CARDIFF
UNIVERSITY
PRIFYSGOL
CAERDYDD

Querying Distributed Heterogeneous Structured and Semi-structured Data Sources

by

Fahad M. Al-Wasil

A thesis submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
in
Computer Science

School of Computer Science

Cardiff University

April 2007

DECLARATION

This work has not previously been accepted in substance for any degree and is not concurrently submitted in candidature for any degree.

Signed ....[illegible]..........(candidate)

Date .....18./.3../...2.0.0.7..............

STATEMENT 1

This thesis is being submitted in partial fulfillment of the requirements for the degree of PhD.

Signed ....[illegible]..........(candidate)

Date ..18./.3../..2.0.0.7................

STATEMENT 2

This thesis is the result of my own independent work/investigation, except where otherwise stated. Other sources are acknowledged by explicit references.

Signed ....[illegible]..........(candidate)

Date ...18./.3../..2.0.0.7..............

STATEMENT 3

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ....[illegible]..........(candidate)

Date ...18./..3./..2.0.0.7................

To my parents,

my wife,

and my children Nora, Mohammed, and Adeem

Acknowledgements

I would like to start by praising Allah (God) Almighty for providing me with faith, patience and commitment to complete this research.

I would like to express my sincere gratitude to my supervisors, Prof. W. A. Gray and Prof. N. J. Fiddian, for their expert guidance and encouragement throughout this research. I am grateful for their careful reading and constructive comments on this thesis and our joint papers.

I would like to thank the paper referees whose comments on my published papers have added to the success of this project.

Special thanks are due to the members of the school for their help, especially Mrs. Margaret Evans who has helped me with travel related issues, Mrs. Helen Williams for her help in administrative issues, and Mr. Robert Evans and Dr. Rob Davies for their technical assistance.

I would also like to express my thanks to my fellow research students in the School of Computer Science at Cardiff University for their friendship and help. I really enjoyed their friendship that I developed while doing this research.

Special admiration and gratitude is due to my parents, brothers and sisters whose prayers, love, care, patience, support and encouragement have always enabled me to perform to the best of my abilities.

Last, but certainly not least, I am indebted to my wife for her endurance and unconditional support which provided vital encouragement during the period of my PhD study. Without her love and devotion, this research would have been impossible. Finally, I would like to mention my beloved children, Nora, Mohammed, and Adeem who have given me happiness during the difficult period of my study.

Abstract

The continuing growth and widespread popularity of the internet means that the collection of useful data available for public access is rapidly increasing both in number and size. These data are spread over distributed heterogeneous data sources like traditional databases or sources of various forms containing unstructured and semi-structured data. Obviously, the value of these data sources would in many cases be greatly enhanced if the data they contain could be combined and queried in a uniform manner.

The research work reported in this dissertation is concerned with querying and integrating a multiplicity of distributed heterogeneous structured data residing in relational databases and semi-structured data held in well-formed XML documents produced by internet applications or human-coded. In particular, we have addressed the problems of: (1) specifying the mappings between a global schema and the local data sources' schemas, and resolving the heterogeneity which can occur between data models, schemas or schema concepts; (2) processing queries that are expressed on a global schema into local queries.

We have proposed an approach to combine and query the data sources through a mediation layer. Such a layer is intended to establish and evolve an XML Metadata Knowledge Base (XMKB) incrementally which assists the Query Processor in mediating between user queries posed over the global schema and the queries on the underlying distributed heterogeneous data sources. It translates such queries into sub-queries -called local queries- which are appropriate to each local data source. The XMKB is built in a bottom-up fashion by extracting and merging incrementally the metadata of the data sources. It holds the data source's information (names, types and locations), descriptions of the mappings between the global schema and the participating data source schemas, and function names for handling semantic and structural discrepancies between the representations.

To demonstrate our research, we have designed and implemented a prototype system called SISSD (System to Integrate Structured and Semi-structured Databases). The system automatically creates a GUI tool for meta-users (who do the metadata integration) which they use to describe mappings between the global schema and local data source schemas. These mappings are used to produce the XMKB. The SISSD allows the translation of user queries into sub-queries fitting each participating data source, by exploiting the mapping information stored in the XMKB.

The major results of the thesis are: (1) an approach that facilitates building structured and semi-structured data integration systems; (2) a method for generating mappings between a global and local schemas' paths, and resolving the conflicts caused by the heterogeneity of the data sources such as naming, structural, and semantic conflicts which, may occur between the schemas; (3) a method for translating queries in terms of a global schema into sub-queries in terms of local schemas. Hence, the presented approach shows that: (a) mapping of the schemas' paths can only be partially automated, since the logical heterogeneity problems need to be resolved by human judgment based on the application requirements; (b) querying distributed heterogeneous structured and semi-structured data sources is possible.

Contents

List of Figures

Acronyms

API	Application Programming Interface
CDM	Common Data Model
DB	Database
DBMS	Database Management System
DBS	Database System
DDB	Distributed Database
DDBMS	Distributed Database Management System
DOM	Document Object Model
DTD	Document Type Definition
FDBS	Federated Database System
FLWR	For-Let-Where-Return
GAV	Global-As-View

GUI	Graphical User Interface
HTML	HyperText Markup Language
JDBC	Java Database Connectivity
JDOM	Java Document Object Model
JXC	Java XML Connectivity
KS	Knowledge Server
LAV	Local-As-View
MDBS	Multi-database System
MDE	Metadata Extractor
MVP	Master View Parser
OEM	Object Exchange Model
QP	Query Processor
SAX	Simple API for XML
SGML	Standard Generalized Markup Language
SISSD	System to Integrate Structured and Semi-structured Databases
SQL	Structured Query Language
SSD	Schema Structure Definition
SSDP	Schema Structure Definition Parser
UDF	User-Defined Function
URL	Uniform Resource Locater
W3C	World Wide Web Consortium
XDSDL	XML Data Source Definition Language
XFEP	XQuery FLWR Expression Parser

XMKB	XML Metadata Knowledge Base
XMKBML	XML Metadata Knowledge Base Mapping Language
XML	Extensible Markup Language
XQIS	XQuery Internal Structure
XSD	XML Schema Definition

CHAPTER 1

Introduction

1.1 Motivation of the Research

Users and application programs in a wide variety of businesses today are increasingly requiring the integration of multiple distributed autonomous heterogeneous data sources [86, 130]. The continuing growth and widespread popularity of the Internet mean that the collection of useful data sources available for public access is rapidly increasing both in number and size. Furthermore, the value of these data sources would in many cases be greatly enhanced if the data they contain could be combined, "queried" in a uniform manner (i.e. using a single query language and interface), and subsequently returned in a machine-readable form. For the foreseeable future, much data will continue to be stored in relational database systems because of the reliability, scalability, tools and performance associated with these systems [68, 133]. However, due to the impact of the web, there is an explosion in complementary data availability: this data can be automatically generated by web-based applications or can be human-coded [102]. Such data is called semi-structured data, which means that although the data may have some structure, the structure is not regular or complete as is the case with data held in traditional database management systems (See [9] for a survey on semi-structured data). In the domain of semi-structured data, the eXtensible Markup Language (XML) is arguably the major data representation language as well as data exchange format. XML has a W3C specification [4] that allows creation and transformation of a semi-structured document conforming to its XML syntax rules which has no referenced DTD or XML schema. Such a document has metadata buried inside the document and is called a well-formed XML document. The well-formed XML documents simply markup pages with descriptive tags. It doesn't need to describe or explain what these tags mean. In other words a well-formed XML document does not need a DTD or XML schema, but is must conform to the XML syntax rules. If all tags in a document are correctly formed and follow XML guidelines, then a document is considered as well-formed. The metadata content of an XML document enables automated processing, generation, transformation and consumption of the semi-structured data in the document by applications. Much interesting and useful data can be published as a well-formed XML document by web-based applications or by human-coding.

Hence, building a data integration system that provides a unified method of access to semantically and structurally diverse data sources is highly desirable as it will be able to link structured data residing in relational databases and semi-structured data held in well-formed XML documents [73, 101]. These XML documents can be XML files on local hard drives or documents held on remote web servers. Such a data integration system will have to find structural transformations and semantic mappings that result in correct merging of the data and allow users to query the resulting so-called mediated schema [100]. This linking is a challenging problem since the pre-existing databases concerned are typically autonomous and located on heterogeneous hardware and software platforms. This means it is necessary to resolve conflicts caused by the heterogeneity of the data sources which can occur between data models, schemas or schema concepts. Consequently, mappings between entities in different sources representing the same real-world objects have to be defined. The main difficulty in this process is that the related data in different sources may be represented in different formats and in incompatible ways. For instance, bibliographical databases of different publishers may use different formats for authors' or editors' names (e.g. full name or separated first and last names), or different units for prices (e.g. dollars, pounds or euros). Moreover, the same expression may have a different meaning, or the same meaning may be specified by different expressions. This means that syntactical data and metadata alone cannot provide sufficient semantics for all potential integration purposes. As a result, the data integration process is often very labour-intensive and demands more computing expertise than most application users have. Therefore, semi-automated approaches are the most promising way forward, where mediation engineers are given an easy to use tool to describe mappings between the integrated (integrated and master are used interchangeably in this thesis) view and local schemas. This produces an integrated schema which is a uniform view over all the participating local data sources [148]. In the thesis we use interchangeably the terms mediated, integrated, master and global to describe the global view created by the integration process.

XML is becoming the de-facto standard format to exchange information over the internet. The advantages of XML as an exchange model - such as rich expressiveness, clear notation and extensibility - make it an excellent candidate to be a data model for an integrated schema. As the importance of XML has increased, a series of standards has grown up around it, many of which were defined by the World Wide Web Consortium (W3C). For example, the XML Schema language provides a notation for defining new types of XML elements and XML documents. XML with its self-describing hierarchical structure and associated language XML Schema provide the flexibility and expressive power needed to accommodate distributed and heterogeneous data. At the conceptual level, the data can be visualized as trees or hierarchical graphs.

This thesis concentrates on the problem of integrating and querying a multiplicity of distributed heterogeneous structured data residing in relational databases and semi-structured data sources held as well-formed XML documents.

1.2 Problem Statement

A vast and growing amount of heterogeneous data sources is available to institutions or companies. As a result integration of such data sources in the public domain is inevitable. Therefore, integrating and querying heterogeneous data sources is a fundamental problem in data management [25, 52]. The problem is concerned with building data integration systems, which provide a unified view over heterogonous data sources. Such a unified view is structured according to a so-called *mediated schema* (often referred to as a global schema), which describes the contents of the data sources and exposes the aspects of the data that might be of interest to the user. The reason for this is that one of the principle goals of a data integration system is to free the user from having to know about the specific data sources and their structure in order to interact with them [35, 119]. A meditated schema is a virtual representation of the data available to its user in the integrated system, (in the sense that the data in the local data sources need not conform to its structure). As a consequence, the data integration system must first reformulate a user query into a query that refers directly to the schemas in the data sources. In order for the system to be able to reformulate a user query, it needs to have a set of data source descriptions, specifying the mapping between the elements in the data sources and the elements in the mediated schema. These descriptions specify the relationship between elements.

In this context, providing a reasonable structured and semi-structured data integration framework for a user to effectively integrate and query distributed heterogeneous structured data residing in relational databases and semi-structured data held in well-formed XML documents has become a challenge for database integration researchers. There is a lack of fully automated schema-mapping processes, and a high degree of logical heterogeneity between the data sources. Another problem impeding data integration is the query translation process, which is one of the most important problems in the design of a data integration system, as it enables the system to reformulate a query posed in terms of the global schema into a set of queries, suited to the local data sources. Thus, tools are needed to mediate between user queries and heterogeneous data sources which transform such queries into local queries. Doing these tasks manually is not only time consuming but also error prone. Hence, methods for simplifying heterogeneous data source integration would be of great theoretical and practical importance. Therefore, our objective is to facilitate the task of a designer building an XML data integration system. In general, building data integration systems requires the designer to address several issues [87]. In this thesis, we concentrate on two basic issues:

1. Specifying the mappings between the global schema and the local data sources.
2. Processing queries expressed against the global schema into queries reflecting local schemas.

1.3 Hypothesis, Aims and Objectives

In our research, the main focus is on integrating and querying distributed heterogeneous structured and semi-structured data sources. Our hypothesis is that:

It is possible to integrate and query the distributed heterogeneous structured data residing in relational databases and semi-structured data held in well-formed XML documents which can be found on a local hard drive or remote web servers, by building in a bottom-up approach a dynamic XML Metadata Knowledge Base (XMKB) of data source meta-data resolving structural and semantic conflicts in the data that is used in rewriting a user query over a chosen view into sub-queries which fit each local data source, by using the mapping information stored in the XMKB.

This thesis shows how to mediate distributed heterogeneous structured and semi-structured data sources in a mediation architecture which enables users to query multiple structured and semi-structured data sources in a uniform manner. Specifically, our goals are to:

1. Facilitate the designer effort involved in building structured and semi-structured data integration systems.
2. Design a system capable of partially automating the integration of distributed heterogeneous structured and semi-structured data sources.
3. Resolve the logical heterogeneity, such as naming, structural, and semantic conflicts which, may occur between the schemas. Thus a solution which overcomes the logical heterogeneity problem is needed.
4. Enable transparent querying of all data sources participating in the integration system without the users needing a detailed knowledge of the underlying data sources, their location and their structure. Thus, formulating a method for translating a user query into local queries is desired.

1.4 Achievement of the Research

The importance of this research lies in its demonstration of the feasibility of building an XML Metadata Knowledge Base (XMKB), in a bottom-up fashion by extracting and merging incrementally the metadata of the data sources, and its demonstration of the benefit of this XMKB in mediating user queries posed over the global schema into local queries on the distributed heterogeneous data sources, by translating such queries into sub-queries which are appropriate to each local data source. The main contributions of this thesis are:

1. Since fully automatic schema mapping generation is infeasible, a semi-automatic approach is demonstrated based on an assisting tool which reduces the designer effort required to build integration systems linking structured and semi-structured data. A solution to overcome the heterogeneity problem is formulated. Two important tasks were developed to solve the problem: (1) establishing appropriate mappings between the global schema and the schemas of the local data sources; (2) users queried the distributed heterogeneous structured and semi-structured data sources in terms of the global schema, with a mapping process and query translation process formulated to transform these queries into local queries.

2. A prototype system is developed to demonstrate that the ideas explored in the thesis are sound and practical.

3. A bottom-up approach is used to establish and evolve the XML Metadata Knowledge Base (XMKB) incrementally from the metadata extracted from the data sources.

4. Tools have been developed which can be used to overcome conflicts, such as naming, structural, and semantic conflicts which may occur between the schemas.

5. A mapping is established between global schema elements and each local data source schema elements to link the elements with the same meaning by using a unique index number generated automatically for the global schema elements.

6. The design of the XML Metadata Knowledge Base (XMKB) to capture:

   a) The mapping information between the global schema elements and the local data sources' elements,
   b) The function names of the functions handling semantic and structural discrepancies,

   and to assist the Query Processor (QP) in generating sub-queries for relevant local data sources.

7. A software tool has been designed and built which extracts metadata from data sources to build the Schema Structure Definition (SSD) for these data sources. This tool can be applied to relational databases, well-formed XML documents which have no referenced DTDs or XML schemas, and also XML documents with referenced DTDs or XML schemas.

1.5 Organization of the Thesis

This section presents an overview of the thesis' organization. The first chapter has presented an introduction to the research undertaken, motivations, the hypothesis to be tested and highlights the aims and objectives of the research and its original achievements.

Chapter 2: *Background and survey of the state-of-the-art*

This chapter presents an overview of the work in the field of integrating distributed heterogeneous data sources and how it relates to this thesis.

Chapter 3: *XML and related technologies*

This chapter presents an overview of XML and related technologies.

Chapter 4: *The SISSD data integration system*

This chapter introduces the main ideas of the thesis. It presents a brief description of the motivation of this work, and describes our approach and its system architecture. In addition, it describes the logical heterogeneity problem, and introduces an application example which is used through out the thesis to show how the integration is accomplished by the system.

Chapter 5: *The mediation process*

This chapter details the mediation process which is the first part of our approach. It is a basic idea of the thesis, as it is proposed as a tool to overcome the heterogeneity problems which may occur among the data sources.

Chapter 6: *The query translation process*

This chapter details the second important point in the thesis that is the query translator process which is an integral part of the mediation layer of the system. It gives a brief introduction to the query translation task in data integration systems, and presents the query translation process developed in this work. Finally, it gives some examples of query translations.

Chapter 7: *The SISSD implementation*

This chapter covers the implementation of the proposed architecture. It presents the implementation of the metadata extracting process. It also presents the implementation of the processes used in creating an XMKB. In addition, it introduces the development of the query parsing and translating processes.

Chapter 8: *Evaluation & Discussion*

This chapter focuses on the evaluation of the prototype system and contains a critical assessment of our research approach and its contribution.

Chapter 9: *Summary, conclusion and future work*

This chapter concludes the thesis with a summary of the accomplishments and issues to be considered in the future.

CHAPTER 2

Background and survey of the state-of-the-art

The integration of data sources poses many challenges due to differences in data management systems, data models, query and data manipulation languages, data types, format (structured, semi-structured), representation, and semantics. This chapter discusses related work and the basic issues affecting the integration of heterogeneous distributed data sources. Firstly, we give an overview of the field of distributed heterogeneous databases. Secondly, since the main topic of this work is querying and integrating data from a network of data sources, we present the approaches for solving this problem. Thirdly, we give an overview of data interoperability. Next, we present a detailed survey on data integration. Finally, we summarize related work on querying and integrating heterogeneous data sources.

2.1 Distributed heterogeneous databases

A database integrates and stores related data in an organized manner. A database system (DBS) [48] consists of software, called a database management system (DBMS), one or more databases that it manages, and any associated application software utilizing the database contents. A DBMS is the software that handles all access to the database. A DBS may be either centralized or distributed. A centralized DBS consists of a single centralized DBMS managing a single database on the same computer. A distributed DBS consists of a single distributed DBMS (DDBMS) managing multiple databases. The databases may reside on a single computer system or on multiple computer systems that may differ in hardware and system software.

A *Distributed Database (DDB)* is defined as a collection of multiple, logically interrelated data distributed over different computers of a computer network [23, 38, 45, 62, 122]. The physical distribution does not necessarily imply that the computer systems are geographically far apart; they could actually be in the same building or even in the same room. It simply implies that communication between them is done over a network instead of through shared memory. Each node of the network has autonomous capability, performs local applications and may participate in the execution of some global applications that require accessing data at several sites. Distributed databases [64] emerged as a merger of two technologies: (1) database technology, and (2) network and data communication technology. They also met the requirement of organizations interested in the decentralization of processing while achieving an integration of the information resources at the logical level within their geographically distributed systems of databases.

A particular property of a distributed database is that it can be homogenous or heterogeneous [136]. A homogenous distributed database (simply called a distributed database) is one in which all the physical components run on the same distributed database management system, and the distributed database system supports a single data model and query language with a single schema.

Conversely, database systems that provide interoperation and varying degrees of integration among multiple databases of different types have been termed heterogeneous distributed database systems (simply called a heterogeneous database). They consist of database systems which differ physically and logically, have different data models, manipulation languages, and schemas. Despite these databases being independently created and managed they must cooperate and interoperate. Users need to access and manipulate data from several databases and applications may require data from a wide variety of the independent databases. Therefore, a new system architecture is required to manipulate and manage distinct and multiple databases, in a transparent way.

There are a number of factors that differentiate types of DDBMS. These factors characterize a set of multiple DBSs in three orthogonal dimensions: *distribution*, *heterogeneity*, and *autonomy* [32, 62, 121, 134-136]. These dimensions characterize systems in which multiple databases may be put together and be managed by multiple DBMS. We introduce each of these dimensions below.

The distribution dimension specifies how the data of a DDBS is distributed among multiple sites in a computer network.

