element name="SSN"  type="xsd:int"/>
            <xsd:element name="name"  type="xsd:string"/>
            <xsd:element name="age"  type="xsd:int"/>
            <xsd:element name="sex"  type="xsd:char"/>
             <xsd:element name="spouse"  type="xsd:int"/>
             <xsd:element name="nation"  type="xsd:string"/>
        </xsd:sequence>
        <xsd:key name="PersonPK">
            <xsd:selector xpath="oodb:Person_Class/oodb:Person_Object"/>
            <xsd:field xpath="oodb:SSN"/>
        </xsd:key>
        <xsd:keyref name="spouseFK"  refer="PersonPK">
            <xsd:selector xpath="oodb:Person_Class/oodb:Person_Object"/>
            <xsd:field xpath="oodb:spouse"/>
        </xsd:keyref>
        <xsd:keyref name="nationFK"  refer="CountryPK">
            <xsd:selector xpath="oodb:Person_Class/oodb:Person_Object"/>
            <xsd:field xpath="oodb:nation"/>
        </xsd:keyref>
    </xsd:complexType>

    <xsd:complexType name="Prerequisite_Class" >
        <xsd:sequence>
            <xsd:element name="Prerequisite_Object"
             type="oodb:Prerequisite_Object" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Prerequisite_Object" >
        <xsd:sequence>
             <xsd:element name="course"  type="xsd:int"/>
        </xsd:sequence>
        <xsd:key name="PrerequisitePK">
```

```
            <xsd:selector
xpath="oodb:Prerequisite_Class/oodb:Prerequisite_Object"/>
            <xsd:field xpath="oodb:course"/>
        </xsd:key>
        <xsd:keyref name="courseFK"  refer="CoursePK">
            <xsd:selector
              xpath="oodb:Prerequisite_Class/oodb:Prerequisite_Object"/>
            <xsd:field xpath="oodb:course"/>
        </xsd:keyref>
    </xsd:complexType>

    <xsd:complexType name="ResearchAssistant_Class" >
        <xsd:sequence>
            <xsd:element name="ResearchAssistant_Object"
             type="oodb:ResearchAssistant_Object" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ResearchAssistant_Object" >
        <xsd:sequence>
            <xsd:element name="StudentSuperclass"  type="xsd:int"/>
            <xsd:element name="StaffSuperclass"  type="xsd:int"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="Secretary_Class" >
        <xsd:sequence>
            <xsd:element name="Secretary_Object"
              type="oodb:Secretary_Object" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Secretary_Object" >
        <xsd:sequence>
            <xsd:element name="wordsPERminute"  type="xsd:int"/>
             <xsd:element name="works_in"  type="xsd:string"/>
            <xsd:element name="PersonSuperclass"  type="xsd:int"/>
        </xsd:sequence>
        <xsd:keyref name="works_inFK"  refer="DepartmentPK">
            <xsd:selector
                 xpath="oodb:Secretary_Class/oodb:Secretary_Object"/>
            <xsd:field xpath="oodb:works_in"/>
        </xsd:keyref>
    </xsd:complexType>


    <xsd:complexType name="Staff_Class" >
        <xsd:sequence>
            <xsd:element name="Staff_Object" type="oodb:Staff_Object"
                        maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Staff_Object" >
        <xsd:sequence>
            <xsd:element name="StaffID"  type="xsd:int"/>
            <xsd:element name="salary"  type="xsd:int"/>
             <xsd:element name="works_in"  type="xsd:string"/>
            <xsd:element name="PersonSuperclass"  type="xsd:int"/>
        </xsd:sequence>
        <xsd:key name="StaffPK">
            <xsd:selector xpath="oodb:Staff_Class/oodb:Staff_Object"/>
```

```
            <xsd:field xpath="oodb:StaffID"/>
        </xsd:key>
        <xsd:keyref name="works_inFK"  refer="DepartmentPK">
            <xsd:selector xpath="oodb:Staff_Class/oodb:Staff_Object"/>
            <xsd:field xpath="oodb:works_in"/>
        </xsd:keyref>
    </xsd:complexType>


    <xsd:complexType name="Student_Class" >
        <xsd:sequence>
            <xsd:element name="Student_Object" type="oodb:Student_Object"
                         maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Student_Object" >
        <xsd:sequence>
            <xsd:element name="StudentID"  type="xsd:int"/>
            <xsd:element name="gpa"  type="xsd:float"/>
             <xsd:element name="student_in"  type="xsd:string"/>
             <xsd:element name="takes1"  type="xsd:string"/>
            <xsd:element name="PersonSuperclass"  type="xsd:int"/>
        </xsd:sequence>
        <xsd:key name="StudentPK">
            <xsd:selector xpath="oodb:Student_Class/oodb:Student_Object"/>
            <xsd:field xpath="oodb:StudentID"/>
        </xsd:key>
        <xsd:keyref name="student_inFK"  refer="DepartmentPK">
            <xsd:selector xpath="oodb:Student_Class/oodb:Student_Object"/>
            <xsd:field xpath="oodb:student_in"/>
        </xsd:keyref>
        <xsd:keyref name="takes1FK"  refer="TakesPK">
            <xsd:selector xpath="oodb:Student_Class/oodb:Student_Object"/>
            <xsd:field xpath="oodb:takes1"/>
        </xsd:keyref>
    </xsd:complexType>


    <xsd:complexType name="Takes_Class" >
        <xsd:sequence>
            <xsd:element name="Takes_Object" type="oodb:Takes_Object"
                         maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Takes_Object" >
        <xsd:sequence>
            <xsd:element name=”studentID” type “xsd:int”/>
            <xsd:element name="grade"  type="xsd:String"/>
             <xsd:element name="course"  type="xsd:int"/>
        </xsd:sequence>
        <xsd:key name="TakesPK">
            <xsd:selector xpath="oodb:Takes_Class/oodb:Takes_Object"/>
            <xsd:field xpath="oodb:studentID"/>
        </xsd:key>
        <xsd:keyref name="courseFK"  refer="CoursePK">
            <xsd:selector xpath="oodb:Takes_Class/oodb:Takes_Object"/>
            <xsd:field xpath="oodb:course"/>
        </xsd:keyref>
    </xsd:complexType>
```

```
    <xsd:element name="UnivSchema">
        <xsd:complexType>
            <xsd:sequence>
            <xsd:element name="Country" type="Country_Class"/>
            <xsd:element name="Course" type="Course Class"/>
            <xsd:element name="Department" type="Department_Class"/>
            <xsd:element name="Person" type="Person_Class"/>
            <xsd:element name="Prerequisite" type="Prerequisite_Class"/>
            <xsd:element name="ResearchAssistant"
                         type="ResearchAssistant_Class"/>
            <xsd:element name="Secretary" type="Secretary_Class"/>
            <xsd:element name="Staff" type="Staff_Class"/>
            <xsd:element name="Student" type="Student_Class"/>
            <xsd:element name="Takes" type="Takes_Class"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