Heterogeneity is concerned with the differences between the local DBSs comprising the DDBS. The types of heterogeneity are caused by technological differences and independent design. These may be classified as system heterogeneity and logical heterogeneity [71]. System heterogeneity covers differences in hardware, operating system, database management system (including data models, languages, transaction management) and communication systems. Logical heterogeneity covers differences in the way the real world is modeled in the databases (i.e. differences in schema and data representation).

Autonomy refers to the distribution of control, not of data. It indicates the degree to which individual DBSs can operate independently [90]. Autonomy is a function of a number of factors such as whether the component systems exchange information, whether they can independently execute transactions, and who is allowed to modify them. Several kinds of autonomy (design, communication, execution and association autonomy) can be identified [136].

2.2 A Taxonomy for Integrating Heterogeneous Data Sources

Integration of heterogeneous data sources continues to receive much attention from the research community [19, 42, 46, 47, 74, 104, 107, 150]. Information systems integration is a complex problem since information systems comprise data, processes and applications. As a consequence their integration must be done at each level [53]. In the context of this thesis, we consider only data integration. Since the main topic of this work is querying and integrating data from a network of data sources, we present other proposed solutions for this problem and highlight their strengths and shortcomings. We then consider a particular approach, Mediation Systems, and characterize it in more detail.

We first distinguish between *materialized* and *virtual* approaches. They are called in [144] the *eager* or *in-advance* approach and the *lazy* or *on-demand* approach. In the materialized approach, data coming from the local data sources are integrated and stored in a single new database. All queries then operate on this comprehensive database. While in the virtual approach, data remains in the local data sources. Thus, queries operate directly on the local data sources and data integration takes place during query processing by combining results. As a consequence, the two approaches have the following advantages and disadvantages:

- In the materialized approach, data must first be prepared before queries can be submitted. The participating data sources are (manually) analyzed; a static view over the data is defined, the local data is used to populate a new integrated database conforming to the static view and queries are formulated against this view. As a consequence, new data sources cannot be easily integrated and made available for querying. This approach is suitable for applications which require specific, exact portions of the available data which are mostly static (for example, financial transactions). A query is evaluated directly using the materialized database and as a result query processing can be optimized for this database. Additionally, there is no need to access the underlying data sources, so connection costs are non-existent. However if the local data is dynamic, updating of the integrated DB is hard. Also some of the materialized data may never be accessed.

- For the virtual approach, a query must first be analyzed in order to find data sources which can answer it, and then it is split into sub-queries which finally are adjusted according to the query capabilities of each data source. As a consequence, query processing is dependent on the availability of the data sources, their connection times and query performance. Query optimization opportunities are limited and an important requirement for this approach is that data sources accept ad-hoc queries. Its main advantage is that new data sources can be easily made available for querying. This approach is suitable for users with "unpredictable needs" [144], i.e. if users have a variety of information needs. It is suited to dynamic databases as the processing occurs on the local data, and there is no need to preprocess data not required by a query.

In our work, we adopt the virtual approach to supporting a read-only data integration of distributed heterogeneous structured and semi-structured data, which means a global schema is created to be used for answering user queries, and not for updating data. Since the number of underlying data sources linked in the integration system may increase or decrease at anytime, and in a materialized approach data is imported into a new integrated repository, this type of dynamic change can not be easily made. The data requirement of the expected users is unpredictable and likely to vary with the resources currently linked. For these reasons, a virtual approach is more suitable as it produces a scalable system with respect to the dynamic nature of the available information resources.

Another classification of approaches for integrating heterogeneous data is based on the structure of the data. Most data sources can usually be classified into one of three categories depending on the kind of data that they are primarily designed to handle:

1. Text retrieval systems are concerned with the management and query-based retrieval of collections of unstructured text documents.
2. Structured database systems are concerned with the management of structured or strictly-typed data, i.e., data that conforms to a well-defined schema (e.g., data held in DBS managed by DBMSs).
3. Semi-structured databases are designed to efficiently manage data that only partially conforms to a schema, or whose schema can evolve rapidly (e.g. XML documents) [9].

There are approaches which consider integrating just one kind of data such as relational databases [28, 93], or Object-Oriented databases [13, 59], or XML documents [18, 120, 148], so query formulation, processing and results accommodate only that particular kind of data. On the other hand there has been a significant interest in combining, integrating, and inter-operating between heterogeneous data that belong to different classes of data sources[86, 113]. The primary motivation for most of the work in this area is that many applications require processing of data that belongs to more than one type. For instance, a medical information system at a hospital must process doctor reports (free text documents) as well as patient records (structured relational data). Similarly, an order processing application might need to handle inventory information in a relational database as well as purchase orders received as (semi-structured) XML documents [126].

Earlier work on database integration [12, 21, 65, 79, 96, 111, 140] focussed on the integration of well-structured databases, with fixed schemas, that support powerful query languages. This thesis focusses on the integration of distributed heterogeneous structured and semi-structured data sources. For the foreseeable future, most data will continue to be stored in relational database systems because of the reliability, scalability, tools and performance associated with these systems. Additionally, much interesting and useful data can be published as a well-formed XML document, this data can be automatically generated by Web-based applications or can be human-coded. Such data is called semi-structured data due to its varying degree of structure. It can also vary between static databases and ephemeral data having a very short life. Hence, with the web's increasing role as a data provider, building a data integration system that provides unified access to semantically and structurally diverse data sources is highly desirable as it will link structured data residing in relational databases and semi-structured data held in well-formed XML documents produced by Internet applications or human-coded.

Since we are targeting a system for querying and integrating distributed heterogeneous structured and semi-structured data sources, our work has adopted a mediation approach. There have been several integration methods which combine data from several data sources such as universal DBMS (UDBMS) method [54, 95], federated databases [34, 61, 111, 136], data warehouse [27, 41, 150], multi-databases [61, 71, 90, 94, 96, 109-111, 134], and mediator method [20, 70, 99, 125, 138, 145]. While other methods are applicable to integration of structured heterogeneous data which is usually stored using a DBMS, a mediation approach is appropriate to integration of unstructured, semi-structured, and structured data.

We now overview some integration approaches in more detail as classified in Figure 2.1 [54]. There are additional features that characterize these approaches which are not presented in this figure, but these will be discussed in the following subsection.

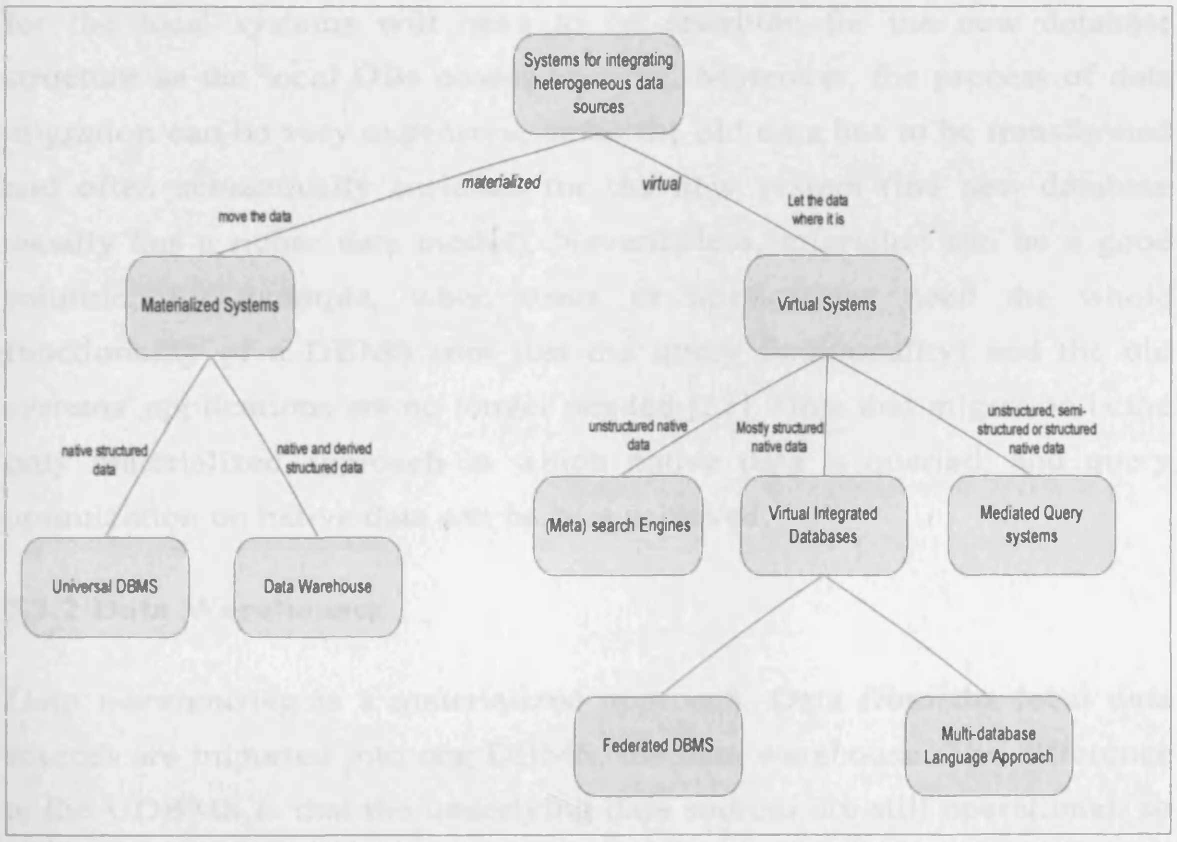

Systems for integrating heterogeneous data sources
materialized
virtual
move the data
Let the data where it is
Materialized Systems
Virtual Systems
native structured data
native and derived structured data
unstructured native data
Mostly structured native data
unstructured, semi-structured or structured native data
Universal DBMS
Data Warehouse
(Meta) search Engines
Virtual Integrated Databases
Mediated Query systems
Federated DBMS
Multi-database Language Approach

Figure 2.1: Classification of Systems for Integrating Heterogeneous Data Sources [54].

2.2.1 Universal Database Management Systems

In a UDBMS approach data is migrated from the local systems to a unique separate DBMS. First, the global integrated schema is defined and then data from the local systems is imported into the new database and the local system ceases to operate. In this way queries can be formulated against the new database and results are presented to users. In this case, the underlying data sources are usually also DBSs and the new DBS must accommodate all types of data available in the underlying sources. Thus, the new database must be able to handle all (or many) types of information, i.e. it must be a *universal* DBS.

During the migration process, data from the underlying systems are extracted, transformed, integrated and stored in the central universal database. The main drawback of this approach is that existing applications for the local systems will have to be rewritten for the new database structure as the local DBs ceases to exist. Moreover, the process of data migration can be very expensive; since the old data has to be transformed and often semantically enriched for the new system (the new database usually has a richer data model). Nevertheless, migration can be a good solution, for example, when users or applications need the whole functionality of a DBMS (not just the query functionality) and the old systems' applications are no longer needed [22]. Note that migration is the only materialized approach in which native data is queried, and query optimization on native data can be best achieved.

2.2.2 Data Warehouses

Data warehousing is a materialized approach. Data from the local data sources are imported into one DBMS, the data warehouse. The difference to the UDBMS is that the underlying data sources are still operational, so the data is in fact replicated deliberately in at least two DB. First, the warehouse schema is defined, data from the underlying sources is processed and stored in the data warehouse. The warehouse data is typically not imported in the same form and volume as it exists in the local data systems. It may be transformed, cleaned and prepared for certain analysis tasks, like data mining and OLAP (Online Analytical Processing). Data warehouses often do not make the most recent data available, since a data warehouse is usually not updated immediately after a local data source has changed because of the overheads associated with immediate. Thus, the warehouse stores historical data, as required by OLAP and data mining applications.

According to [144] the data warehouse approach is suitable for the following kinds of clients:

- Clients who do not need the most recent data available, since a data warehouse is usually not updated immediately after a local data source has changed;
- Clients who require historical, derived and specific information - for this reason data may need to be transformed, cleaned, aggregated and prepared for certain analytical tasks, such as data mining and OLAP (Online Analytical Processing); or
- Clients who require high query performance - since large amounts of complex data must be queried; data warehouses are optimized for the dominant business scenario but are less than optimal for others.

2.2.3 Metasearch Engines

Regarding the querying of unstructured distributed sources, *search engines* and *metasearch engines* have gained importance in recent years, mainly because of the development of the Web. Search engines are systems which accept as queries only natural language keywords (or simple combinations of them) and return documents as answers. The main characteristic of search engines is that data can be easily made available for querying and queries can be formulated in a simple way.

Search engines are characterized by:

- search efficiency which means how fast the results are returned, and
- search effectiveness which indicates how good the results are, or "the ability to retrieve what the user wants to see" [129].

In order to achieve high effectiveness, search engines use heuristics for finding the meaning of input queries and for retrieving documents which may match them. However, if we consider a heterogeneous environment like the Web, it is very difficult to find the right meaning of queries and in such cases search engines perform quite poorly.

To increase effectiveness different search engines are combined to form metasearch engines. Their users formulate queries against a uniform interface, which are processed (for example, stop words are eliminated) and split into sub-queries which are then sent to the individual search engines. Finally, the results are collected, combined and presented in a unified way. Examples are SavvySearch [57] and MetaCrawler [131, 132]. Metasearch engines do not prepare the data to be queried. They simply use the query interfaces of the underlying search engines and prepare the input queries for them. They also need to implement suitable heuristics for combining the results from the different sources. Metasearch engines are thus examples of systems for querying data available in a network of data sources. However, the underlying sources must be unstructured and for this reason they are not suitable for querying sources where the data is structured in any way and would be inappropriate when linking structured and semi- structured data.

2.2.4 Virtual Integration of Databases

For many years the virtual integration of data that is stored in different databases has been an active research topic. The approaches to enabling virtual integrated access to multiple databases can be roughly classified in two categories: the **federated database system** [136] and the **multi-database system** [110]. These approaches are suitable for structured data and support precise searches. In the following subsections, we overview these types.

2.2.4.1 Federated database systems

In many situations, data are managed or accessed exclusively through their DBS by applications which respect the management system boundaries of the local systems, though much more powerful applications can be created when these data are integrated. One possibility of integrating two different databases is called a gateway. A gateway is a special program that simulates access from one database to another by coding protocols of interaction. However this has its limitations, such as the amount of time needed to design the gateways and that data accessing through gateways makes further data transparency difficult to achieve. An alternative approach would be a middleware architecture [48] which provides a transparent and uniform view of multiple data sources and maintains this interface for database applications in case a new data source becomes available. An example of integrating multiple databases through a middleware approach is the concept of Federated Database System (FDBS) [136]. A FDBS is a collection of distributed, heterogeneous and semi-autonomous DBSs integrated through a federation layer. One of the significant aspects of a FDBS is that its component DBS can continue local operations and at the same time participate in a federation. DBSs participating in a FDBS are always heterogeneous and distributed within this FDBS. These participating DBSs are called Component DBSs

(CDBS). A typical FDBS architecture is shown in [136]. This architecture is called a five-level schema architecture, as it is composed of five schema types:

- Local schema: schema of each local database that comprises the federation;
- Component schema: local schema translated to the canonical model;
- Export schema: subset of the component schema to be accessed by the federation;
- Federated or Global schema: schema generated by the integration of export schemas;
- External schema: global schema view, available to a group of users and/or applications.

In FDBS, the amount of integration does not have to be comprehensive as in global schema integration, but depends on the needs of the users, as FDBS may be either tightly or loosely coupled systems. The integration of component DBSs may be managed either by the users of the federation or by the administrators of the component DBSs. In loosely coupled FDBSs, the federation schema creation is done by the users, whereas in tightly coupled FDBSs, the creation and maintenance of federated schema and access to export schemas is controlled by federation administrators. Thus in loosely coupled approaches the linkage of terms is undertaken at query time by the user while in a tightly coupled approach it is undertaken when the DB joins the federation.

2.2.4.2 Multi-database systems

Multi-database systems (MDBSs) [111] provide access to multiple preexisting databases that support their own applications and end users.

The MDBS should be able to identify data stored in different databases and support multi-database queries and updates by resolving data incompatibilities, performing query decomposition, and executing multi-database transactions. This process can be wholly or partially transparent to the end user. However, local systems must have full control over their data, and thus preserve their own autonomy.

MDBSs meet the need for organizations to interoperate their databases already in service by supporting new global applications that access multiple databases. The fundamental difference between MDBSs and DDBSs relates to the definition of the global conceptual schema. In the case of logically integrated DDBSs, the global conceptual schema defines the conceptual view of the entire set of databases available, while in the case of distributed MDBSs; it represents only a collection of local databases that are being linked for a specific purpose. Thus the definition of a global schema is different in MDBSs than in distributed DDBSs. In a DDBS the global database is equal to the union of all the local databases, whereas in the MDBS it is only a subset of the same union.

A MDBS allows each local database system to continue to operate independently. Global users access data stored in various local database systems (LDBSs) and local users access data stored in a single LDBS. The basic MDBS architecture consists of a global transaction manager which handles the execution of global transactions and is responsible for dividing them into sub-transactions for submission to the LDBSs participating in the MDBS. Global transactions are posed using the global view constructed by integrating the local views provided by each local database system.

2.2.5 Summary of previous approaches

The approaches presented in the previous sections have the following characteristics:

- they consider either structured or unstructured data,
- but do not combine structural heterogeneous data;
- they offer either an exact or a fuzzy kind of search; and
- for the structured underlying sources, they integrate data based on a common schema.

However, when dealing with the large amount of data available online which might be unstructured, semi-structured or structured data, and the high number of users searching with different knowledge levels and aims, there is a need for a new approach to querying heterogeneous data. Thus, new techniques are needed for building a system which allows integration of heterogeneous data in a way that could easily connect and disconnect underlying sources and support all kinds of users formulating queries which integrate web available data with traditional structured data held in DBSs.

A solution to this problem was introduced at the beginning of the 1990s by Wiederhold [145], when he defined the concept of mediator.

2.2.6 Mediation System

A mediator is defined in [145] as: "A mediator is a software module that exploits encoded knowledge about certain sets or subsets of data to create information for a higher layer of applications."

When querying integrated heterogeneous data sources, mediators have the primary task of offering to the user a homogeneous integrated view over the data. Thus mediation deals with various types of heterogeneity, with data mismatch and supports users in the formulation of queries.

The typical mediator architecture contains three layers, (see Figure 2.2). The upper layer (*query interface*) is the user and application interface. The middle layer (*mediator*) contains application-specific mediators, which use a unified data model. On the lower layer there are data sources with their corresponding *wrappers*. Wrappers are software modules which translate the request coming from the middle layer to a query for a data source, and translate the results returned from sources into the unified data model representation of the system. Consequently, they implement a homogenization of the data sources, which means users of the interface are unaware of heterogeneity present at the data sources level.

Systems based on the mediation approach do not retrieve data from the data sources until the data is requested. A user query is decomposed by the mediator component-- a software module responsible for creating a virtual integrated view of the data sources in the system. The mediator determines which data sources contain relevant information and queries those data sources. The mediation approach guarantees that the retrieved data are always up to date as it is accessing the local data source itself. This approach is also known as the lazy approach (or on-demand, or virtual approach), i.e., the queries are unfolded and rewritten at runtime as they flow downwards in the architecture from the query interface to the data sources. Query processing in this case is very similar to the metasearch engine case, with the difference that data in the underlying sources may be heterogeneous in its representation, i.e. structured, semi-structured or unstructured. The TSIMMIS Project at Stanford [42, 69] and the MIX project at University of California at San Diego [20] employ this approach.

Since in materialized approaches data is imported into a new repository (either a universal database or a data warehouse), changes can not be easily made to the local data source, a virtual approach is more suitable for a scalable system where the local data sources are dynamic. However, there are many factors which influence the scalability of a system and not every virtual system is scalable per se. Furthermore, if the underlying data sources are only structured data sources, a materialized approach or a federated database approach or multi-database approach may also be suitable. If the underlying data sources are all unstructured then a metasearch engine approach is suitable. However, if all or some kinds of data (structured, semi-structured and unstructured) have to be queried, mediation is the only suitable approach since it is the only approach dealing with dynamic, structurally heterogeneous data sources.