</xsd:schema>
```

Appendix D

Generated Flat XML Document from UNIVERSITY Object-Oriented Database Schema Example

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:xmldoc xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:oodb="http://www.brad.ac.uk/xml">
    <oodb:UNIVERSITY>
        <oodb:CountryClass>
            <oodb:CountryObject>
                <oodb:name>United Kingdom</oodb:name>
                <oodb:area>244820</oodb:area>
                <oodb:population>60,000,000</oodb:population>
            </oodb:CountryObject>
            <oodb:CountryObject>
                <oodb:name>Jordan</oodb:name>
                <oodb:area>100000</oodb:area>
                <oodb:population>6,000,000</oodb:population>
            </oodb:CountryObject>
            <oodb:CountryObject>
                <oodb:name>United States</oodb:name>
                <oodb:area>9372610</oodb:area>
                <oodb:population>300,000,000</oodb:population>
            </oodb:CountryObject>
            <oodb:CountryObject>
                <oodb:name>Canda</oodb:name>
                <oodb:area>9000000</oodb:area>
                <oodb:population>28,000,000</oodb:population>
            </oodb:CountryObject>
            <oodb:CountryObject>
                <oodb:name>Japan</oodb:name>
                <oodb:area>400000</oodb:area>
                <oodb:population>120,000,000</oodb:population>
            </oodb:CountryObject>
        </oodb:CountryClass>

        <oodb:PersonClass>
            <oodb:PersonObject>
                <oodb:SSN>115</oodb:SSN>
                <oodb:name>Tushi Imamura</oodb:name>
                <oodb:age>50</oodb:age>
                <oodb:sex>M</oodb:sex>
                <oodb:spouse>50</oodb:spouse>
                <oodb:nation>Japan</oodb:nation>
            </oodb:PersonObject>
            <oodb:PersonObject>
                <oodb:SSN>132</oodb:SSN>
                <oodb:name>Mohammad Taher</oodb:name>
                <oodb:age>25</oodb:age>
                <oodb:sex>M</oodb:sex>
```

```
        <oodb:spouse></oodb:spouse>
        <oodb:nation>Jordan</oodb:nation>
    </oodb:PersonObject>
    <oodb:PersonObject>
        <oodb:SSN>80</oodb:SSN>
        <oodb:name>Mary Tomson</oodb:name>
        <oodb:age>22</oodb:age>
        <oodb:sex>F</oodb:sex>
        <oodb:spouse>60</oodb:spouse>
        <oodb:nation>United Kingdom</oodb:nation>
    </oodb:PersonObject>
    <oodb:PersonObject>
        <oodb:SSN>50</oodb:SSN>
        <oodb:name>Suzuki Yoshikawa</oodb:name>
        <oodb:age>40</oodb:age>
        <oodb:sex>F</oodb:sex>
        <oodb:spouse></oodb:spouse>
        <oodb:nation>Japan</oodb:nation>
        <oodb:></oodb:>
    </oodb:PersonObject>
    <oodb:PersonObject>
        <oodb:SSN>60</oodb:SSN>
        <oodb:name>Fong Loo</oodb:name>
        <oodb:age>32</oodb:age>
        <oodb:sex>M</oodb:sex>
        <oodb:spouse></oodb:spouse>
        <oodb:nation>United States</oodb:nation>
    </oodb:PersonObject>
    <oodb:PersonObject>
        <oodb:SSN>70</oodb:SSN>
        <oodb:name>James Robert</oodb:name>
        <oodb:age>29</oodb:age>
        <oodb:sex>M</oodb:sex>
        <oodb:spouse></oodb:spouse>
        <oodb:nation>Sweden</oodb:nation>
    </oodb:PersonObject>
    <oodb:PersonObject>
        <oodb:SSN>71</oodb:SSN>
        <oodb:name>Alan Barter</oodb:name>
        <oodb:age>22</oodb:age>
        <oodb:sex>M</oodb:sex>
        <oodb:spouse></oodb:spouse>
        <oodb:nation>United Kingdom</oodb:nation>
    </oodb:PersonObject>
    <oodb:PersonObject>
        <oodb:SSN>73</oodb:SSN>