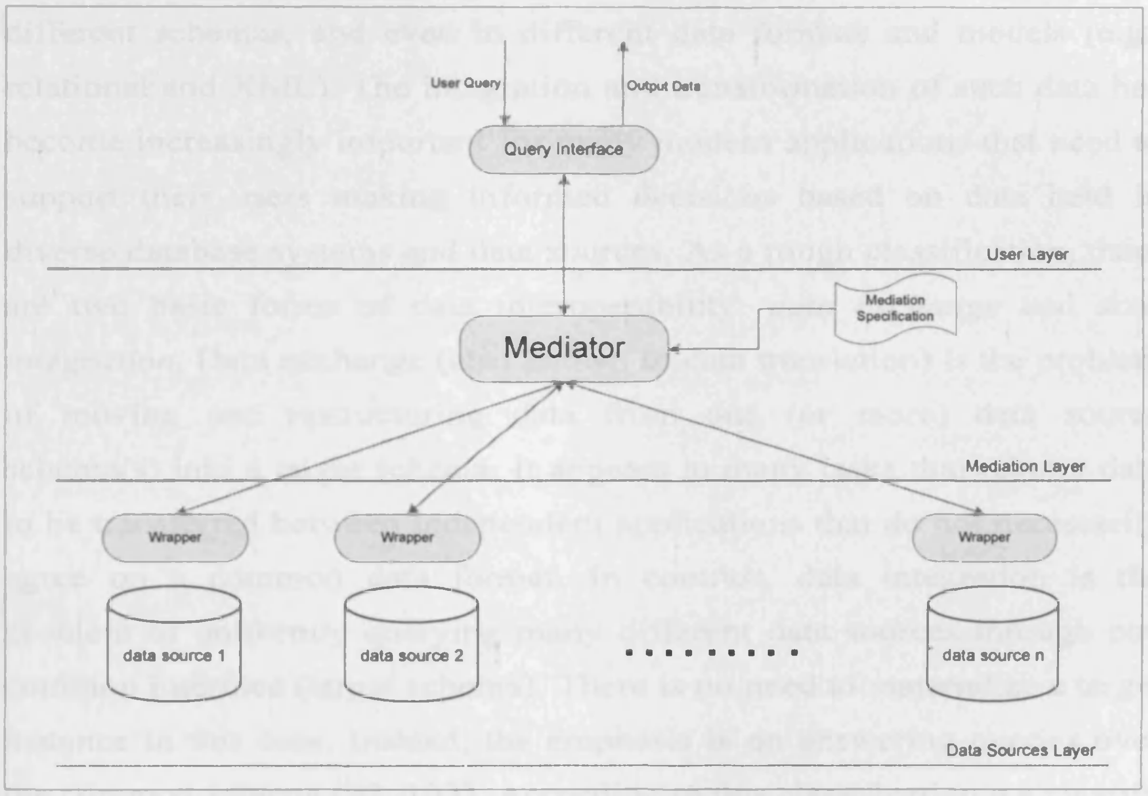


Figure 2.2: The three-tier mediator architecture.

The focus of this thesis is on querying dynamic heterogeneous data sources, since many users and applications today need just this functionality. Thus the mediation approach can be used in our system. However, there are a lot of applications which also need updates on the underlying data sources or even full database functionality such as access control, and transaction management. In these cases, approaches such as the materialized and federated database can be used and are more appropriate.

2.3 Data interoperability

Interoperability is the magic word that is expected to allow heterogeneous data sources to talk to each other and exchange information in a meaningful way. The data interoperability problem occurs when this is hard to achieve and arises from the fact that data, even within a single domain of application, are available at many different sites, in many different schemas, and even in different data formats and models (e.g., relational and XML). The integration and transformation of such data has become increasingly important for many modern applications that need to support their users making informed decisions based on data held in diverse database systems and data sources. As a rough classification, there are two basic forms of data interoperability: *data exchange* and *data integration*. Data exchange (also known as data translation) is the problem of moving and restructuring data from one (or more) data source schema(s) into a target schema. It appears in many tasks that require data to be transferred between independent applications that do not necessarily agree on a common data format. In contrast, data integration is the problem of uniformly querying many different data sources through one common interface (target schema). There is no need to materialize a target instance in this case. Instead, the emphasis is on answering queries over the common schema [92, 103]. According to this classification we classify our work as a data integration problem as we use a virtual global schema over different data sources and data held in these data sources can be combined and queried through this global schema.

Data *interoperability* [77, 83, 110] is the ability of distributed, heterogeneous data sources, which are independently created and administrated and have different semantics and schemas to cooperate and interoperate in a transparent way to the user while maintaining their autonomy and objectives. Data interoperability [84, 149] can be achieved by integration of existing data in virtual databases, i.e. databases which are logically defined but not physically materialized.

However, the integration of structured and semi-structured data sources poses some fundamental challenges [81]. The heterogeneity that may exist between a set of independently designed data sources is one of these challenges. Data is stored within distinct heterogeneous data sources. This means the important kind of heterogeneity in our context is structural heterogeneity [54, 67].

2.4 Heterogeneity of the data sources

If the contents of data sources are related in some way, they are still likely to show variety in many aspects. These differences can make both the design and modeling phase and the operation phase of a data integration system very difficult. The major issue in building a data integration system is resolving these differences between the data sources that may occur at different levels. This issue is generally referred to as *heterogeneity* of the data sources.

Heterogeneity arises at different levels for various reasons. Firstly, an organization for various reasons, may adopt different platforms for its applications. It may choose different hardware, operating systems and different communication protocols. Secondly, as a result of advances in software and technology developments, different data sources may become available over time; these data sources may have different data models, query languages and/or other facilities. Thirdly, the independent design of the component databases may lead to semantic heterogeneity, where the designers of these databases may have different opinions about how to model the same real world objects.

Broadly speaking, the heterogeneity may be classified as System Heterogeneity (low level) and Logical Heterogeneity (high level) [71].

System heterogeneity comes from adopting different platforms for the computer installation. Platform differences include: hardware systems, operating systems, data management systems and networking protocols. Logical heterogeneity occurs when people use different approaches to model the same real world objects [21, 71, 97, 98]. Both types of heterogeneity result from the autonomy of development of systems.

Researchers and developers have been working on resolving system heterogeneity for many years. The causes of such heterogeneity are well understood [71, 97] and may not exist if the same hardware, system software (e.g., operating system) and communication protocols are used. While research on logical heterogeneity started more recently, it still reflects more than 20 years of research [21].

Detecting and resolving logical heterogeneity is acknowledged to be a difficult problem, because it requires a good understanding of the data's meaning, the inconsistencies present in the data and the level of incomplete information. Unfortunately, it is not possible to fully capture real world semantics by using available data modeling techniques [128]. Therefore nearly all the tools that deal with detecting and reconciling semantic heterogeneity depend on user interaction to complement and validate their results [71, 97, 98, 137].

Schema Conflicts	
Conflict Type	**Description**
Table Name Conflicts	Using different names for equivalent tables (Synonym) or the same name for different tables(Homonym)
Table Structure Conflicts	One table contains more attributes than another table with equivalent concepts
Table Constraint Conflicts	Incompatible key and update constraints
Multiple Table Conflicts	Using different numbers of tables to store information

Attribute Name Conflicts	Using different names for equivalent attributes (Synonym) or the same name for different attributes (Homonym)
Multiple Attribute Conflicts	Representing a concept using more attributes in one database than another
Table versus Attribute Conflict	Representing a concept as a table in one database and as a field in another
One-to-Many Element Conflicts	This type of conflict arises when information captured in one element in the global schema is equivalent to that split into more than one elements in the local data sources.
Many-to-One Element Conflicts	This type of conflict occurs when more than one element in the global schema corresponding to one element in a local data source.
Data Conflicts	
Data Type Conflicts	The same element may have incompatible type definitions in different databases. For example, social security number could be of type 'character' in one database and 'numeric' in another.
Unit Conflicts	Numerical data represented using different units.
Precision Conflicts	This conflict occurs when two data sources use values from the domains of different cardinalities for the same data.
Expression Conflicts	This conflict arises when different expressions are used to represent the same data.
Representation Conflicts	The same concept is represented by different constructs of the model.
Granularity Conflicts	Data elements representing measurements differ in granularity levels, e.g., sales per month or annual sales.
Default values conflict	This conflict arises when the default values of semantically equivalent elements in different data sources are different.
Key Conflicts	Different keys are assigned to the same concept in different schemas.
Behavioral Conflicts	These arise when different insertion/deletion policies are associated with the same class of objects in distinct schemas.
Wrong data Conflicts	It occurs when equivalent attributes in different data sources, which are expected to have the same value, have different values.

Figure 2.3: Conflicts Classification.

Various classifications of heterogeneities have been suggested in papers related to data integration, without necessarily providing full classifications. In an analysis of schema integration methodologies [21, 98], structural and semantic diversity categories were specified as those involving naming conflicts and those involving structural conflicts.

Figure 2.3 shows a classification of conflicts that may exist between a set of independently designed data sources. It is based on the classifications of [21, 97, 98]. One of our goals in this research is to resolve the heterogeneity, such as naming, structural, and semantic conflicts which, may occur between the schemas (see [16]). Thus a solution which overcomes the heterogeneity problem is needed. Later, we will describe how our system SISSD can be used to handle some of conflicts identified in Figure 2.3 (see sections 4.4, 5.4.4 and 8.1.6).

2.5 Data integration

Data integration has received significant attention since the early days of databases. Much research has been devoted to solving the problem of data integration. With the popularity of the Internet, access to data becomes independent of its physical storage location. Additionally, users can access a variety of data sources that are related in some way to find out useful information, but this is often cumbersome. Therefore, integrating heterogeneous data sources so that users can easily access and combine the data is an important challenge. Much of the research on integration has focused on so-called data integration [80, 103]. Data integration is the process of combining the data residing at different sources, and providing the user with a unified view of these data. Such a unified view is structured according to a so-called global schema, which provides the elements to express queries over the data of the integration system. Data integration is an important data management application because it is a common user requirement. The main objective of a data integration system is to facilitate users in focussing on specifying 'what' data they want, rather than on describing 'how' to obtain data. To achieve this task, an integrated view of the data stored in the underlying data sources should be provided. In data integration systems, users are interested mainly in querying the integrated data rather than updating the data through the integrated view.

The problem of the differences between data sources is of great importance. Usually, the contents of data sources are related in some way, but show diversity in many representational aspects. This diversity, which is usually referred to as heterogeneity [136], causes the design of a data integration system to be a challenge. Heterogeneity is one of the most complicated issues that are taken into consideration when building a data integration system. Hence, resolving the differences between the data sources is a crucial issue.

There are different layers of heterogeneity beginning from hardware heterogeneities and continuing to differences in the operating systems or communication protocols. On a higher level there is logical heterogeneity, which refers to the degree of dissimilarity between the component data sources that make up a data integration system. Logical heterogeneity is one of the most complicated issues taken into consideration in a data integration system. It comes from different understanding and modeling of the same concept. Subsequently, the construction of a data integration system must handle different mechanisms for attributing meaning to the data (semantic conflicts), for referencing data (naming conflicts), and for storing data (structural conflicts). The distinction between semantic and structural heterogeneity is not always precise. Structural heterogeneity refers basically to the structure of the data, while semantic heterogeneity refers to the domain of concepts (their interpretation).

Basically, to build a data integration system, relationships or mappings must be established between the data source schemas and the global schema [35, 147].

Definition 2.5.1 (A data-integration system)

A data-integration system I is a triple (G, S_i, M_j), where G is a global schema, S_i is a set of n source schemas, and M_j is a set of m source-to-global mappings, such that for each source schema S_i there is a mapping M_j from S_i to G, $(1 \leq i \leq n)$, $(1 \leq j \leq m)$.

A crucial issue in data integration is how elements of the global schema and elements of the data sources are mapped. Based on the direction of mappings between a data source and global schema, the approaches are classified into the so called *global-as-view* (GAV) and *local-as-view* (LAV) approaches [80, 103]. The following sections describe each of these approaches. We will further use the symbol $\Rightarrow$ that means an implication relationship between the global and local schemas' elements exists.

2.6 Global-As-View (GAV) approach

In a Global-As-View (GAV) approach, a global schema is defined in terms of a set of local source schemas. That is, the global schema is defined as a *view* over the local sources' schemas [20, 70, 75, 141].

In a GAV approach, query reformulation reduces to simple rule unfolding (standard execution of views in ordinary databases). However, changes in data sources or adding a new data source requires revision of the global schema to take into account the changes, and requires a revision of the mappings between the global schema and data source schemas. Thus, GAV is not scalable for large applications [24, 147].

In recent years, many systems have been developed in research projects on data integration using the GAV technique. Below we discuss briefly well-known research projects and prototypes of the better known of these systems.

2.6.1 GAV systems

- TSIMMIS [42, 108, 124], one of the first GAV systems and the most representative of the GAV approach. This system uses the OEM (Object Exchange Model) to convey information between the components of the system. The first basic component of a mediated system, the mediator, is specified using MSL (Mediator Specification Language). It is a logic-based, object-oriented language that can be seen as a view definition language, targeted to the OEM data model. The second component is wrappers which are specified using a WSL (Wrapper Specification Language). WSL is an extension of MSL, supporting the description of source contents and source query capabilities.

- MIX [20] MIX stands for Mediation Information using XML. It is a successor of TSIMMIS. The basic difference from TSIMMIS is that XML is used as the language (i) to represent the global schema and (ii) to exchange data between the mediator and the XML sources (instead of OEM). The query language of MIX is XMAS (XML Matching and Structure Language). XMAS uses features from several XML query languages, queries are formulated in terms of the mediated schema, and are written as XMAS queries that refer to the source views exported by the wrappers. These queries are then sent to the wrapped sources for evaluation.

- Nimble [55, 56] is a commercial system similar to MIX. Nimble integrates XML sources. The architecture of the Nimble system is based on a set of mediated schemas, which are defined as views over the schemas of the data sources. The query language used by the Nimble system is XML-QL. When a query is posed to the integration system, it is decomposed into multiple source queries based on the data sources. The compiler translates each such query into an appropriate query language for the destination source.

- Clio [117, 118] was developed by IBM around 2000. Clio is a research prototype of a schema mapping creation tool. The focus is on discovering queries that map values from the data sources to values in the global schema. Both source and global schema are considered to be either relational or XML. Clio produces a set of mappings between the source schema and the global schema, given a set of high-level correspondences defined by the user. It also involves transforming legacy data into a new target schema. Clio introduces an interactive schema mapping paradigm, based on value correspondences. The user specifies how a value of a target attribute can be created from a set of values of source attributes using a query/browsing GUI.

2.7 Local-As-View (LAV) approach

In the Local-As-View (LAV) approach, a global schema is defined independently of the local source schemas. Each source is described in terms of the global schema relations. That is, the sources are described as a materialized view of the global schema [11, 36, 99, 103].

The LAV approach makes it very simple to add or remove data sources from the system, but it also complicates the query reformulation task. It is scalable and better suited to integrating a large number of autonomous read-only data sources accessible over communication networks.

Furthermore the LAV approach provides a flexible environment able to accommodate the continual change and update of data source schemas.

In recent years, many systems have been developed in various research projects on data integration using the LAV technique. We discuss briefly the best known research projects and prototypes of the more prominent representative LAV systems.

2.7.1 LAV systems

- Information Manifold [99, 104, 107] handles the problem of data integration by providing a mechanism to describe declaratively the contents and the query capabilities of information sources. In the Information Manifold system the global schema is relational. A source description is a conjunctive query over the global schema relations, which will be referred to as a view. User queries, posed in Information Manifold, are conjunctive queries like source descriptions. They are expressed in terms of the global schema relations.

- Infomaster [58, 74] is an information system which provides integrated and uniform access to multiple distributed, heterogeneous, structured sources. Data available in a source is also seen as a set of relations, called site relations. Between site relations and interface relations, a set of base relations are defined. Interface relations are defined as views on the base relations. User queries are expressed in terms of the interface relations. Queries are rewritten in terms of the site relations.

- Agora [113, 114] system supports querying and integration of heterogeneous relational and XML information sources. The global schema is an XML DTD and a virtual relational schema is used as an interface between the sources and this schema. Relational and

XML sources are modeled as SQL queries over a relational global schema. Users formulate XQuery queries in terms of this global DTD. These queries are normalized and translated into an SQL query over the generic relational schema.

- DDXMI [120, 148] (for Distributed Database XML Metadata Interface) is a system prototype that has been built to generate a tool to do the metadata integration, producing a master DDXMI file, which is then used to generate queries to local databases from the master queries. It builds on the XML Metadata Interchange. DDXMI is a master file including database information. In this approach local sources were designed according to DTD definitions. Therefore, the integration process is started from the DTD parsing that is associated with each source.

2.8 Related Work

Data integration has received significant attention since the early days of databases. In recent years, there have been many research projects focussing on distributed heterogeneous data source integration. Most of them are based on the common mediator architecture [145] such as Garlic [37], the Information Manifold [99], Disco [141], TSIMMIS [42], Yat [43], Mix [20], MedMaker [123] and Agora [113]. The goal of such systems is to provide a uniform user interface to query integrated views over heterogeneous data sources. A user query is formulated in terms of the integrated view, to execute a query, the system translates it into sub-queries expressed in terms of the local schemas, sends the sub-queries to the local data sources, retrieves the results, and combines them into the final result provided to the user. Mainly, they can be classified into structural approaches and semantic approaches.

In structural approaches, local data sources are assumed to be crucial. The integration is done by providing or automatically generating a global unified schema that characterizes the underlying data sources. On the other hand, in semantic approaches, integration is achieved by sharing a common ontology among the data sources. According to the mapping direction, the approaches are further classified into two categories: global-as-view (GAV) and local-as-view (LAV) [103]. In GAV approaches, each item in the global schema is defined as a view over the source schemas. In LAV approaches, each item in each source schema is defined as a view over the global schema. The LAV approach is well-suited to supporting a dynamic environment, where data sources can be added or removed from the data integration system without restructuring the global schema.

Projects and prototypes such as Garlic, TSIMMIS, MedMaker, and Mix are structural approaches and take a global-as-view approach. A common data model is used, e.g., OEM (Object Exchange Model) in TSIMMIS and MedMaker. Mix uses XML as the data model; an XML query language XMAS was developed and used as the view definition language. Many efforts are being made to develop semantic approaches, based on RDF (Resource Description Framework) and Knowledge-based Integration [112]. Several ontology languages have been developed for data and knowledge representation to assist data integration from a semantic perspective, such as F-logic [115] which is employed to represent knowledge in the form of a domain map used to integrate data sources at the conceptual level.

DDXMI [120, 148] builds on XML Metadata Interchange. DDXMI is a master file including database information, XML path information (a path for each node starting from the root), and semantic information about XML elements and attributes. A system prototype has been built that generates a tool to do the metadata integration, producing a master DDXMI file, which is then used to generate queries to local databases from master queries. In this approach local sources were designed according to DTD definitions. Therefore, the integration process is based on DTD parsing associated with each source. XIQM [18] is an approach to mediating heterogeneous XML data sources. A tool is proposed for the XML data integration system to combine and query XML documents through a mediation layer. This layer is intended to describe the mappings between the global XML schema and local heterogeneous XML schemas. It produces a uniform interface over the local XML data sources and provides the required functionality to query these sources in a uniform way. It involves two important units: the XML Metadata Document (XMD) and the Query Translator. XMD is an XML document containing metadata, in which the mappings between global and local schemas are defined. XML Query Translator which is an integral part of the system is introduced to translate a global user query into local queries by using the mappings that are defined in XMD. The XML data sources are described by the XML Schema language.