        <oodb:name>Karen Duncan</oodb:name>
        <oodb:age>20</oodb:age>
        <oodb:sex>F</oodb:sex>
        <oodb:spouse></oodb:spouse>
        <oodb:nation>United Kingdom</oodb:nation>
    </oodb:PersonObject>
    <oodb:PersonObject>
        <oodb:SSN>75</oodb:SSN>
        <oodb:name>Diana Booth</oodb:name>
        <oodb:age>23</oodb:age>
        <oodb:sex>F</oodb:sex>
        <oodb:spouse></oodb:spouse>
        <oodb:nation>Canada</oodb:nation>
```

```
        </oodb:PersonObject>
    </oodb:PersonClass>

    <oodb:StaffClass>
        <oodb:StaffObject>
            <oodb:person>115</oodb:person>
            <oodb:staffID>922</oodb:staffID>
            <oodb:salary>26,000</oodb:salary>
            <oodb:work_in>Physics</oodb:work_in>
        </oodb:StaffObject>
        <oodb:StaffObject>
            <oodb:person>132</oodb:person>
            <oodb:staffID>924</oodb:staffID>
            <oodb:salary>33,000</oodb:salary>
            <oodb:work_in>Computing</oodb:work_in>
        </oodb:StaffObject>
        <oodb:StaffObject>
            <oodb:person>60</oodb:person>
            <oodb:staffID>960</oodb:staffID>
            <oodb:salary>15,000</oodb:salary>
            <oodb:work_in>Computing</oodb:work_in>
        </oodb:StaffObject>
    </oodb:StaffClass>

    <oodb:DepartmentClass>
        <oodb:DepartmentObject>
            <oodb:name>Computing</oodb:name>
            <oodb:head>924</oodb:head>
        </oodb:DepartmentObject>
        <oodb:DepartmentObject>
            <oodb:name>Physics</oodb:name>
            <oodb:head>922</oodb:head>
        </oodb:DepartmentObject>
    </oodb:DepartmentClass>

    <oodb:SecretaryClass>
        <oodb:SecretaryObject>
            <oodb:person>50</oodb:person>
            <oodb:wordsPERminute>35</oodb:wordsPERminute>
            <oodb:works_in>Physics</oodb:works_in>
        </oodb:SecretaryObject>
        <oodb:SecretaryObject>
            <oodb:person>80</oodb:person>
            <oodb:wordsPERminute>45</oodb:wordsPERminute>
            <oodb:works_in>Computing</oodb:works_in>
        </oodb:SecretaryObject>
    </oodb:SecretaryClass>

    <oodb:CourseClass>
        <oodb:CourseObject>
            <oodb:code>COMP3100</oodb:code>
            <oodb:title>Data Structure</oodb:title>
            <oodb:credits>3</oodb:credits>
            <oodb:prerequisite>COMP2100</oodb:prerequisite>
        </oodb:CourseObject>
        <oodb:CourseObject>
            <oodb:code>PHYS1000</oodb:code>
            <oodb:title>Fundamental of Physics</oodb:title>
            <oodb:credits>4</oodb:credits>
```

```
            <oodb:prerequisite></oodb:prerequisite>
        </oodb:CourseObject>
        <oodb:CourseObject>
            <oodb:code>PHYS3210</oodb:code>
            <oodb:title>Classical Physics</oodb:title>
            <oodb:credits>4</oodb:credits>
            <oodb:prerequisite></oodb:prerequisite>
        </oodb:CourseObject>
        <oodb:CourseObject>
            <oodb:code>COMP4500</oodb:code>
            <oodb:title>Compilers Construction</oodb:title>
            <oodb:credits>4</oodb:credits>
            <oodb:prerequisite>COMP3100</oodb:prerequisite>
        </oodb:CourseObject>
        <oodb:CourseObject>
            <oodb:code>COMP2100</oodb:code>
            <oodb:title>Java Programming</oodb:title>
            <oodb:credits>4</oodb:credits>
            <oodb:prerequisite></oodb:prerequisite>
        </oodb:CourseObject>
        <oodb:CourseObject>
            <oodb:code>COMP3200</oodb:code>
            <oodb:title>Algorithms</oodb:title>