We classify our work as being in the structural category but we differ from the others such as Garlic, Disco, TSIMMIS, Mix, MedMaker and Yat by following a local-as-view (LAV) approach as this approach is well-suited to supporting a dynamic environment, where data sources can be added or removed from the system without restructuring the global schema. It is better suited and scalable for integrating a large number of autonomous read-only data sources accessible over communication networks. Furthermore the LAV approach provides a flexible environment able to accommodate the continual change and update of data source schemas, especially suited to XML documents on Web servers which are not static and often subject to frequent update. Projects like Information Manifold, Agora, DDXMI and XIQM are integration systems with a LAV architecture; however, in Information Manifold the local and global schemas are relational, while in DDXMI and XIQM the local and global schemas are XML. The Agora system supports querying and integrating data sources of diverse formats, including XML and relational sources under an XML global schema, but assumes explicit schemas for XML data sources. Our work [14-16] focuses on querying and integrating distributed heterogeneous structured data residing in relational databases and semi-structured data held in XML documents. The XML documents that we are interested in are well-formed XML documents, while DDXMI targets XML documents designed according to DTD definitions, and XIQM targets XML documents satisfying an XML schema. Thus we are dealing with all types of XML document unlike these systems. Also our work differs from DDXMI and XIQM by using an incremental tool to build the XML Metadata Knowledge Base (XMKB). This tool starts from an existing XMKB file and slightly modifies it in light of minor changes to data source schema structures or when data sources are added or removed from the system, instead of regenerating it from scratch. Thus it facilitates evolution reflecting the dynamic nature of the data being targetted.

CHAPTER 3

XML and related technologies

Machine readable data files are text files or binary files. There has been an aim to find a universal format which combines the features of both these types with rich information storage capability. An early attempt to combine a universally interchangeable data format with rich information storage capabilities was SGML (Standard Generalized Markup Language) [8, 76]. This is a text-based language that can be used to mark up data by adding meta-data in a way which is self describing. SGML was designed to be a standard way of marking up data for any purpose. It is a complicated language that it is not well suited for data interchange over the Web [31]. A very well-known language, based on SGML is Hypertext Markup Language (HTML) [2]. However, despite HTML being incredibly successful, it was limited in its scope, as it is only intended for displaying documents in a browser. Therefore, XML (eXtensible Markup Language) [4] was created to address this limitation. Development of XML started in summer 1996 by the setting up of an XML Working group by the W3C

(World Wide Web Consortium) [1]. In many aspects, HTML gives a good introduction for understanding XML: the ASCII representation of XML data and HTML share the same syntactical notions, which is part of SGML.

Since XML is chosen to be a data model for our data integration system and in this thesis we used some XML related technologies we give in this chapter an overview of XML and some related technologies.

We start with an introduction to XML. Next, we present an overview of the Document Type Definition (DTD) grammar language and XML Schema language since these languages are used to describe the structure of an XML document. Then, we introduce XML Application Programming Interfaces, and finally we introduce XML query languages.

3.1 XML

W3C is an open, public organization whose task is to develop technology and standards for the Internet. It has developed XML standards for efficient information exchange across the Web. The basic concept behind XML is that data should be self-describing by means of tags associated with the data. XML provides no predefined tags, instead it is a meta-markup language which provides standards for users so that they can define their own tags, document structure and the definition of the tag.

XML [4] has quickly emerged as a standard for data representation and data exchange over the Web. XML is a subset of SGML. It specifies a set of rules for putting data structures into a text file. Although XML is text, it is not primarily meant to be read by humans but rather by machines, with standardized XML parsers. The power of XML as a description language lies in the fact that an XML document contains self-describing, hierarchically structured data, and it allows association of markup terms with data elements. These markup terms serve as metadata allowing formalized description of the content and structure of the accompanying data. XML appears to subsume HTML and its successor XHTML as the communication language for the Internet [66]. By associating metadata terms with data elements, XML has enabled documents to be communicated between organizations in a way that enables their semantics to be completely understood both by human and machine agents. In other words, just as HTML is used to render texts so that they can be processed by humans, XML renders data structures so that they can be processed by computers so that the processed document can be presented on a human interface.

Data in XML are grouped into elements delimited by tags. The first line of an XML document (see Figure 3.1) is a mandatory statement that tells the XML processor[1] that it is dealing with XML version 1.0 in this case [60]. The rest of an XML document is composed of tags and text. Every opening tag must have a matching closing tag, and the tag must be properly nested. A tag consists of text enclosed in a pair of angle brackets. A tag is also called a markup. The document has a root element that contains all other elements. Any properly nested piece of text of the following form

<tag> …… </tag>

is called an XML element, and the name of that element is the tag. Figure 3.1 is a simple example of an XML document.

In this figure, *<bookstore>*, *<book>*, … …, and *</bookstore>* are tags. The text between the opening and closing tag is called the *content* of the element. Elements directly nested within other elements are called *children*. XML also defines the *ancestor/descendant* relationships among elements, which are important for querying XML documents. An ancestor

[1] The XML processor is a module that reads an XML document to find out its structure and contents.

is a *parent*, a *grandparent*, etc., and a descendant is a *child*, a *grandchild*, etc.

```
<?xml version = "1.0"?>
<bookstore>
  <book category="WEB">
  <isbn> 0-321-12226-7 </isbn>
<title> Learning XML </title>
<author> Erik T. Ray </author>
<year> 2005 </year>
<price> 29.99 </price>
</book>
</bookstore>
```

Figure 3.1: An example of a simple XML document.

An opening tag can have attributes. An element can have any number of user-defined attributes. XML attributes are useful in data representation as they offer a richer representation than elements can offer. Attribute values can only be strings, which strictly limits their usefulness, while XML elements can have children elements, which make them much more versatile. Some features of attributes are-. First, the order of attributes in an element does not matter; second, an attribute can occur at most once in an element, while elements with the same tag can be repeated; third, using attributes can lead to briefer representation. A useful feature of an XML attribute is that it can be declared to have a unique value and can also be used to enforce a limited referential integrity. This can not be declared with elements alone in plain XML. In the above example, *Price* is defining as an *attribute* in the *element Book.*

There are two important concepts of XML documents, *well-formed* and *validity*. *Well-formed* deals with the physical structure referring to tags which are properly matched and nested while, *validity* focuses on the logical structure of elements.

Definition 3.1: An XML document is well formed if it has a root element, every opening tag is followed by a matching closing tag, the elements are properly nested, and any attribute can occur at most once in a given opening tag and its value must be provided.

An XML document must be well-formed to be processed. That is, it must be syntactically correct. The validity concept is provided by an XML schema grammar language which is introduced in the next section.

Our system is designed to deal with well-formed XML documents which conform to the XML syntax rules but have no referenced DTD or XML schema. However, it can also deal with XML documents which have DTD or XML schema by bypassing the DTD or the XML schema. It accesses the document itself to extract its structure and uses our simple language XDSDL (XML Data Source Definition Language) to describe the *actual* structure of the data source not the *possible* one described in the referenced DTD and XML Schema.

3.2 DTD and XML Schema

There are many grammar languages that can describe the structure of an XML document. The most common are: *DTD* [30] and *XML Schema* [3]. XML schema is an optional document-structure grammar which is used to make sure the XML document is valid. XML documents can be defined according to a schematic representation defined in a DTD or XML Schema. An XML document that conforms to a DTD or XML Schema is called a valid XML document.

3.2.1 DTD

DTD is a set of rules for structuring an XML document. It is a context-free-grammar for the document. The DTD describes a document type by specifying which tags are allowed, their attributes, and the allowed nesting.

Roughly, the DTD corresponds to the schema definition in relational or object-oriented databases. The schema of an XML document may be defined by a DTD, which describes a grammar for semi-structured documents.

The basic components of a DTD grammar are *elements*, *attributes*, and *entities*. The structure of the contents of *elements* is defined by

<!ELEMENT content-model >

where, a *content-model* in a DTD may involve the following types:

- EMPTY type: a simple element with no content, but may have attributes.
- ANY type: elements of this type may have arbitrary content.
- #PCDATA type: a simple element of only character data
- Expression: a composite element which is a regular expression over element names.
- A composite element may be defined by the following constructs:
  - ",": to define a sequence of ordered component elements.
  - "| ", to define alternatives of component choice.
  - "*", an element that can appear arbitrarily often.
  - "+", as "*" but must appear at least once.
  - "?", optional element can appear 0 or 1 times.

Attributes can be associated to an element. Each attribute has a name, a data type and optional constraints that restrict its permitted values to an enumeration or a fixed value, or defines it as a required property. An element with attributes is also considered to be a composite element. The allowed *attributes* of elements can be declared as:

- #REQUIRED: the attribute must be given for each instance of the element type.
- #IMPLIED: the attribute is optional.
- #FIXED *value*: a value which is allowed for the attribute type.

Figure 3.2 shows a DTD that captures the XML document in Figure 3.1.

```
<!ELEMENT bookstore (book)+>
<!ELEMENT book(isbn , title , author , year , price)*>
<!ATTLIST category CDATA #REQUIRED>
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

Figure 3.2: A DTD of an XML document in Figure 3.1.

3.2.2 XML Schema

Although DTDs have served well for years as the primary mechanism for describing structured information in the SGML and HTML communities, they are considered too limited for many data-interchange applications [31]. For example, DTDs can only specify that elements are text strings. Furthermore, they are not formulated in XML syntax and provide only very limited support for types or name spaces. This led to the XML Schema being introduced to overcome some of the deficiencies of DTD [66]. XML Schema is a data definition language for XML documents which has become a recommendation of W3C. *XML Schema Definition (XSD)* is an XML-based grammar declaration for XML documents. XML

Schema allows very precise definition for both simple and complex data types, and allows the derivation of new type definitions.

The purpose of XML schema is to specify the structure of instance elements together with the data type of each *element/attribute*. Declarations in XML Schema can have richer and more complex internal structures than declarations in DTDs. The motivation for XML Schema is dissatisfaction with DTDs. It was developed in response to the limitations of the DTD mechanism. XML Schema is seen as an advance over DTD. The integration with namespaces is one of the important items missing in DTDs. A DTD can define any number of tags, but there is no way to associate tags with a namespace. An XML schema document describes the structure of XML documents. It begins with a declaration of the namespaces to be used in the schema. Its main features are:

- It uses the same syntax as used for an ordinary XML Schema.
- It is integrated with the namespace mechanism which means different schema can be imported from different namespaces and integrated into one schema.
- It provides built-in types, such as string, integer, and time.
- It provides the means to define complex types from simple ones.
- It supports key and referential integrity constraints.

Figure 3.3 is an XML schema definition that captures the Figure 3.1 document.

The root element of the XSD is the *<schema>*. Within the *<schema>* element, the namespace declaration should be included first, and then an *<element>* declaration. We declare *bookstore* as an element of a user-defined type that contains a sequence of ordered elements each of a new type. Each user-defined type can be declared as either a complex type or simple type form.

To specify the cardinality of the elements, XML Schema uses standard modifiers: *minOccurs* and *maxOccurs*, that correspond to minimum and maximum values for the lower and upper bounds respectively in terms of occurrence of the element.

```
<?xml version="1.0" ?>
<xs:schema xmlns="http://www.w3.org/2001/XMLSchema">
 <xs:element name = "bookstore">
  <xs:complexType>
   <xs:sequence>
    <xs:element name = "book" maxOccurs = "unbounded">
     <xs:complexType mixed="true">
      <xs:sequence>
       <xs:element name = "isbn" type = "string"/>
       <xs:element name = "title" type = "string"/>
       <xs:element name = "author" type = "string"/>
       <xs:element name = "year" type = "integer"/>
       <xs:element name = "price" type = "decimal"/>
      </xs:sequence>
      <xs:attribute name="category" use="required" type="string"/>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

Figure 3.3: An XML schema of an XML document in Figure 3.1.

3.3 XML application programming interfaces

XML documents have to be parsed in order to be used by application programs. *Application Programming Interfaces (APIs)* are used to process an XML document by accessing its internal structure. There are three major standardized ways for users to access the content of XML documents: *DOM*, *SAX*, and *JDOM*.

3.3.1 DOM

The *DOM (Document Object Model)* [6] is an application program interface *(API)* for XML instances defined by W3C. It is a tree structured-based API which converts the document that defines an abstract data type which implements the abstract XML tree model for storing and managing XML instances. DOM is a set of Java interfaces which describe the facilities for a programmatic representation of a parsed XML document. Using DOM, the Web document is modeled in an *object-oriented* way. That is, the DOM represents a Web document in terms of *objects* (the parts of the document, such as *elements*, *attributes*, *text*, etc.). A *document builder* is used to read the XML data and construct a DOM tree. Once a document is read, its DOM representation has been created in memory, and the objects can be accessed and manipulated.

An XML document is read by an XML processor (or XML parser), which converts it into a parsed XML document, which is the internal representation of the hosting implementation (i.e., a DOM implementation). Client applications access the parsed document by means of the functions and methods defined in the DOM API [146].

3.3.2 SAX

The *SAX (Simple API for XML)* is an *event-driven* and *serial access* API defined by XML-DEV group for accessing XML documents [116]. Since SAX is simple, it is supported by most of the available XML processors (parsers). A SAX parser does not create an in-memory tree representation of an XML document. It reads an input XML document and generates *events*, such as the start of an element, the end of an element, and so on. SAX is not an XML processor as such, but it is a common interface implemented for many different XML processors.

Once an XML document is input to the SAX parser, the first step of a SAX parser usually consists of splitting up the source document into tokens. The most basic way to tokenize a document is to use the occurrences of the brackets: < and > as an orientation. Furthermore, the programmer has some control over low-level features like character sets that are used in the document.

3.3.3 JDOM

JDOM (Java Document Object Model) is a new and open source XML API [88, 89]. It is lightweight and fast, and is optimized for the Java developer so that they can read, change, and write XML data much more easily than before. JDOM integrates well with both DOM and SAX, and takes the best features from them. It is designed to perform quickly in a small-memory footprint. JDOM also provides a full document view with random access but, surprisingly, it does not require the entire document to be in memory. Additionally, JDOM supports easy document modification through standard constructors and normal set methods. Therefore, JDOM has the ability to interoperate seamlessly with existing program components built using SAX or DOM. JDOM documents can be built from XML files, DOM trees, or SAX events. It is also possible to create a JDOM document from scratch. Moreover, it provides support for the XML namespaces and validation at the same time. With other XMP APIs, that is not possible.

3.4 XML query languages

This is an overview of XML query languages. Data extraction, transformation, and integration are all well-understood databases problems concerned with managing data. Their solution relies on a query language, either relational (SQL) or object-oriented (OQL). These query languages do not apply immediately to XML, because XML data differs from traditional relational or object-oriented data. We introduce XPath, XQL XML-QL, and Quilt and then present a brief introduction to XQuery language.

3.4.1 XPath

Until recently, most query languages in the XML world were based on XPath (XML Path Language). We describe XPath 1.0, and then give an overview of the features of its successor XPath 2.0.

3.4.1.1 XPath 1.0

XPath 1.0 [44] is a specification that defines how a specific item within an XML document can be located. The primary purpose of XPath is to address parts of an XML document. XPath models an XML document as a tree of nodes which includes *element nodes*, *attribute nodes*, and *text nodes*. It is designed to be employed by most XML query languages. It also provides basic facilities for manipulation of strings, numbers, and Booleans in the logical structure of an XML document. XPath is intended to be simple and efficient. It is based on the idea of path expressions. An expression is evaluated to one of the following basic value objects:

- *node-set* (an unordered collection of nodes without duplicates),
- *boolean* (true or false),
- *number* (a floating-point number),
- *string* (a sequence of characters).

One important kind of XPath expression is the location path. A location path declaratively selects a set of nodes from a given XML document. The result of the evaluation of a location path is the node-set containing the nodes selected by the location path. The core rules of XPath are shown in Figure 3.4.

```
[1] LocationPath ::= RelativeLocationPath | AbsoluteLocationPath

[2] AbsoluteLocationPath ::= '/'RelativeLocationPath?

[3] RelativeLoctionPath ::= Step | RelativeLocationPath '/' Step
```

Figure 3.4: The core rules of XPath.

A location path can be written in the following form:

$$\varphi Step_1\ T_1\ Step_2\ T_2\ \ldots\ldots\ Step_m\ T_m$$

where φ can be the *empty symbol* or '/', T_i is '/', $Step_i$ is a location step, such that: $m \geq 1$, and $i \in \{1, \ldots\ldots, m\}$.

The input to every location step is a *node-set*, called the *context* (the input to the first step is the set containing only the document node). From this set, a new node-set (called the *result set*) is computed which then serves as input for the next step. For this computation, the input node set is processed, evaluating the location step for every node in it, appending its result set to the overall result, and proceeding with the next node. The location step is of the form:

*axis :: nodetest[filter]**

which specifies that navigation goes along the given axis in the XML document. The first part of a step is the *axis* which specifies the tree relationship between the nodes selected by the location step and the current context node. The second part is the *nodetest*. It specifies the node type and the name of the nodes to be selected by the location step which satisfy the given filter. The third part is the *filter*. The filter contains *predicates* over expressions. A predicate filters a node set with respect to an axis to produce a new node set.

The semantics of XPath expressions is defined in terms of node-sets, i.e., unordered forests. When evaluating individual steps, there is a temporary node list (context). For every navigation step, the axis specifies the direction of navigation in the tree. All *forward axes* enumerate the nodes in document order; on the other hand all *backward axes* enumerate them in reverse document order. For example, terms of predicates over the expressions can be of the following forms:

- Booleans over predicates,
- Arithmetic expressions over numbers and string operations,
- Function calls: used for instance to state conditions on the relationship between the current *context node* and its *context*, for example:
  - Last(): returns n such that n is the size of the context,
  - Position(): returns the index of the context node in the current context,
  - Count(*nodeset*): returns the number of nodes in a nodeset,
  - Id(*expr*): returns the node(s) in the current XML instances whose id(s) result from evaluating expr with respect to the context node.

Inside filters, *relative* or *absolute* location paths can be used. Relative location paths are evaluated with respect to the current context node. Where an absolute location path begins with "/", which is similar to the

UNIX directory notation, they are evaluated with respect to the root node of the XML document.

The following XPath query example, finds the students who have taken the Database Systems course within the current document:

//Student[*CrsTaken/CrsName* = *"Database Systems"*].

3.4.1.2 XPath 2.0

XPath 2.0 [26] is a superset of XPath 1.0. It uses the same axis as XPath 1.0, followed by a node test, followed by a predicate. However, XPath is an expression language that allows processing of values conforming to its model, it supports sequences instead of node-sets. The result of an XPath expression may be a selection of nodes from the input documents, or an atomic value, or more generally any sequence allowed by the data model. Thus, every XPath expression evaluates to a sequence. Here, the sequence can be defined as follows:

- A sequence is an ordered collection of zero or more items.
- An item is either an atomic value or a node.
- An atomic value is a value in the value space of the XML Schema.
- A node is defined in the XQuery and XPath 2.0 data model.
- A sequence of exactly one item is called a singleton sequence.
- A sequence containing zero items is called an empty sequence.

There are many differences between XPath 1.0 and its successor XPath 2.0. XPath 1.0 does not support explicit quantification, e.g. to concatenate the first and last names. The most basic additional features for XPath 2.0 are:

- For-loop expression,
- If-Then-Else conditions,
- Functions,

- Quantified expressions,
- Logical expressions,
- Processing Instructions, and
- Schema validation.

In total, although XPath 2.0 has many additional features compared to XPath 1.0, it also has some limitations. The basic one is that it returns XML tree nodes and not an XML document. Despite its interesting features, XPath is not an expressive query language. Compared with the relational algebra, a full join operator is missing (semi-equijoins are in fact provided by the path operator, and filters). As a result XPath is a lightweight XML querying language. It is only an addressing mechanism which selects node sets in XML documents. Its purpose is to provide the common addressing mechanism for XML, and to serve as a base for XML querying and manipulation languages and further concepts.

3.4.2 XQL

The *XML Query Language (XQL)* [7, 127] was an early proposal for a simple querying language which was designed specifically for XML documents. It is a declarative rather than procedural language.