            <oodb:credits>3</oodb:credits>
            <oodb:prerequisite>COMP1100</oodb:prerequisite>
        </oodb:CourseObject>
        <oodb:CourseObject>
            <oodb:code>COMP1000</oodb:code>
            <oodb:title>Introduction to CS</oodb:title>
            <oodb:credits>3</oodb:credits>
            <oodb:prerequisite></oodb:prerequisite>
        </oodb:CourseObject>
    </oodb:CourseClass>

    <oodb:PreresquisiteClass>
        <oodb:PreresquisiteObject>
            <oodb:course>COMP3100</oodb:course>
        </oodb:PreresquisiteObject>
        <oodb:PreresquisiteObject>
            <oodb:course>COMP1100</oodb:course>
        </oodb:PreresquisiteObject>
        <oodb:PreresquisiteObject>
            <oodb:course>COMP2100</oodb:course>
        </oodb:PreresquisiteObject>
    </oodb:PreresquisiteClass>

    <oodb:StudentClass>
        <oodb:StudentObject>
            <oodb:person>75</oodb:person>
            <oodb:studentID>08201</oodb:studentID>
            <oodb:gpa>3.3</oodb:gpa>
            <oodb:student_in>Computing</oodb:student_in>
            <oodb:takes>08201</oodb:takes>
        </oodb:StudentObject>
        <oodb:StudentObject>
            <oodb:person>71</oodb:person>
            <oodb:studentID>09001</oodb:studentID>
            <oodb:gpa>2.7</oodb:gpa>
            <oodb:student_in>Computing</oodb:student_in>
```

```
                <oodb:takes>09001</oodb:takes>
            </oodb:StudentObject>
            <oodb:StudentObject>
                <oodb:person>70</oodb:person>
                <oodb:studentID>10005</oodb:studentID>
                <oodb:gpa>3.7</oodb:gpa>
                <oodb:student_in>Physics</oodb:student_in>
                <oodb:takes>10005</oodb:takes>
            </oodb:StudentObject>
            <oodb:StudentObject>
                <oodb:person>60</oodb:person>
                <oodb:studentID>06110</oodb:studentID>
                <oodb:gpa></oodb:gpa>
                <oodb:student_in>Computing</oodb:student_in>
                <oodb:takes></oodb:takes>
                <oodb:></oodb:>
            </oodb:StudentObject>
        </oodb:StudentClass>

        <oodb:TakesClass>
            <oodb:TakesObject>
                <oodb:studentID>08201</oodb:studentID>
                <oodb:course>COMP2100</oodb:course>
                <oodb:grade>B-</oodb:grade>
            </oodb:TakesObject>
            <oodb:TakesObject>
                <oodb:studentID>08201</oodb:studentID>
                <oodb:course>COMP3200</oodb:course>
                <oodb:grade>B+</oodb:grade>
            </oodb:TakesObject>
            <oodb:TakesObject>
                <oodb:studentID>09001</oodb:studentID>
                <oodb:course>COMP2100</oodb:course>
                <oodb:grade>C+</oodb:grade>
            </oodb:TakesObject>
            <oodb:TakesObject>
                <oodb:studentID>09001</oodb:studentID>
                <oodb:course>PHYS1000</oodb:course>
                <oodb:grade>B-</oodb:grade>
            </oodb:TakesObject>
            <oodb:TakesObject>
                <oodb:studentID>10005</oodb:studentID>
                <oodb:course>PHYS1000</oodb:course>
                <oodb:grade>A+</oodb:grade>
            </oodb:TakesObject>
        </oodb:TakesClass>

        <oodb:ResearchAssistantClass>
            <oodb:ResearchAssistantObject>
                <oodb:student>06110</oodb:student>
                <oodb:staff>960</oodb:staff>
            </oodb:ResearchAssistantObject>
        </oodb:ResearchAssistantClass>
    </oodb:UNIVERSITY>
</xsd:xmldoc>
```