The basic idea and syntax is to use paths and filters for navigation. Roughly, XQL is the fragment of an XPath which can be built without using axis. Additionally, union and intersection on results are allowed. An expression in XQL is always evaluated with respect to a search context. A search context is a set of nodes through which an expression may search to yield the value of the expression. All nodes in the search context have the same parent name. XQL allows a query to select between using the current context and the root context as the input context. A query prefixed with '/' uses the root context, while './' is used for the current context. A query may use the '//' operator to indicate recursive descendent. The prefix './/'

allows a query to perform a recursive descent relative to the current context.

Some XQL queries are:

- to find all author elements anywhere within the current document: *//author*
- to find all title elements, one or more levels deep in the bookstore: *bookstore//title*
- to find the format attribute for all book elements: *book/@format*
- to find all author elements that contain at least one degree and one publication: *author[degree and publication]*
- to find all authors containing a first-name child whose text is 'John': *author[first-name!text()= 'John'*

The central extension, making XQL a *query language* instead of a pure addressing mechanism, is the generation of the result tree (instead of a node-set) as a projection of the input document.

3.4.3 XML-QL

XML-QL [50, 51] is another early language (1998, non-W3C) that was proposed as a query language for XML data. It can express queries which extract pieces of data from XML documents, as well as transformations. In contrast to XQL, XML-QL does not employ navigation and paths. The basic idea was influenced by the SQL query structure in that it partitions XML-QL queries into a selection part *(WHERE IN)* and a construction part *(CONSTRUCT)* it has the form:

WHERE xml-pattern$_1$ IN url
CONSTRUCT xml-pattern$_2$

where, *xml-pattern$_1$* is matched against an XML instance given by the *url*. Therefore, every match yields variable bindings which are used as join variables, and propagate the result. Hence, this is again an XML pattern specifying the result. Similarly to SQL, the *CONSTRUCT* part may contain nested XML-QL queries. Here the *IN* part applies, either to a *url*, or to the *content* of a variable which has been assigned in the *WHERE* part.

The following example produces all authors of books whose publisher is Addison-Wesley in the 'bib.xml' document.

```
 WHERE <book>
   <publisher> <name> Addison-Wesley </></>
   <title> \$t </>
   <author> \$a </>
</> IN "bib.xml" CONSTRUCT <result>
   <author> \$a </> <title> \$t </>
</>
```

The above query matches every *<book>* element in the "bib.xml" XML document which has at least one *<title>* element, one *<author>* element, and one *<publisher>* element whose name is "Addison-Wesley". For each such match it returns both *<author>* and *<title>* and groups them in a new *<result>* element.

XML-QL can map XML data between DTDs and can integrate XML data from different sources. Therefore, we can query several sources simultaneously and produce an integrated view of their data. The query in Figure 3.5 is introduced in [49]. It produces all the pairs of names and social-security numbers of the employees by querying the sources *'www.a.b.c/data.xml'* and *'www.irs.gov/taxpayers.xml'*. The two sources are joined on the social-security number, which is bound to *$ssn* in both expressions. The result contains only those elements that have both a *name* element in the first source and an *income* element in the second source.

```
WHERE <person>
   <name></> ELEMENT_AS \$n
   <ssn> \$ssn </>
 </> IN "www.a.b.c/data.xml",
 <taxpayer>
  <ssn> \$ssn </>
  <income> </> ELEMENT\_AS \$i
 </> IN www.irs.gov/taxpayers.xml
CONSTRUCT <result> \$n \$i </>
```

Figure 3.5: The XML-QL query.

3.4.4 The Quilt query language

Quilt [39] is a query language for XML. It is the base of XQuery which will be discussed next. Quilt is the first XML query language that embeds XPath syntax into higher-level constructs similar to SQL/OQL [17]. Quilt can operate on flat structures, such as rows from relational databases, and generate hierarchies based on the information contained in these structures. It is able to express queries based on document structure and to produce query results that either preserve the original document structure or generate a new structure. It can also express queries based on *parent/child* relationships or document sequence, and can preserve these relationships or generate new ones in the output document.

Quilt queries consist of a series of clauses that declaratively describe:

- what information is to be used,
- which additional conditions apply, and
- how the result is to be constructed.

The structure of Quilt queries as a whole is very similar to XML-QL (*WHERE xml-pattern IN url CONSTRUCT xml-pattern*). The main difference is that the extraction part in XML-QL also uses an XML pattern which is matched whereas Quilt uses iteration and collections over XPath expressions.

A simple form of a Quilt query consists of FOR, WHERE, and RETURN clauses. The *FOR-clause* uses XPath expressions for binding the values of one or more variables. In general, an XPath expression evaluates to a set of nodes. The FOR-clause generates an ordered list of tuples, each containing a value for each of the bound variables. A tuple is generated for each possible way of binding the list of variables to nodes that satisfy their respective XPath expressions. When a node is bound to a variable, its descendant nodes are carried along with it. The WHERE-clause applies a filter to the tuples and retains only those tuples that satisfy a given search condition. The RETURN-clause then generates a new document structure using the values of the bound variables.

The following example finds every book written by Crockett Johnson. The FOR-clause generates a list of bindings. First, the *$b* variable is bound to individual book elements in the document found at the given URL. Then, the *$a* variable is bound to individual author elements that are descendants of *$b*. The WHERE-clause retains only those tuples of bindings in which the author is Crockett Johnson, and the RETURN clause uses the resulting values of *$b* to generate a list of books. By default, the ordering of book elements in the original document is preserved.

```
FOR $b IN
  document("http://www.biblio.com/books.xml")//book,
  $a IN $b/author
WHERE $a/firstname = "Crockett" AND $a/lastname = "Johnson"
RETURN $b
```

Additionally, Quilt is supported with FOR-LET-WHERE-RETURN-clause (*FLWR-expressions*) which is of the form:

```
FOR variable IN xpath-expr
LET additional-variable: = xpath-expr
  (FOR | LET)*
WHERE filters
RETURN xml-expr
```

Bounded variables can be defined by a FOR-clause to the elements which are iterating over the result set of XPath expressions. Additional variables may be defined in the LET-clause, computed from the ones defined in the FOR-clause. The variables in the FOR-clause iterate over the corresponding *xpath-expr*, whereas the variables in the LET-clause are bound to the *result* of the corresponding *xpath-expr*. Variables defined in the FOR-clause or LET-clause can then be used in subsequent IN clauses. The result from the FOR-LET clauses is sequences of variable binding used in generating the result, using the XPath filter syntax. Then the RETURN-clause generates an XML sub-tree for each variable binding.

The Quilt language provides the usual operators used in database queries. Quilt allows for joins in the FOR-clause by specifying "*var IN xpath-expr*" arguments, or by a sequence of FOR-LET clauses. Each FOR-LET clause may contain references to variables defined before. Additional *join* functionality is provided by using FLWR expressions in the RETURN part. Here, also the inner FLWR expression may access variables from the outer clause.

In Quilt, *selection* functionality is explicitly provided by the WHERE-clause, but also the XPath expressions (filters, extensional semantics) provide functionality which is implemented in SQL by selection.

Projection is supported by the definition of variables in the FOR-LET clauses, and mainly by a FILTER operator which extends the XPath syntax of the expression in the form:

xpath - expr$_1$ FILTER xpath - expr$_2$

Results in a tree which contains exactly the nodes of the result set of *xpath-expr$_1$*, retaining the document structure and order. Nodes are taken without attributes or sub-elements, i.e., only the tags are kept.

Quilt is a subset of XQuery. It provides the user with the ability to use built-in functions and user-defined functions. These are very important features which are used in our approach in order to resolve logical heterogeneity problems. The examples given in Section 6.5 show the importance of such functions and how they are used.

3.4.5 XQuery

XQuery [29] is a potential standard XML query language. It is a powerful XML query language derived from Quilt. With some minor revisions, Quilt query language has become the XQuery Language (Feb. 2001 Working draft). XQuery is a full-featured query language. It has borrowed features from several other languages, including XPath 1.0, XQL, XML-QL, SQL, and OQL. XQuery is designed to meet the requirements identified by W3C XML Query Working Group. It is created to be a language in which queries are concise and easily understood. The requirement was for both human-readable query syntax and XML-based query syntax. It is defined as a superset of XPath. XQuery version 1.0 is an extension of XPath 2.0. Therefore, any expression that is syntactically valid and executes successfully in both XPath 2.0 and XQuery 1.0 will return the same result in both languages. The shortcoming in XPath 2.0 is that it returns XML tree nodes and not an XML document, when querying or navigating an XML document usually demands an XML output. Hence, the ability to produce or restructure an XML document is valuable and is offered by XQuery. Therefore, it covers the aspects of both document-oriented and data-oriented documents. Queries in XQuery often combine information from one or more sources and restructure it to create a new result.

XQuery expressions have some similarity with SQL in their structure:

```
FOR variable-declaration
WHERE condition
RETURN result
```

The *FOR-clause* plays the same role as the *FROM-clause* does in SQL, and the *WHERE-clause* is borrowed from SQL with the same functionality, and the *RETURN-clause* is similar to *SELECT.*

For data integration, the documents to be integrated in general use their own namespaces. XQuery allows access to the namespace definitions and assigns them to constants which can then be used for selecting navigation steps according to the namespaces.

XQuery is also a functional language in which a query is represented as an expression. The expression that is most commonly used for combining and restructuring the XML details is the *FLWR-expression* which is the same as Quilt.

Example 3.4.5 the following example of an XQuery query returns the title of all books published in or before the year 2000 and the total number of such books in the bibliography bib.xml document.

```
<books>
FOR $book IN document("bib.xml")/book
LET $titles = $book/title
WHERE $book/@year <= 2000
RETURN
$book/title
<total> count($titles)</total>
<books>
```

In this query, we can see that XQuery expressions are FLWR-expressions. Each FOR iteration binds the *$book* variable. Then, the LET-clause binds the *$title* variable without iteration. Next, the FOR-LET clause filters using the predicate *$book/@year<=2000,* in the WHERE-clause. And finally, the RETURN clause generates the output. In this query, *$book/title* is an XPath expression, and *<books>. . .</books>* wraps the query result into a new XML document.

CHAPTER 4

The SISSD data integration system

This chapter introduces the project. We give a brief introduction to the motivation for this work. Then, we introduce our approach, followed by the proposed system architecture. Next, we describe the heterogeneity problem, and then introduce an application example which is used throughout the thesis to show how the integration is accomplished.

4.1 Introduction

Integrating and querying heterogeneous data sources is a fundamental problem in databases, which has been studied extensively in the last two decades both from a formal and a practical point of view [103]. Recently, this research area has been driven by the need to integrate data sources on the Web, much of the previous research on integration has focused on so called *data integration* [103, 105]. Data integration is the problem of combining the data residing at different sources usually in databases, and providing the user with a unified view of these data, by means of a global (or mediated) schema, over which queries to the data integration system are expressed. A data integration system has to free the user from needing to know which sources contain the data of interest, how such data are structured at the sources, and how such data are to be merged and reconciled to answer user queries [35, 119, 147]. Regarding data integration techniques we differentiate between the logical and physical stages. The first stage integrates schemas from multiple data sources. The result of this schema integration process is the *mediated schema* and *mapping rules* which define how to map concepts in the data sources onto the mediated schema. Data integration in the physical stage uses these mapping rules to transform users' queries on the mediated schema into local queries [80].

In the research community, to build data integration systems two approaches are used. These both use the following two steps:

1. Accept a query, determine the appropriate set of data sources to answer the query, and generate the appropriate sub-queries for each data source.
2. Obtain results from the data sources, perform appropriate translation, filtering, merge the data, and return the final answer to the user or application.

The first approach is referred to as a *virtual approach*, where data remains in the local data sources. Thus, queries operate directly on the local data sources and data integration takes place during the query processing. This means data is extracted from the data sources only when queries are posed. This process also may be referred to as a *mediator-wrapper approach* [145].

The second approach is called the *materialized approach* [27], since data coming from the local data sources are integrated and stored in a single new database or warehouse. All queries then operate on this comprehensive database. In this approach:

1. Data from each data source that may be of interest to the anticipated users is extracted in advance, translated and filtered as appropriate, and merged with relevant data from other data sources in a logical centralized repository.
2. When a query is posed, the query is evaluated directly at the repository without accessing the original data sources.

This approach is referred to as *data warehousing* since the repository serves as a warehouse storing the data of interest. A data warehouse is a decision support database that is extracted from a set of data sources. The extraction process requires transforming data from the source format into the data warehouse format [63].

The mediator-wrapper approach is used to integrate data from different databases and other data sources. It is appropriate for data that changes rapidly, for clients with unpredictable needs, and for queries that operate over vast amounts of data from a very large number of information sources (e.g., the World Wide Web) [143]. It has two main components: a mediator and one wrapper for each data source. The wrappers are interfaces to data sources that translate data into a common data model used by the mediator. The mediator performs the following actions in the system:

1. Receiving a query formulated on the unified schema from the user.
2. Translating this query into sub-queries to individual sources based on source descriptions.
3. Sending sub-queries to the wrappers of individual sources, which in turn transform these sub-queries into queries suited to the source's data model and schema.

The construction of a mediator can be classified into two main types, namely *structural approaches* and *semantic approaches*. In structural approaches, local data sources are the main source of information when the mediated schema is constructed. The integration is done by providing or semi-automatically generating a global unified schema that characterizes the underlying data sources. On the other hand, in semantic approaches, the integration is achieved by using a common ontology covering the domain of the data sources to identify the elements in the local schema that should be linked.

Our objective is to facilitate the designer in building structured and semi-structured data integration systems. Providing a reasonable framework for database integration designers to effectively integrate and query heterogeneous distributed structured and semi-structured data has become another challenge for databases integration researchers [15]. The main difficulty in this task lies in the lack of a fully automated schema-mapping process. The key problem in creating this arises from the existing high degree of logical heterogeneity between the source schemas. This means, it is necessary to resolve several conflicts caused by the heterogeneity of the data sources with respect to the common data model, schema or schema concepts. Therefore, the mapping between entities from different sources representing the same real-world objects has to be defined. This task is not easy since the data at different sources may be represented in different formats and in incompatible ways. For example, the bibliographical databases of different publishers may use different formats for authors' or editors' names (e.g., full name or separated first name and last name), or different units of prices. Moreover, the same expression may have a different meaning (homonym problem), and the same meaning may be specified by different expressions (synonym problem). This implies that syntactical data and metadata alone can not provide enough semantics for all potential integration purposes. Another difficulty impeding structured and semi-structured data integration is the query translation process. This is one of the most important problems in the design of a data integration system, in that the system should be able to reformulate the query that is posed in terms of the global schema into a set of queries suited to the local data sources.

The data integration process is often very labour-intensive and demands more computing expertise than most application users have. Therefore, semi-automated approaches seem the most promising, where mediation engineers are given a tool with which to describe the mappings between the integrated schema and local data source schemas, to produce a uniform view over the local databases [120, 148].

4.2 An overview of our approach

In general, building data integration systems requires addressing several different issues. In this thesis, we concentrate on two basic issues:

1. Establishing a Knowledge Base to describe the mappings between the integrated view (master view) and the participating data sources.
2. Processing user queries expressed over the master view into queries suited to the local data sources.

As, we are restricting our attention to integration systems which combine structured data residing in relational databases and semi-structured data held in well-formed XML documents. The system will provide the user with an integrated view (master view) over heterogeneous distributed structured and semi-structured data sources; such an integrated view will be best represented by XML because the advantages of XML as an exchange model, such as rich expressiveness, clear notation and extensibility. The system will enable users to query its data sources in a uniform way. Although fully automatic data integration may not be possible in the dynamic environment that we have considered, we should be able to achieve a high degree of automation which requires only some human intervention by using semantic mapping. Since a fully automatic process for data integration is infeasible, we propose an approach that can be used as an assisting tool to reduce the total designer effort in building data integration systems. Therefore, the issues of establishing a suitable Knowledge Base and processing a user queries have to be addressed. A basic concept will be resolving the logical heterogeneity problem which may occur among the schema's elements. To achieve this task, we will follow a mechanism in which the correspondences among the schemas' elements are expressed through a set of mappings. These mappings are a powerful tool for expressing the correspondences between the schemas, and capturing and overcoming the heterogeneity of the various data sources. Mappings are usually able to bridge these differences.

The integration architecture we have adopted in the project is based on a mediator architecture (see Figure 4.1). The system prototype is called SISSD (System to Integrate Structured and Semi-structured Databases). It requires the generation of a tool for a meta-user (who does the metadata integration) to describe mappings between the master view and local data sources. It produces an XML Metadata Knowledge Base (XMKB) to capture the mapping information, which is then used to generate the sub-queries to local data sources from user queries posed over the master view. These tasks are performed through a mediation layer. Such a layer is introduced to manage the following:

1. Establishing and evolving an XML Metadata Knowledge Base (XMKB) incrementally to maintain the mapping information between the master view and the local data sources participating in the integration system [16].
2. Querying heterogeneous distributed structured and semi-structured data sources in terms of the master view [14].

This is achieved in three steps. First, the data source metadata is extracted and a Schema Structure Definition (SSD) is built for each participating data source. The SSD is the description of the data source metadata in XML format. We do not aim to capture all details of the data source metadata, but rather to capture its essential features and abstract only the structure of the data source which meets the basic requirements of our approach. This is achieved through an automatic process that accesses the specified data source without violating its local autonomy. Then, its metadata is detected and extracted to build a local view (Schema Structure Definition (SSD)) in XML format for this data source. The resulting view describes the structure of the data source schema using the XML Data Source Definition Language (XDSDL).

The second step performs the task of the mediation layer, by establishing and evolving an XML Metadata Knowledge Base (XMKB) incrementally to assist the Query Processor in mediating user queries posed over the master view to local queries over the distributed heterogeneous data sources. This translates such queries into sub-queries –also called local queries- which fit each local data source. This is achieved through a semi-automatic process that generates a tool to assist a meta-user to specify the mappings between the master view and local data sources. We introduce here the XML Metadata Knowledge Base (XMKB) module as the basis of a mediation tool to overcome the heterogeneity problems between data sources. The XMKB module maintains the mapping information between the master view and local data sources' views participating in the integration system. In fact, we model a Schema Structure Definition (SSD) as a tree structure. Thus, each node is identified by its path in the tree, called a *master path* for an element of the master view and a *local path(s)* for the corresponding element(s) of a local Schema Structure Definition (SSD). The relationship between a master path and a local path is created as a mapping. This distinction between elements and paths is important, because an element may occur several times in a schema tree structure with different meanings, while a path always identifies a unique element. Hence, for each path of the master view, the objective is to keep the set of paths that have the same meaning in the local schemas and a user-defined function if it is needed to perform specific operations to overcome representational differences. Such a function is defined explicitly by the designer.

The third step is concerned with the query translation process which is an integral part of the system. A Query Processor module is developed for this process. It transforms a user query into local queries which it then translates by order of this process using the mappings that are defined in the XMKB. In order to obtain local queries for a query issued against the master view, the system must identify the data sources relevant to a given query. The basic idea is that when a query is posed against the master view, called a *master query (global query)*, it is automatically rewritten into sub-queries, called *local queries*, which are appropriate to each local data source's format using the information stored in the XMKB. This task is accomplished by the Query Processor module. The XMKB contains the path information and functions to be applied for each local source. The path expressions in a master query are parsed by the query parser and replaced by their correspondence paths in each local data source. This is achieved by consulting the XMKB to check if there are correspondence paths for the given query. If not, a null query is generated for the corresponding path in the local query, which means that this query cannot be applied to that local data source. Each local query generated will be sent to its corresponding local source, which will execute the query and return the result for the master query.