Appendix E

Generated Nesetd XML Document from db4o Object-OrientedDatabase

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:nesteddoc xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:oodb="http://scim.brad.ac.uk/xml">
<oodb:DB>
      <Person>
            <SSN>115</SSN>
            <name>Tushi Imamura</name>
            <age>50</age>
            <sex>M</sex>
            <spouse>
                  <SSN>50</SSN>
                  <name>Suzuki Yoshikawa</name>
                  <age>40</age>
                  <sex>F</sex>
                  <spouse>null</spouse>
                  <nation>
                        <name>Japan</name>
                        <area>400000</area>
                        <population>120000000</population>
                  </nation>
            </spouse>
            <nation>
                  <name>Japan</name>
                  <area>400000</area>
                  <population>120000000</population>
            </nation>
      </Person>
      <Person>
            <SSN>50</SSN>
            <name>Suzuki Yoshikawa</name>
            <age>40</age>
            <sex>F</sex>
            <spouse></spouse>
            <nation>
                  <name>Japan</name>
                  <area>400000</area>
                  <population>120000000</population>
            </nation>
      </Person>
      <Person>
            <SSN>132</SSN>
```

```
            <name>Mohammad Taher</name>
            <age>25</age>
            <sex>M</sex>
            <spouse></spouse>
            <nation>
                  <name>Jordan</name>
                  <area>100000</area>
                  <population>6000000</population>
            </nation>
      </Person>
      <Person>
            <SSN>80</SSN>
            <name>Mary Tomson</name>
            <age>22</age>
            <sex>F</sex>
            <spouse>
                  <SSN>60</SSN>
                  <name>Fong Loo</name>
                  <age>32</age>
                  <sex>M</sex>
                  <spouse>null</spouse>
                  <nation>
                        <name>United States</name>
                        <area>9372610</area>
                        <population>300000000</population>
                  </nation>
            </spouse>
            <nation>
                  <name>United Kingdom</name>
                  <area>244820</area>
                  <population>60000000</population>
            </nation>
      </Person>
      <Person>
            <SSN>60</SSN>
            <name>Fong Loo</name>
            <age>32</age>
            <sex>M</sex>
            <spouse></spouse>
            <nation>
                  <name>United States</name>
                  <area>9372610</area>
                  <population>300000000</population>
            </nation>
      </Person>
      <Person>
            <SSN>70</SSN>
            <name>James Robert</name>
            <age>29</age>
            <sex>M</sex>
            <spouse></spouse>
            <nation>
                  <name>Canda</name>
```

```
                <area>9000000</area>
                <population>28000000</population>
        </nation>
    </Person>
    <Person>
        <SSN>71</SSN>
        <name>Alan Barter</name>
        <age>22</age>
        <sex>M</sex>
        <spouse></spouse>
        <nation>
                <name>United Kingdom</name>
                <area>244820</area>
                <population>60000000</population>
        </nation>
    </Person>
    <Person>
        <SSN>73</SSN>
        <name>Karen Duncan</name>
        <age>20</age>
        <sex>F</sex>
        <spouse></spouse>
        <nation>
                <name>United Kingdom</name>
                <area>244820</area>
                <population>60000000</population>
        </nation>
    </Person>
    <Person>
        <SSN>75</SSN>
        <name>Diana Booth</name>
        <age>23</age>
        <sex>F</sex>
        <spouse></spouse>
        <nation>
                <name>Canda</name>
                <area>9000000</area>
                <population>28000000</population>
        </nation>
    </Person>

    <Staff>
        <personsuperclass>
                <SSN>115</SSN>
                <name>Tushi Imamura</name>
                <age>50</age>
                <sex>M</sex>
                <spouse>
                        <SSN>50</SSN>
                        <name>Suzuki Yoshikawa</name>
                        <age>40</age>
                        <sex>F</sex>
                        <spouse>null</spouse>
```

```
                    <nation>
                          <name>Japan</name>
                          <area>400000</area>
                          <population>120000000</population>
                    </nation>
              </spouse>
              <nation>
                    <name>Japan</name>
                    <area>400000</area>
                    <population>120000000</population>
              </nation>
        </personsuperclass>
        <staffID>922</staffID>
        <salary>26000</salary>
        <works_in>
              <name>Physics</name>
              <head>
                    <personsuperclass>
                          <SSN>115</SSN>
                          <name>Tushi Imamura</name>
                          <age>50</age>
                          <sex>M</sex>
                          <spouse>
                                <SSN>50</SSN>
                                <name>Suzuki Yoshikawa</name>
                                <age>40</age>
                                <sex>F</sex>
                                <spouse>null</spouse>
                                <nation>
                                      <name>Japan</name>
                                      <area>400000</area>

    <population>120000000</population>
                                </nation>
                          </spouse>
                    </personsuperclass>
                    <staffID>922</staffID>
                    <salary>26000</salary>
                    <works_in>
                          <name>Physics</name>
                          <head></head>
                    </works_in>
              </head>
        </works_in>

    </Staff>

    <Staff>
        <personsuperclass>
              <SSN>132</SSN>
              <name>Mohammad Taher</name>
              <age>25</age>
              <sex>M</sex>
```

```
                <spouse></spouse>
                <nation>
                      <name>Jordan</name>
                      <area>100000</area>
                      <population>6000000</population>
                </nation>
          </personsuperclass>
          <staffID>924</staffID>
          <salary>33000</salary>
          <works_in>
                <name>Computing</name>
                <head>
                      <personsuperclass>
                            <SSN>132</SSN>
                            <name>Mohammad Taher</name>
                            <age>25</age>
                            <sex>M</sex>
                      <spouse></spouse>
                      </personsuperclass>
                      <staffID>924</staffID>
                      <salary>33000</salary>
                      <works_in>
                            <name>Computing</name>
                            <head></head>
                      </works_in>
                </head>
          </works_in>

    </Staff>

    <Staff>
          <personsuperclass>
                <SSN>60</SSN>
                <name>Fong Loo</name>
                <age>32</age>
                <sex>M</sex>
                <spouse></spouse>
                <nation>
                      <name>United States</name>
                      <area>9372610</area>
                      <population>300000000</population>
                </nation>
          </personsuperclass>
          <staffID>960</staffID>
          <salary>15000</salary>
          <works_in>
                <name>Computing</name>
                <head>
                      <personsuperclass>
                            <SSN>132</SSN>
                            <name>Mohammad Taher</name>
                            <age>25</age>
                            <sex>M</sex>
```

```
                    <spouse></spouse>
                    </personsuperclass>
                    <staffID>924</staffID>
                    <salary>33000</salary>
                    <works_in>
                         <name>Computing</name>
                         <head></head>
                    </works_in>
               </head>
          </works_in>

     </Staff>

     <Secretary>
          <personsuperclass>
               <SSN>50</SSN>
               <name>Suzuki Yoshikawa</name>
               <age>40</age>
               <sex>F</sex>
               <spouse></spouse>
               <nation>
                    <name>Japan</name>
                    <area>400000</area>
                    <population>120000000</population>
               </nation>
          </personsuperclass>
          <wordsPERminute>35</wordsPERminute>
          <works_in>
               <name>Physics</name>
               <head>
                    <personsuperclass>
                         <SSN>115</SSN>
                         <name>Tushi Imamura</name>
                         <age>50</age>
                         <sex>M</sex>
                         <spouse>
                              <SSN>50</SSN>
                              <name>Suzuki Yoshikawa</name>
                              <age>40</age>
                              <sex>F</sex>
                              <spouse>null</spouse>
                              <nation>
                                   <name>Japan</name>
                                   <area>400000</area>

     <population>120000000</population>
                              </nation>
                         </spouse>
                    </personsuperclass>
                    <staffID>922</staffID>
                    <salary>26000</salary>
                    <works_in>
                         <name>Physics</name>
```

```
                        <head></head>
                    </works_in>
                </head>
            </works_in>

    </Secretary>

    <Course>
            <Code>COMP1100</Code>
            <title>Introduction to CS</title>
            <credits>3</credits>
            <prerequisite></prerequisite>

    </Course>

    <Course>
            <Code>COMP2100</Code>
            <title>Java Programming</title>
            <credits>4</credits>
            <prerequisite></prerequisite>

    </Course>

    <Course>
            <Code>COMP3100</Code>
            <title>Data Structure</title>
            <credits>3</credits>
            <prerequisite>
                <Code>COMP2100</Code>
                <title>Java Programming</title>
                <credits>4</credits>
            </prerequisite>