4.3 The SISSD architecture and Components

In this section, we present an overview of the SISSD architecture and summarize the functions of its main components. The architecture we adopted is shown in Figure 4.1. At the bottom layer there can be any number *n* of heterogeneous structured (e.g., relational database) and semi-structured (e.g., XML document) data sources, where $n \in \{1, \ldots\ldots, m\}$. The XML documents can be a well-formed XML document with no referenced DTD or XML schema, where the associated metadata are buried inside the data, and also can be XML documents with referenced DTDs or XML schemas. However, for our purposes it is the structure of a given XML document that is crucial for data integration. Therefore, we investigate issues related to abstracting the structure of an XML documents in the cases where the sources have no explicitly defined DTDs or XML schemas. At the top layer of our system is the *master view* which is used as a global interface to the participating local data sources. The advantages of XML as an exchange model [72, 113] - such as rich expressiveness, clear notation and extensibility - make it an excellent candidate to be a data model for the master view. At the middle layer, the architecture consists of the following associated modules:

- **Metadata Extractor (MDE):** The MDE needs to deal with heterogeneity at the hardware, software and data model levels without violating the local autonomy of the data sources. It interacts with the data sources via JDBC (Java Database Connectivity) if the data source is a relational database or via JXC (Java XML Connectivity) if the data source is an XML document. The MDE extracts the metadata of all data sources and builds a Schema Structure Definition (SSD) in XML form for each data source. We developed JXC using a JDOM (Java Document Object Model) interface to detect and extract the schema structure of well-formed XML documents that have no referenced DTD or XML schema.

JXC can also deal with XML documents with referenced DTDs or XML schemas.

- **Schema Structures Definition (SSD):** Typically, the heterogeneous data sources use different data models to store their data (e.g. the relational model and XML model). This type of heterogeneity is referred to as syntactic heterogeneity. The solution commonly adopted to overcome syntactic heterogeneity is to use a common data model and to map all schemas to this common model. XML is a good candidate as a common data model for our integrated data model for two reasons: it can represent with ease any type of data whether it is structured in some way or not, XML also fits the context of current web technologies and has rich and powerful tool support [10, 33, 113]. The metadata extracts generated from the data sources by using this data model are referred to as Schema Structure Definitions (SSDs). We define a simple language called XML Data Source Definition Language (XDSDL) for describing and defining the relevant identifying information abstracted from the data structure of a data source. The XDSDL output is represented in XML and is composed of two parts. The first part provides a description of the data source name, location and type (relational database or XML document). The second part provides a definition and description of the data source structure and content. The emphasis is on making these descriptions readable by automated processors such as parsers and other XML-based tools. This language can be used to describe the structure and content of relational databases, well-formed XML documents which have no referenced DTD or XML schema, and XML documents with referenced DTDs or XML schemas.

- **Schema Structure Definition (SSD) & Master View Parsing:** used for reading and parsing the Schema Structure Definition (SSD) and the master view to check the syntactic correctness and test whether it conforms to the XML syntax rules (well-formed).

- **Tree Structure Generation:** used to generate a tree structure for each data source SSD and the master view.
- **GUI Generation:** used to produce a convenient simple GUI form for each schema tree. It is used as a tool to facilitate the generation of the paths mapping.

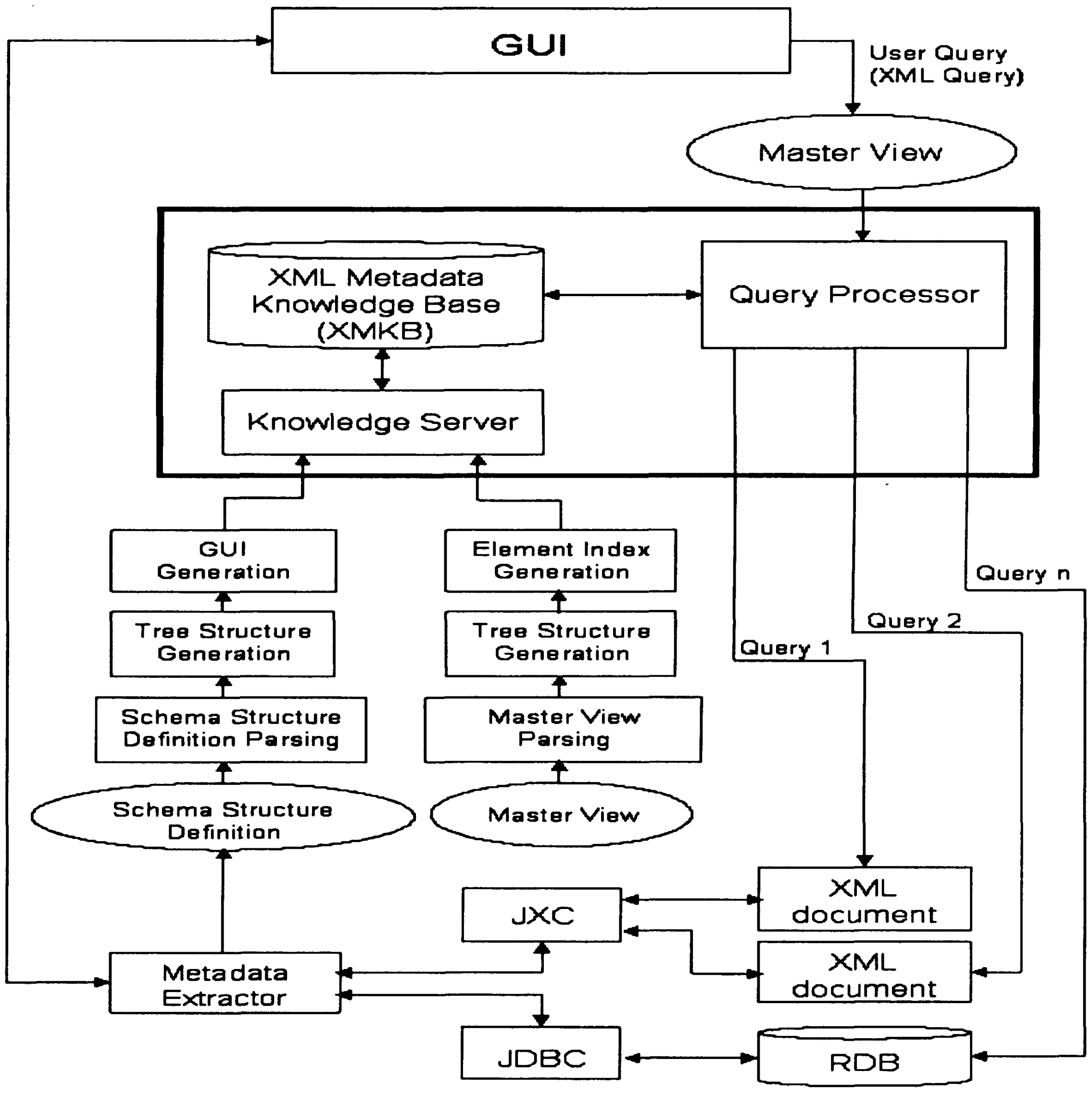


Figure 4.1: The SISSD Architecture.

- **Element Index Generation:** generates automatically a unique index number for each element in the master view tree structure.

- **Knowledge Server (KS):** the central component of the SISSD. It establishes, evolves and maintains the XML Metadata Knowledge Base (XMKB), which holds information about the data sources' structures and semantics and provides the necessary functionality for its role in assisting the Query Processor (QP) module.

- **XML Metadata Knowledge Base (XMKB):** contains knowledge about the data sources' structures and formats represented by XML. It includes the data sources' information (name, type and location) participating in the integration system, the metadata, defining the mappings between the master view and Schema Structure Definitions (SSDs) of the local data sources, and the function names for handling semantic and structural discrepancies.

- **Query Processor (QP):** is responsible for receiving a user query (master query) over a master view processing it and returning the result to the user. It mediates between a user query posed over the master view and the underlying distributed heterogeneous data sources, to automatically rewrite the query into sub-queries - called local queries - which fit each local data source.

4.4 Heterogeneity issues in the SISSD system

In section 2.4 different types of conflicts' that may exist between a set of independently designed data sources are identified. In this section we will show how the fundamental types of these conflicts are resolved in the SISSD system. We choose these types as a representative of each group of these conflicts identified in Figure 2.3. At the end of this section we present the conflict types identified in Figure 2.3 and state the conflict types that can be handled by our system SISSD.

- *Naming conflicts (Table name conflicts, Attribute name conflicts)*: this type of conflict can occur between table names or attribute names when different designers use their own terminologies to describe real world concepts. This may lead to synonym and homonym problems. The former occurs when two different names are used by different designers to describe the same concept. For example one designer may represent a set of employees as element EMPLOYEE in one data source (say DS1), while another designer may represent the same set as element WORKER in another data source (say DS2). In our approach we resolve this type of conflict in the following way. We map elements that are synonyms to the element with the same meaning in the global schema by assigning the same index number generated automatically for the global schema element (more explanation for more information on how these index numbers are generated, see section 5.4.3) to the elements that are synonyms in the local schema structures. A homonym occurs when the same name is used by different designers to represent unrelated concepts. For instance, the element COURSE in DS1 may denote a set of courses taken by a student, on the other hand the element COURSE in DS2 may refer to the available dishes in a restaurant where that student eats. Therefore to resolve this conflict the homonym elements are mapped to different elements in the global schema by assigning different index numbers generated automatically for the global schema elements to the elements that are homonyms in the local schema structures.

- *Unit conflicts:* conflicts of this type arise when two semantically similar elements are represented using different units and measures. For instance, employee salary in two data sources might be represented in UK pounds in one data source and in US dollars in the other. In our approach we resolve this type of conflict, by mapping elements having different units of measurement to appropriate elements in the global schema by assigning index number of that element to the elements in the local schema structures which correspond to it and defining transformer functions which convert data in the different units to the common unit subscribed to by the global schema.

- *Precision conflicts:* conflicts of this type arise when two semantically similar elements are represented using different precisions. For example, Student mark takes an integer value from 1 to 100 in DS1,while Student grade takes a string value of {A, B, C, D, F} in DS2. This type of conflict is usually reconciled by means of a mapping table as shown in Figure 4.2.

Marks	Grades
81-100	A
61-80	B
41-60	C
21-40	D
1-20	F

Figure 4.2: Mapping between Marks and Grades.

In our approach we resolve this type of conflict, by mapping elements having differing precision in their measurements to appropriate elements in the global schema and defining transformer functions to convert data to the type of measurement used by the global schema. In this case, the functions may have to do a table lookup. In this lookup table an isomorphism (mapping) is defined between the different precisions of measurement.

- *One-to-Many Element conflicts:* a special case of a conflict of type one-to-many elements arises when information captured in one element in the global schema is equivalent to the concatenation of more than one element in the local data sources. For example, the name of person is broken into *firstname* and *lastname* in a local data source DS1, while it is simply *name* in the global schema. In our approach, this type of conflict is resolved by mapping the elements in the local data source into corresponding elements in the global schema by assigning the index number generated automatically for the global schema element to the elements in the local schema structures which correspond to it and defining a function to concatenate the elements in the local data source to get the element in the global schema.

- *Many-to-One Element conflicts:* this type of conflict occurs when more than one element in the global schema corresponds to one element in a local data source. For instance, the address information may be represented as *street*, *city*, and *postcode* elements in the global schema, while a local data source DS1 represents it as a single element *address*. In our approach, we resolve this type of conflict by mapping each element containing information about the address in the global schema to the *address* element in the local data source DS1. This mapping is done by assigning the index numbers generated automatically for the global schema elements which contain the information of address to the element *address* in the local data source DS1 separated by comma (,) and assigning derivation functions to be associated with the index numbers assigned to the local data source element to allow these functions to extract the required information from the local data source element.

In Figure 4.3 we summarise the types of conflicts identified in Figure 2.3 that can be handled by our SISSD system and which can't be.

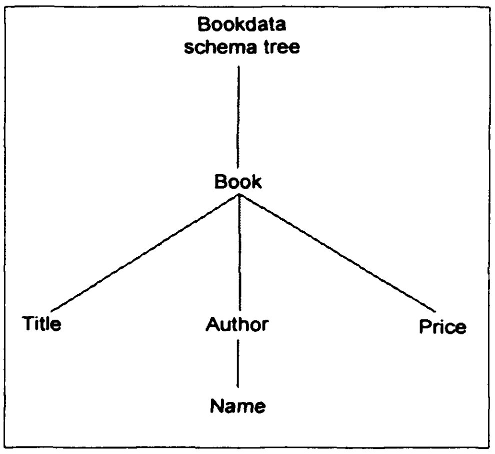
Schema Conflicts	
Conflict Type	**Handled by SISSD**
Table Name Conflicts	Yes
Table Structure Conflicts	Yes
Table Constraint Conflicts	No
Multiple Table Conflicts	Yes
Attribute Name Conflicts	Yes
Multiple Attribute Conflicts	Yes
Table versus Attribute Conflict	Yes
One-to-Many Element Conflicts	Yes
Many-to-One Element Conflicts	Yes
Data Conflicts	
Data Type Conflicts	Yes
Unit Conflicts	Yes
Precision Conflicts	Yes
Expression Conflicts	Yes
Representation Conflicts	Yes
Granularity Conflicts	Yes
Default values conflict	No
Key Conflicts	No
Behavioral Conflicts	No
Wrong data Conflicts	No

Figure 4.3: Summary of Conflicts supported by SISSD system.

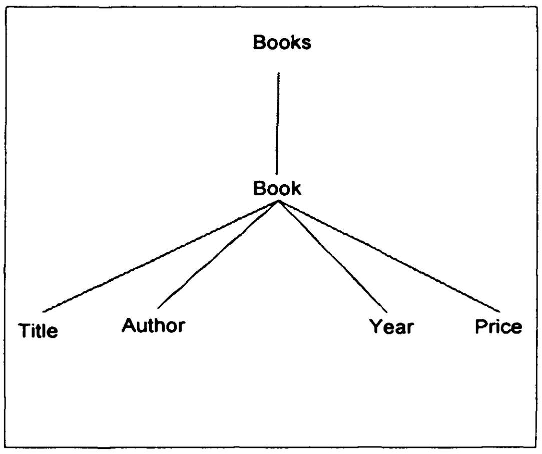
There are types in Figure 4.3 that we have given a yes to but have not described in this section how the SISSD system handles them. This is because these types are more or less similar to the cases described already.

4.5 An application example

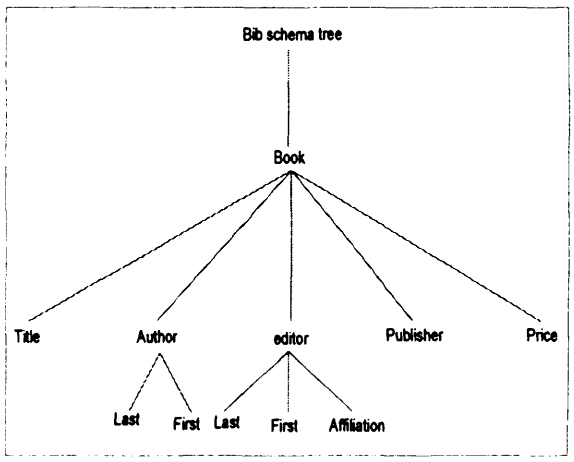
In order to clarify our approach, we introduce an example which will be used throughout to illustrate the key ideas. In a data integration system, we have a set of preexisting data sources which form the application's domain. Each of these data sources may use different schemas to structure their data. Therefore, each data source needs to be mapped to the relevant parts of the mediated schema. In our example four publishers' heterogeneous distributed data source sites are used. Although all these data sources contain information about books, the data structures are different. Our objective is to create a uniform interface over these sites and provide the required functionality to query these data sources in a uniform way. For instance, a teacher or a student may look for a text book for a specific course. In this case instead of posing her/his query to each data source individually, it is possible to pose the query to the unified view over these different data sources.



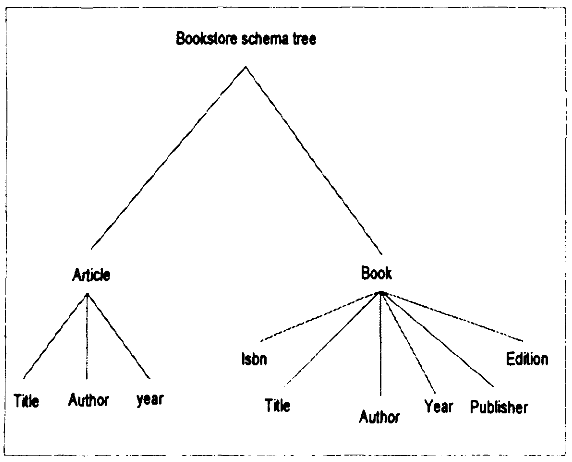
(a): tree structure for bookdata source



(b): tree structure for books source



(c): tree structure for bib source



(d): tree structure for bookstore source

Figure 4.4: A part of the tree structure of four data sources.

In this application example (see Figure 4.4), the referenced data source publishers are a relational database (bookstore.db) and three XML documents (bib.xml, bookdata.xml, books.xml). Each data source contains information about available books, such as titles, authors, prices, and so on. Therefore, the structure of each site was automatically extracted and their Schema Structure Definitions (SSDs) were defined. The Schema Structure Definitions (SSDs) of the participating data sources are described by XML Data Source Definition Language (XDSDL). A part of the tree structures of these data sources are shown in Figure 4.4.

CHAPTER 5

The mediation process

The mediation of distributed heterogeneous structured and semi-structured data sources is proposed as a tool to overcome logical heterogeneity problems which may occur when integrating data sources. It is a basic consideration of this thesis. By mediation, we mean matching the schema elements which are logically equivalent but are represented in different ways. In this chapter we introduce the mediation process, which has the following steps: (1) generate the Schema Structure Definition (SSD); (2) extract SSD components and generate paths; (3) establish the mappings and generate the mediation information (XMKB).

5.1 Generating Schema Structure Definition (SSD)

Our data integration system SISSD supports the integration of distributed heterogeneous structured data residing in relational databases with semi-structured data held in well-formed XML documents produced by internet applications. The SISSD is intended to establish and evolve an XML

Metadata Knowledge Base (XMKB) incrementally to assist the Query Processor in mediating between user queries posed over the master view and the local queries required to access the distributed heterogeneous data sources. The XMKB is established when the first data source is joined to the SISSD system. This is achieved by the same process as joins subsequent data sources, by adding their data to the XMKB. The first step in this process is to construct a Schema Structure Definition (SSD) for this data source. For our purposes it is the structure of the given data source that is crucial for data integration. Therefore, we do not need all the details of the data source metadata, but rather need to capture its essential features so as to abstract only the structure of the data source which meets the basic requirements of our approach. Each data source's Schema Structure Definition (SSD) is described using the constructs of an XML **D**ata Source **D**efinition Language (XDSDL). This is a simple schema definition language which describes and defines the relevant identifying information and the data structure of a data source. This language can be used to describe the structure and content of structured data sources such as relational databases and semi-structured data sources such as the well-formed XML documents.

A data source is called structured if it adheres to a well-defined schema that defines its composition out of other data elements and the schema has the following properties:

- It is defined using a type system.
- It is defined a priori, i.e., before a data element is stored.
- It is explicit, i.e., it is stored separately from the data.
- It is rigid, i.e., the data element must always conform to the structure.

- It is exposed, i.e., it can be queried and used when querying data elements.

Examples of structured data are data stored in relational databases and other databases managed by a DBMS. A query to structured data elements is a structured query and is used to perform a precise search. A structured query is based on the structure of the data elements and the type system as defined in the schema.

A data source is called semi-structured if it has a structure, but the structure is not rigid, and/or the structure definition (or parts of it) is not necessarily separated from the data element, i.e. it may be implicit. The second issue is related to the way the schema is defined. For relational databases, the schema is defined separately, and the data is stored accordingly. For a semi-structured data source, the schema or parts of it might not (and cannot be) defined in this way, and may be "hidden" in the data themselves.

The SSD of a relational database is obtained by:

1. The names of all the tables defined in the DB schema are retrieved.
2. These tables are defined as elements in the target Schema Structure Definition (SSD).
3. For each table, the attribute names are extracted and analysed, and then these attributes are defined as child elements of that table element in the target Schema Structure Definition (SSD).