    </Course>

    <Course>
            <Code>PHYS1000</Code>
            <title>Fundamental of Physics</title>
            <credits>4</credits>
            <prerequisite></prerequisite>

    </Course>

    <Course>
            <Code>PHYS3210</Code>
            <title>Classical Physics</title>
            <credits>4</credits>
            <prerequisite></prerequisite>

    </Course>

    <Course>
            <Code>COMP4500</Code>
```

```
            <title>Compilers Construction</title>
            <credits>4</credits>
            <prerequisite>
                  <Code>COMP3100</Code>
                  <title>Data Structure</title>
                  <credits>3</credits>
                  <prerequisite></prerequisite>
            </prerequisite>
      </Course>

      <Course>
            <Code>COMP3200</Code>
            <title>Algorithms</title>
            <credits>3</credits>
            <prerequisite>
                  <Code>COMP1100</Code>
                  <title>Introduction to CS</title>
                  <credits>3</credits>
                  <prerequisite></prerequisite>
            </prerequisite>

      </Course>
</oodb:DB>
</xsd:nesteddoc>
```

Appendix F

Generated XML Schema from ODL Example 6.1

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xsd:schemaxmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="database">
  <xsd:complexType>
   <xsd:choice maxOccurs="unbounded">
    <xsd:element name="person">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="firstName" type="xsd:string"/>
       <xsd:element name="lastName" type="xsd:string"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="book">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="ratings">
        <xsd:complexType>
         <xsd:sequence maxOccurs="unbounded">
          <xsd:element name="item" type="xsd:int"/>
         </xsd:sequence>
        </xsd:complexType>
       </xsd:element>
       <xsd:element name="ISBN" type="xsd:int"/>
       <xsd:element name="chapterPageNumbers">
        <xsd:complexType>
         <xsd:sequence maxOccurs="unbounded">
          <xsd:element name="item">
           <xsd:complexType>
            <xsd:sequence>
             <xsd:element name="key" type="xsd:int"/>
             <xsd:element name="value" type="xsd:int"/>
            </xsd:sequence>
           </xsd:complexType>
          </xsd:element>
         </xsd:sequence>
        </xsd:complexType>
       </xsd:element>
       <xsd:element name="writtenby">
        <xsd:complexType>
         <xsd:attribute name="ID" type="xsd:int"/>
```

```
        </xsd:complexType>
       </xsd:element>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="author">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="firstName" type="xsd:string"/>

       <xsd:element name="ID" type="xsd:int"/>
       <xsd:element name="lastName" type="xsd:string"/>
       <xsd:element name="gender" type="author.sex.enum"/>
       <xsd:element maxOccurs="unbounded" name="write">
        <xsd:complexType>
         <xsd:attribute name="ISBN" type="xsd:int"/>
        </xsd:complexType>
       </xsd:element>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
   </xsd:choice>
  </xsd:complexType>
  <xsd:key name="person.key">
  <xsd:selector xpath="./person|./author"/>
   <xsd:field xpath="firstName"/>
   <xsd:field xpath="lastName"/>
  </xsd:key>
  <xsd:key name="book.key">
   <xsd:selector xpath="./book"/>
   <xsd:field xpath="ISBN"/>
  </xsd:key>
  <xsd:keyref name="book.writtenby.ref" refer="author.key">
   <xsd:selector xpath="./book/writtenby"/>
   <xsd:field xpath="@ID"/>
  </xsd:keyref>
  <xsd:key name="author.key">
   <xsd:selector xpath="./author"/>
   <xsd:field xpath="ID"/>
  </xsd:key>
  <xsd:keyref name="author.write.ref" refer="book.key">
   <xsd:selector xpath="./author/write"/>
   <xsd:field xpath="@ISBN"/>
  </xsd:keyref>
 </xsd:element>
 <xsd:simpleType base="xsd:string" name="author.sex.enum">
```

```
  <xsd:restriction>
   <xsd:enumeration value="FEMALE"/>
   <xsd:enumeration value="MALE"/>
  </xsd:restriction>
 </xsd:simpleType>
</xsd:schema>
```