The structure of the XML document is automatically detected in that each element is found in the document, which elements are child elements and the order of child elements is then determined. The XML document is read and the start tags for the elements are detected. Each start tag is checked, as to whether it has child elements or not: if it has then this element is defined as a complex element in the target Schema Structure Definition

(SSD), otherwise it is defined as a simple element. The defined elements in the target Schema Structure Definition (SSD) take the same name as the start tags.

```
Algorithm: SSD generation for well-formed XML document
Input: well-formed XML document
Output: SSD
Step1: get the root R. If it has child nodes, get its list of children, L.
    a) get the first node in L, N. For every other node N' in L that
       has the same tag as N do:
       • copy and append the list of children of N' to the list of
         children of N.
       • delete N' and its subtree.
    b) get the next child from L and process it in the same way as
       the first child, N, in step (a).
Step2: R now has a new list of children $L_{new}$. Apply step (1) to
    every node $N_{new}$ in $L_{new}$.
```

Figure 5.1: Algorithm to generate SSD for XML document.

The algorithm in Figure 5.1 shows the main steps in the process for obtaining the SSD of a well-formed XML document, where the metadata are buried inside the data.

5.2 paths generation

The Schema Structure Definition (SSD) is itself an XML document. It is a sequence of components where each component is an element of simple or complex type. We model SSD as a tree structure, whose nodes are the components of the corresponding SSD. Each component corresponds to either the occurrence of a tag, or to the content of a tag, and so on. We formulate an SSD Model, through which the SSD can be described. We consider a set of nodes *N* that can be represented as: *E* for *element names*. We do not aim at a complete formalization of all the details of the Schema Structure Definition, but aim to capture its essential model features which meet the basic requirements of our approach. We consider the following functions to be the basic set for characterizing an SSD:

- root: $\varnothing \longrightarrow N$ returns the unique root node of the document,
- children: $N \longrightarrow [N]$ returns the ordered list of children of a node, or the empty list [ ] in the case of a leaf node,
- tag: $N \longrightarrow E$ returns the unique tag (e.g. element name) of a node.

An SSD Model of a Schema Structure Definition is a tree that holds a set of nodes N which can be a set E of *element names*. These elements are either *simple type* T_s or *complex type* T_c, where

$$T = T_s \cup T_c$$

For example in Figure 5.2, a book title is represented by the title element (a simple type, i.e. T_s), while the author element is defined as a complex type (T_c).

```
Element bib
  complexType
    Element book
      complexType
        Element title
        Element author
          complexType
            Element last
            Element first
        Element editor
          complexType
            Element last
            Element first
            Element affiliation
        Element publisher
        Element price
```

Figure 5.2: The SSD Model structure for the bib schema structure.

In order to locate the corresponding nodes of a source's tree structures, we need to generate a unique path for each element of the Schema Structure Definitions (SSDs). Due to the possible occurrence of the same name several times in the same schema tree structure, this path uniquely identifies the node. Hence, naming conflicts can be easily resolved. This is achieved by forming and then searching the SSD tree structure model of each schema and extracting out the components of interest. The SSD path generation process is based on the SSD tree structure model discussed above. The algorithm for schema path generation is shown in Figure 5.3.

```
Algorithm: schema paths generation
Input: SSD schema
Output: SSD paths
Step1: parse SSD;
Step2: for each parsed SSD do
        1. construct an SSD tree structure model M;
        2. perform a depth-first traversal on M:
           - extract the value of each node in set E;
           - give a unique number to each extracted value;
           - construct a CHILD function C for the extracted values;
        3. perform a depth-first traversal on C;
        4. generate a unique path for each node in C;
end do.
```

Figure 5.3: Algorithm to generate SSD paths.

The process of schema path generation is achieved by:

1. The SSD is parsed and its tree structure model formed.
2. The value of each node that belongs to the set E is extracted and a unique number is given to each extracted value.
3. A *CHILD* function is constructed to obtain the children of each extracted value of each node.

Figure 5.4 shows the constructed tree structure model with the unique number given to each node for the *bib* data source. We observe that the node book (*1.1*) is a complex type, with an associated set of children (here represented as an array) *[title 1.1.1, author 1.1.2, editor 1.1.3, publisher 1.1.4, price 1.1.5]*.

```
Element bib                   1
  complexType
    Element book              1.1
      complexType
        Element title         1.1.1
        Element author        1.1.2
          complexType
            Element last      1.1.2.1
            Element first     1.1.2.2
        Element editor        1.1.3
          complexType
            Element last      1.1.3.1
            Element first     1.1.3.2
            Element affiliation 1.1.3.3
        Element publisher     1.1.4
        Element price         1.1.5
```

Figure 5.4: The tree structure model for bib SSD.

The tree structure model is navigated to generate the unique path for each node starting from the root. Figure 5.5 shows a part of the generated paths of the bib data source elements with their numbering. The number of digits in this unique number indicates the element's level in the tree. For example the element with unique number *1.1.3.2* is on the fourth level.

5.3 paths correspondence

Our integration system is based on schema mappings which are used to translate queries posed over the master view into sub-queries - called local queries - which are appropriate to a local data sources. The goal of a schema mapping is to capture structural and semantic as well as terminological correspondences between schemas.

1	/bib
1.1	/bib/book
1.1.1	/bib/book/title
1.1.2	/bib/book/author
1.1.2.1	/bib/book/author/last
1.1.2.2	/bib/book/author/first
1.1.3	/bib/book/editor
1.1.3.1	/bib/book/editor/last
1.1.3.2	/bib/book/editor/first
1.1.3.3	/bib/book/editor/affiliation
1.1.4	/bib/book/publisher
1.1.5	/bib/book/price

Figure 5.5: The generated paths of the bib data source.

The main aim of a data integration system is to allow a user to query distributed heterogeneous data sources. Its users can only view the global schemas while the data is held in the local data sources. Thus, relationships or *mappings* from global schema concepts to local data source schema concepts must be established [109]. Mappings are often specified as high-level, declarative assertions that state how groups of related elements in a data source schema correspond to groups of related elements in the global schema [149].

The schema mapping is defined as a relation $\sum \times \Gamma$, through which each component of the global schema is mapped to a corresponding component of a local schema. These mappings are established by identifying semantically similar concepts (i.e. schema components) in the schemas [139]. In our integration system SISSD, these mappings are used to generate valid local queries. The schema mapping process is not completely automated in our system, since this process required a human intervention to provide some information about how different elements correspond to each other [14, 16].

In general, the major difficulty of integrating different data sources is the establishment of mappings between the global schema and the local data source schemas. We believe that the development of a schema mapping should be based on human user (integrator) interaction. Since two similar terms may refer to different concepts and thus may not have the same meaning, only a skilled human user is able to guarantee the semantic consistency of such a mapping. Consequently, the interactions of human integrators are an essential part of the schema analysis and mapping process [18].

The main approaches to establishing the mapping between each data source schema and the global schema, are classified into two categories: global-as-view (GAV) and local-as-view (LAV) [67, 80, 103].

In the GAV approach, each item in the global schema is defined as a view over the data source schemas. The GAV approach greatly facilitates query reformulation as it simply becomes a view unfolding process. However handling the addition or removal of a data source in a GAV approach is difficult as it requires modification of the global schema to take into account the changes.

In the LAV approach, each item in each data source schema is defined as a view over the global schema. Thus each individual data source must provide a description of its schema in terms of the global schema, making it very simple to add or remove data sources, while making the query reformulation process harder.

Clearly both of these approaches have positive and negative consequences, but LAV is considered to be much more appropriate for large scale ad-hoc integration because of the low impact changes to the data sources have on the system maintenance. Also a LAV approach provides a more flexible environment which can meet users' evolving and changing information requirements across the disparate data sources available over the global information infrastructure (Internet). It is better suited and scalable which suits integrating a large number of autonomous read-only data sources accessible over communication networks. Furthermore the LAV approach provides a flexible environment able to accommodate continual change and update of data source schemas. This makes it especially suitable for XML documents on Web servers, since these remote documents are not static and are often subject to frequent modification. It is also better able to support a dynamic environment, since it allows data sources to be added to or removed from the integration system without the need to restructure the global schema. However, GAV is the preferred approach when the set of data sources being integrated is known and stable [14, 85].

In SISSD, a local-as-view mapping description is used to map between each data source schema and the global schema. This makes handling the addition or removal of a data source easy. When generating the XMKB, the mapping direction is changed from the original local-as-view to a global-as-view, to make query rewriting straightforward. A global query from a user is then translated into local queries on data sources by looking up the corresponding paths in the XMKB. Hence the SISSD combines the virtues of both the GAV and LAV approaches.

In our approach, the designer specifies the global schema (master view), where the basic notions in the application domain are described. The user can alternatively choose the Schema Structure Definition (SSD) of any data source which meets his/her requirements to be the master view, since finding the correspondences among the schemas' elements often depends on the application context. Hence, matching two elements depend on deciding how they correspond to each other, i.e. if they are logically equivalent. This can be determined when they refer to the same real-world entity, or can be inferred by performing specific operations. For example, two elements that share the same name can refer to different real-world entities. The reverse problem also often occurs in that two elements with different names actually refer to the same real-world entity. For these reasons and others, matching is often a subjective process, depending on the application. Hence, a skilled integrator is often involved in the matching process because of the need to interpret the terms' semantics and resolve problems.

Consequently in this project, the process of constructing the global schema is not fully automated. The application domain involves a set of data sources that are associated with heterogeneous schemas. The integration is achieved through a virtual global schema (master view) that characterizes the underlying local data sources. We often use an assumption of paths instead of elements, because the Schema Structure Definitions are trees and each element is identified uniquely by its path in the tree.

For a node *x* in a tree structure, the path *P* to the node *x* is the sequence of nodes from the root node of the tree to the node *x* itself. The path to the first name of an author of a book could be:

P(first_name) = /book/author/full_name/first_name.

Thus to express a correspondence between a global path and a set of paths in a data source schema, we need to study more deeply the semantics of paths. We observe that each path considered can be described by a sentence in natural language, e.g.:

The first name of the name of the author of a given book is John,

where John is the content of the element *first-name* on the path from the root node to the *first-name* node. Hence, as the first step in our data integration system we need to match paths of the Schema Structure Definitions of new data sources with paths of the global schema to determine if they correspond.

Definition 5.1.1 (corresponding paths)

Two paths P_1 and P_2 *correspond,* if and only if, their respective information capabilities are logically equivalent, i.e. $ICap(P_1) \Leftrightarrow ICap(P_2)$. We denote the correspondence relation with $\approx$.

It is clear that building such a framework is hard to automate. Therefore, any decision about the semantic correspondence of paths or sets of paths will be based on an analysis by the integrator.

Using the corresponding paths definition (Definition 5.1.1) we match paths in the global and data source schemas if they correspond. Thus if:

- P_i is a set of paths in the global schema G.
- P_j is a set of local data source schema paths.
- S_k is a set of local data sources.

Then, a set of paths can be matched if they satisfy any one of the following conditions:

1. If $P_i \approx P_j$, such that $(0 < i, j \leq n)$.
2. There is a function $f : G(P_i) \longrightarrow S_k(P_j)$, where $G(P_i) \approx f(S_k(P_j))$, such that $(0 < i, j, k \leq n)$.
3. There is a function $g : S_k(P_j) \longrightarrow G(P_i)$, where $S_k(P_j) \approx g(G(P_i))$, such that $(0 < i, j, k \leq n)$.

Hence, the equivalent paths in schemas are determined by analyzing the information capabilities of each element path. Then, in order to resolve an identified type of heterogeneity conflict, functions *f* or *g* can be provided to perform specific operations which match related elements despite the conflict. These functions are implemented as a user-defined function (UDF). In fact, a UDF definition is not provided during the global schema construction stage, it is defined when developing the query translation module. It is explicitly defined by the designer based on an analysis of path equivalences.

In our example scenarios, the four Schema Structure Definitions use different structures and our goal is to establish mappings between the global schema elements and the local data sources' SSD elements to capture structural and semantic as well as terminological correspondences between the schemas. The analysis of the data sources' elements produces a set of correspondence assertions by using the above correspondence conditions.

Correspondence 1 (C1): an analysis of the information capabilities of the data sources presented in section 4.5, *books Schema Structure Definition* (SSD_1) (see Figure 4.4(b)), *bib* (SSD_2) (see Figure 4.4(c)), *SCMFMA* (SSD_3) (see Figure 4.4(d)) and *bookdata* (SSD_4) (see Figure 4.4(a)) shows that they are logically equivalent. This is a case of naming conflicts in the definitions. Therefore, in order to resolve this conflict we need to establish a mapping between the global path and the corresponding path in the local SSD which has the same meaning. The same number is specified for these paths and must be unique as it identifies the path. The correspondence assertion for these paths is as follows:

$$SSD_1/bookstore \approx SSD_2/bib \approx SSD_3/book \approx SSD_4/bookdata \Longrightarrow global/book$$

Correspondence 2 (C2): the analysis of the information capability of */book/editor/full_name* in the global schema (see Figure 5.13) shows that this path corresponds to both

/bib/book/editor/last and */bib/book/editor/first*

in *bib* (SSD_2). This is a case of structural conflicts in the definitions. In order to resolve this conflict, a function *f* should be provided in which a concatenate operation is performed which merges the first and the last name elements to get the full name. Then, as in C1, a unique number must be assigned to all of the correspondence paths. In this case the function is:

f(SSD_2*/bib/book/editor/last*, SSD_2*/bib/book/editor/first*) $\Rightarrow$
global/book/editor/full_name

Correspondence 3 (C3): in this case, an author name of a book is represented in the global schema as first-name and last-name while in the *books Schema Structure Definition* (SSD_1), the *SCMFMA* (SSD_3) and the *bookdata* (SSD_4) it is represented as full-name. In order to resolve this type of conflict, we need a unique number for these two paths in the global schema which are the first-name path and the last-name path. Also, a function *f* is needed for SSD_1, SSD_3 and SSD_4 which performs an operation to split the full-name value so that it can answer a global query for an author's names. Hence, C3 is formulated as:

f(SSD_1*/bookstore/book/author*) $\approx$ *f*(SSD_3*/scmfma/book/author*) $\approx$
f(SSD_4*/bookdata/book/author/name*) $\Rightarrow$
global/book/author/full_name/first_name,
global/book/author/full_name/last_name

5.4 Creating XMKB

In order to prepare the local queries for a query posed against the global schema (master view); the data sources relevant to a given query must be identified. For this task, the *XML Metadata Knowledge Base (XMKB)* was developed to hold the correspondences between the components of the data sources. For each component of the master view, the objective is to record the set of components having the same meaning in the local data source Schema Structure Definitions and to provide the discrepancy resolution function if it is needed to match the information. The XMKB is used in mediation between the global and local schemas to overcome heterogeneity conflicts in the data sources' schemas and thus to assist the Query Processor in mediating between user queries posed over the master view of the distributed heterogeneous data sources, when it translates such queries into sub-queries which suit a local data source, and to integrate the results from the data sources of the query. In the following subsections, we:

1. Describe the structure of the XMKB;
2. Present the generation process of the XMKB;
3. Describe the mechanism for generating index numbers for the master view (global schema) elements; and
4. Describe the different types mapping between elements.

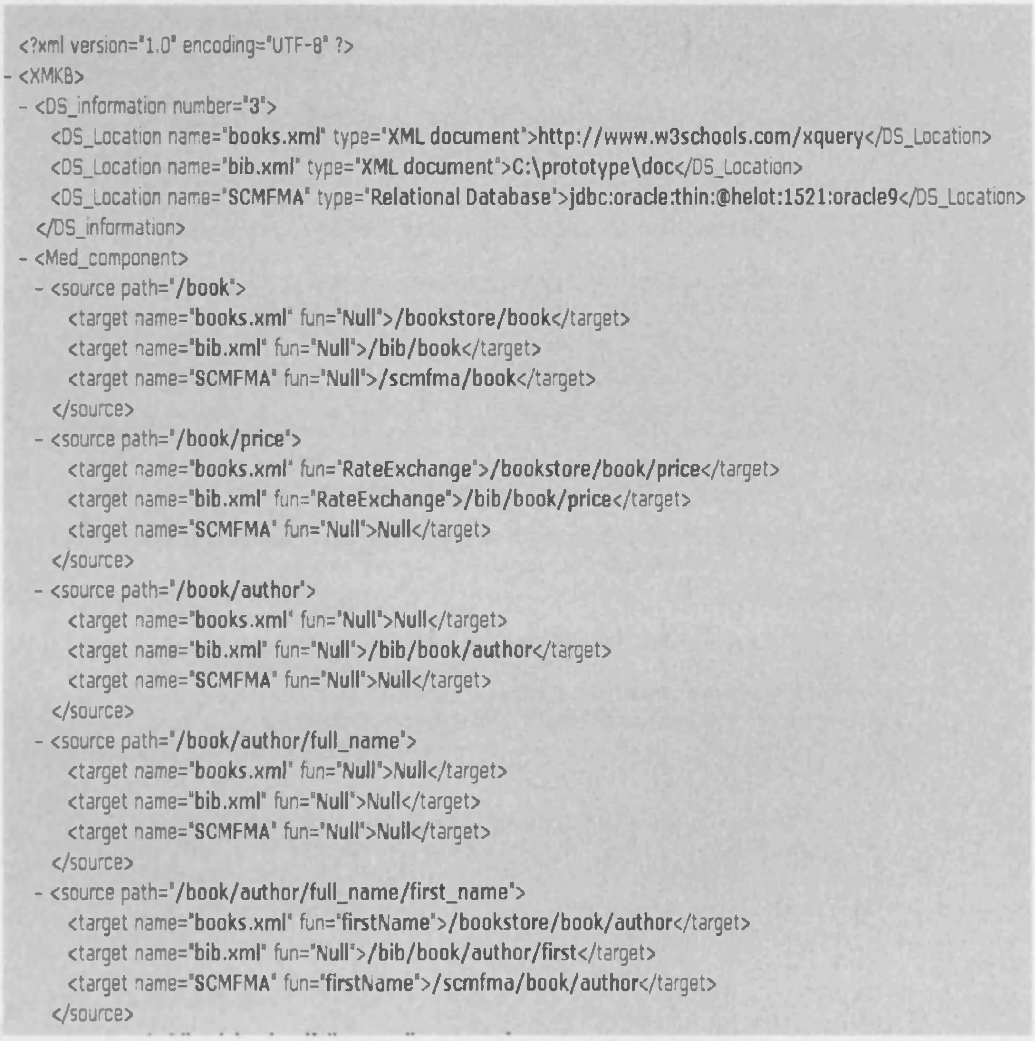
5.4.1 The Structure of XMKB

The XML Metadata Knowledge Base (XMKB) contains several types of metadata about each data source. The first of these types of metadata is a structural and semantic description of the contents of the data sources. It is an XML document composed of two parts. The first part contains information about data source names, types and locations. The second part contains meta-information about the relationships of paths in the data sources, and the function names for handling semantic and structural discrepancies. In the SISSD integration system we have developed a technique for semi-automatic creation of mappings between the mediated schema of the data integration system and data sources. We have defined a simple declarative mapping language called **X**ML **M**etadata **K**nowledge **B**ase **M**apping **L**anguage (XMKBML) for specifying the mapping between the virtual master view elements and the Schema Structure

Definitions (SSDs) elements of the data sources. The XMKBML mapping specifications are written in XML. XMKBML, as a markup language in its own right, provides a vocabulary to describe XMKB mappings. The two main elements in this vocabulary are *DS_information* and *Med_component*. The first one represents the data source's information (names, types and locations), while the second one represents the mapping itself it is created by linking the global paths and the corresponding local paths. Figure 5.6 presents a sample of the XML Metadata Knowledge Base (XMKB) and Figure 5.7 shows the syntax of the XMKBML, given as an XML schema definition. The *<DS_information>* element contains data source names, types and locations; its element has an attribute called number which holds the number of data sources participating in the integration system (3 in the example in Figure 5.6) and also has child elements called *<DS_Location>*. Each *<DS_Location>* element contains the data source name, its type (relational database or XML document) as an attribute value and the location of the data source as an element value. This information is used by the Query Processor to specify the type of sub-query to be generated (SQL if the data source type is a relational database or XQuery if it is an XML document) and the data source location that the system should submit the generated sub-query to.

The *<Med_component>* element in Figure 5.6 contains the path mappings between the master view elements and the local data source elements, and the function names for handling semantic and structural discrepancies. The master view element paths are called *<source>* elements, while the corresponding element paths in the local data sources are called *<target>* elements. The *<source>* elements in the XMKB document have an attribute called path which contains the path of the master view elements. These *<source>* elements have child elements called *<target>* which contain the corresponding paths for the master view element paths in each local data source, or null if there is no corresponding path. The *<target>*

elements in the XMKB document have two attributes. The one is called *name* and contains the name of the local data source, while the second is called *fun* and contains the function name that is needed to resolve semantic and structural discrepancies between the master view element and the local data source element concerned, or null if there is no discrepancy or no available function.

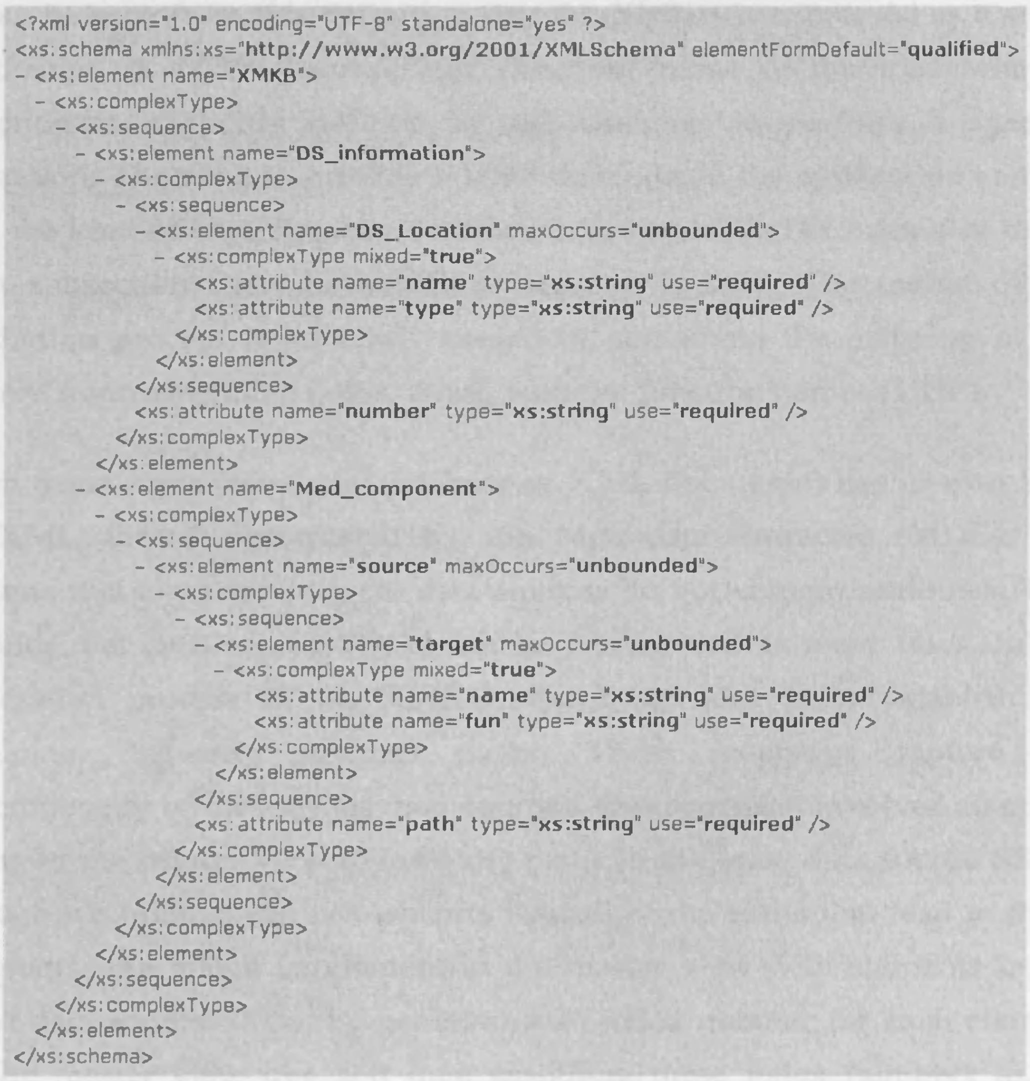


```
<?xml version="1.0" encoding="UTF-8" ?>
- <XMKB>
  - <DS_information number="3">
      <DS_Location name="books.xml" type="XML document">http://www.w3schools.com/xquery</DS_Location>
      <DS_Location name="bib.xml" type="XML document">C:\prototype\doc</DS_Location>
      <DS_Location name="SCMFMA" type="Relational Database">jdbc:oracle:thin:@helot:1521:oracle9</DS_Location>
    </DS_information>
  - <Med_component>
    - <source path="/book">
        <target name="books.xml" fun="Null">/bookstore/book</target>
        <target name="bib.xml" fun="Null">/bib/book</target>
        <target name="SCMFMA" fun="Null">/scmfma/book</target>
      </source>
    - <source path="/book/price">
        <target name="books.xml" fun="RateExchange">/bookstore/book/price</target>
        <target name="bib.xml" fun="RateExchange">/bib/book/price</target>
        <target name="SCMFMA" fun="Null">Null</target>
      </source>
    - <source path="/book/author">
        <target name="books.xml" fun="Null">Null</target>
        <target name="bib.xml" fun="Null">/bib/book/author</target>
        <target name="SCMFMA" fun="Null">Null</target>
      </source>
    - <source path="/book/author/full_name">
        <target name="books.xml" fun="Null">Null</target>
        <target name="bib.xml" fun="Null">Null</target>
        <target name="SCMFMA" fun="Null">Null</target>
      </source>
    - <source path="/book/author/full_name/first_name">
        <target name="books.xml" fun="firstName">/bookstore/book/author</target>
        <target name="bib.xml" fun="Null">/bib/book/author/first</target>
        <target name="SCMFMA" fun="firstName">/scmfma/book/author</target>
      </source>
```

Figure 5.6: A sample XMKB.

5.4.2 The generation process of the XMKB

The building of the XMKB is performed through a semi-automatic process. XMKB is generated by using the mappings between the master view and the local data source SSDs, and it includes the data source's information (names, types and locations), meta-information about the relationships of paths (a path for each node starting from the root) in the data sources, and function names for handling semantic and structural discrepancies. The XMKB is built in a bottom-up fashion by extracting and merging incrementally the metadata of the data sources.



```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
- <xs:element name="XMKB">
  - <xs:complexType>
    - <xs:sequence>
      - <xs:element name="DS_information">
        - <xs:complexType>
          - <xs:sequence>
            - <xs:element name="DS_Location" maxOccurs="unbounded">
              - <xs:complexType mixed="true">
                  <xs:attribute name="name" type="xs:string" use="required" />
                  <xs:attribute name="type" type="xs:string" use="required" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="number" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      - <xs:element name="Med_component">
        - <xs:complexType>
          - <xs:sequence>
            - <xs:element name="source" maxOccurs="unbounded">
              - <xs:complexType>
                - <xs:sequence>
                  - <xs:element name="target" maxOccurs="unbounded">
                    - <xs:complexType mixed="true">
                        <xs:attribute name="name" type="xs:string" use="required" />
                        <xs:attribute name="fun" type="xs:string" use="required" />
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute name="path" type="xs:string" use="required" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 5.7: The XMKB XML schema definition.

The XMKB is an XML document that contains the mappings between related heterogeneous schemas' paths and the required user-defined functions. It can be expressed as follow:

$$XMKB = PATHS \cup FUNCTIONS;$$
$$PATHS: G(P_i) \rightarrow S_j(P_k);$$
$$FUNCTIONS: G(P_i) \rightarrow S_j(UDF);$$

Where UDF = function-name | null, $G(P_i)$ is a set of global paths, S_j is a set of local sources, P_k is a set of source paths. Such that: $i, j, k \in \{1, \dots n\}$.

It can be seen from this definition that the XMKB is expressed as a set of mappings. A *UDF (user-defined function)* name is provided when a function is explicitly defined by the designer to perform a specific operation. The need to provide a UDF depends on the application context and the kind of heterogeneity conflict to be resolved. The examples in the next subsection show how such a function is built. The output of the mediation process is an XML document containing the mapping of the source's corresponding paths, along with the function names (UDF).

Each data source (relational database or XML document) has its own SSD in XML format constructed by the Meta-data Extractor (MDE). We assume that elements in local data sources do not contain attributes. This implies that data source SDDs can be represented as n-ary trees. In the generation process of an XMKB, the basic idea is to establish the mappings between schemas paths. These mappings capture the heterogeneity of the various data sources. Our approach involves mapping paths in the master view to (sets of) paths in the local data source SSDs, though we often speak of elements instead of the paths that lead to these elements. We match an element in the master view with elements in the local data source SSDs, by generating an index number for each element in the master view tree and then assigning these index numbers to the element(s) with the same meaning in the local schema structure trees. Hence elements with the same number have the same meaning. By collecting together all elements with the same numbers, the source and target paths can be generated automatically, and the XMKB can be easily constructed. An especially convenient special case is where an element in the master view exactly matches one in a local SSD, in that its field has the same meaning as the one in the master view. Elements in local SSDs should not appear in the XMKB if their meaning does not relate to any element in the master view.

Constructing the XMKB manually is an error prone and tedious job, so machine support is highly desirable. Hence, we have developed a system that automatically establishes and evolves an XMKB incrementally. This system has been built to act as a tool which assists a meta-user (who does the metadata integration) to describe mappings between the master view and local data sources. This tool parses the master view to generate automatically a unique index number for each element and parses local SSDs to generate a path for each element, and produce a convenient GUI. By using the GUI, index numbers are assigned to match local elements to corresponding master elements and to specify the function names which are needed to resolve any heterogeneity conflicts by performing specific operations. These functions can be built-in or user-defined functions. The XMKB is then generated from the mappings by combination over index numbers. The algorithm in Figure 5.8 shows the main steps in the generation process of the XMKB.

For example, Figure 5.9 presents part of a GUI for the local SSD shown in Figure 5.10. The first column in Figure 5.9 is used to assign the unique index numbers for master view elements to the equivalent elements in the local SSD. Elements without an equivalent index number are not included in the XMKB. The second column is used to specify the function names which are needed for handling semantic and structural discrepancies.

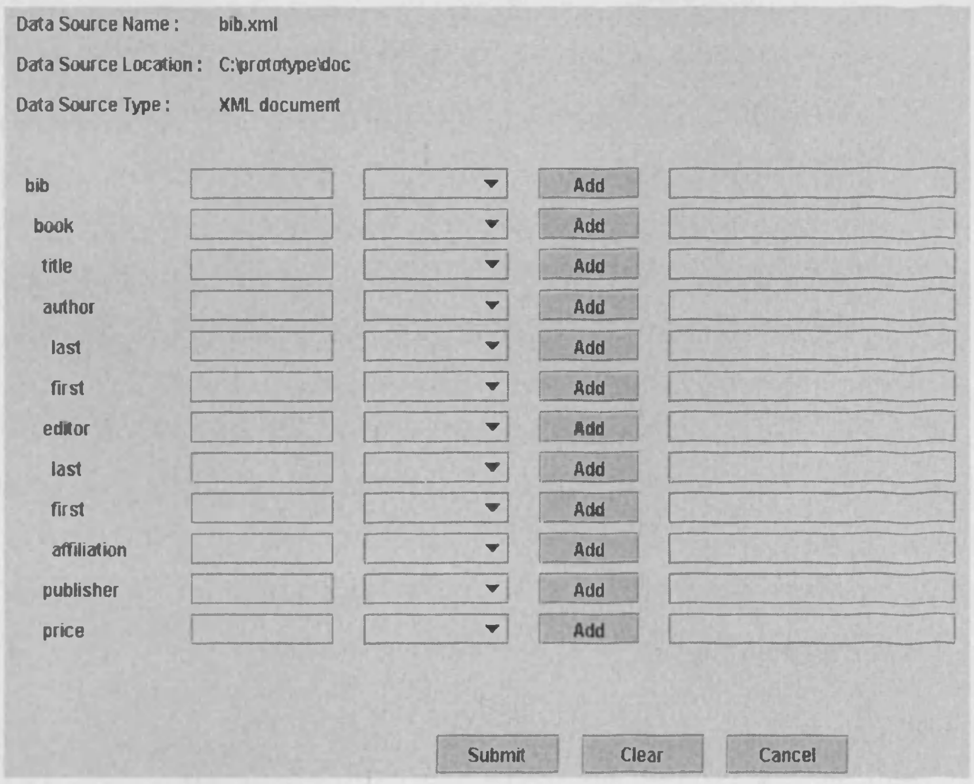
This approach provides a flexible environment able to accommodate the continual change and update of data source schemas, and is especially suitable for XML documents on web servers since these remote documents are not static and are often subject to frequent update. The SISSD gives flexibility to remove any data source schema from the XMKB and then add this data source again with an updated or altered schema without any other impact on the XMKB or the need to regenerate it from scratch.

```
Algorithm: XMKB generation process
Input: master view, data sources Schema Structure Definitions
       (SSDs)
Output: XML Metadata Knowledge Base (XMKB)
Step1: generate a unique index number for each master view
       element;
Step2: ∀ Schema Structure Definitions (SSDs) do
         generate path for each element;
         assign the index number for the equivalent local
                Schema Structure Definition (SSD) paths;
         specify a UDF name if there is an operation required;
       end do;
Step3: collect local paths with their global path, according to the
       assigned Index numbers;
Step4: if the data source is the first one joining to the integration
       system
        then
           establish an XMKB for capturing these mappings
           information;
        else
           update an XMKB for capturing these mappings
           information;
```

Figure 5.8: Algorithm for XMKB generation process.

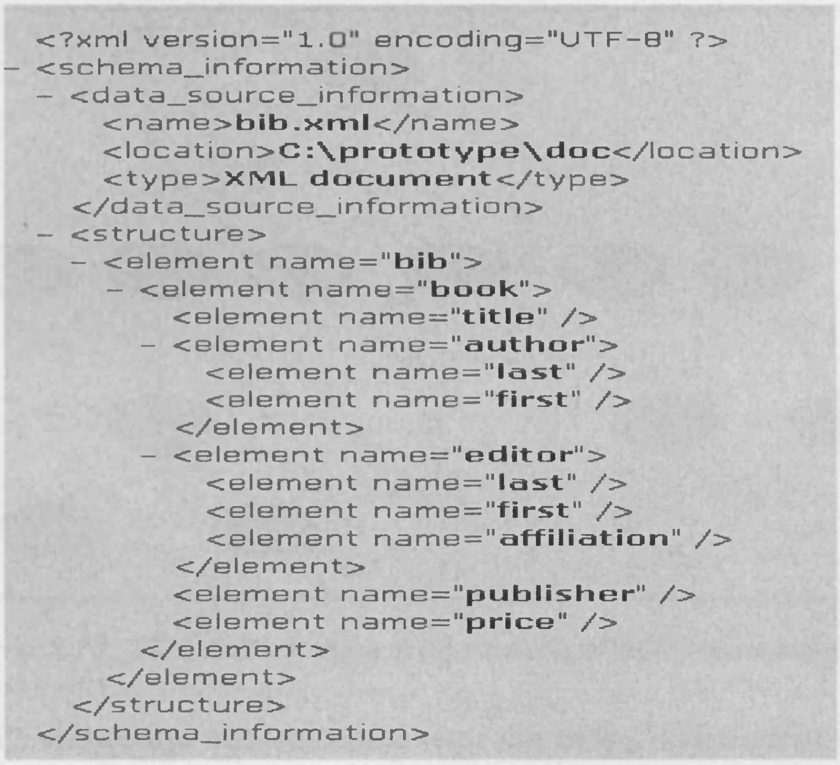
5.4.3 Index number generation for the master view elements

The generated index numbers for the master view elements are used to match local elements to corresponding master elements. We employ a mechanism to generate such index numbers using JDOM technology. By applying this mechanism, a unique index number is generated for each element in the XML document whatever the nesting complexity of the document. This mechanism uses JDOM to read and parse the master view and generate a tree structure for it. By using the tree structure, the root element node of the master view is identified and the number 1 is assigned to it. For each element in the master view including the root element, the children of this element are obtained and then assigned a sequential number starting from 1 for each child to represent the order of the children for that parent. By combining the number given to each child with the index number of its parent separated by dot (.) we produce the unique index number of this child. For example, if the root element has four child elements, the index number of the first child element will be 1.1 and the index number of the second child element will be 1.2, and so on. Furthermore, if the element with index number 1.2.1 has two children, the index number for the first child will be 1.2.1.1 and the index number for the second child will be 1.2.1.2. The algorithm in Figure 5.11 shows the main steps in the generation of the index numbers for the master view.



Data Source Name :	bib.xml
Data Source Location :	C:\prototype\doc
Data Source Type :	XML document

Figure 5.9: A GUI for Schema Structure Definition shown in Figure 5.10.



```
 <?xml version="1.0" encoding="UTF-8" ?>
- <schema_information>
  - <data_source_information>
      <name>bib.xml</name>
      <location>C:\prototype\doc</location>
      <type>XML document</type>
    </data_source_information>
  - <structure>
    - <element name="bib">
      - <element name="book">
          <element name="title" />
        - <element name="author">
            <element name="last" />
            <element name="first" />
          </element>
        - <element name="editor">
            <element name="last" />
            <element name="first" />
            <element name="affiliation" />
          </element>
          <element name="publisher" />
          <element name="price" />
        </element>
      </element>
    </structure>
  </schema_information>
```

Figure 5.10: Schema Structure Definition (SSD) of bib XML document.

```
Algorithm: Index numbers generation process for master view elements
Input: master view
Output: unique index number for each master view element
Step1: parse a given master view and generate a tree structure
       for it;
Step2: identify root element;
Step3: assign number 1 for the root element;
Step4: ∀ elements in the master view do
          get all children of this element;
          assign sequential number starting from 1 for each child;
          combine the number given for each child with the
           index number of its parent separated by dot;
       end do;
```

Figure 5.11: Algorithm to generate index numbers.

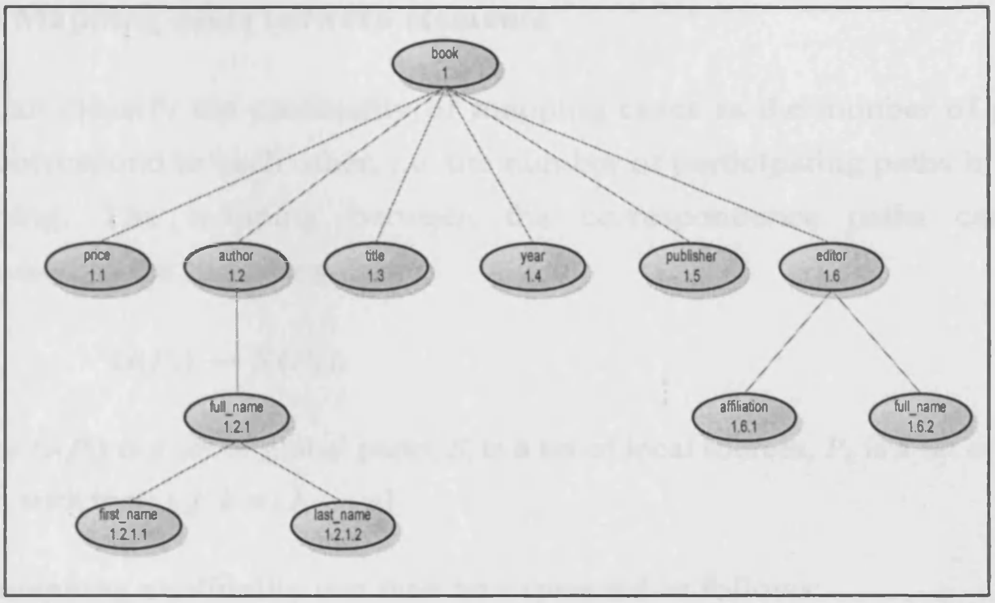
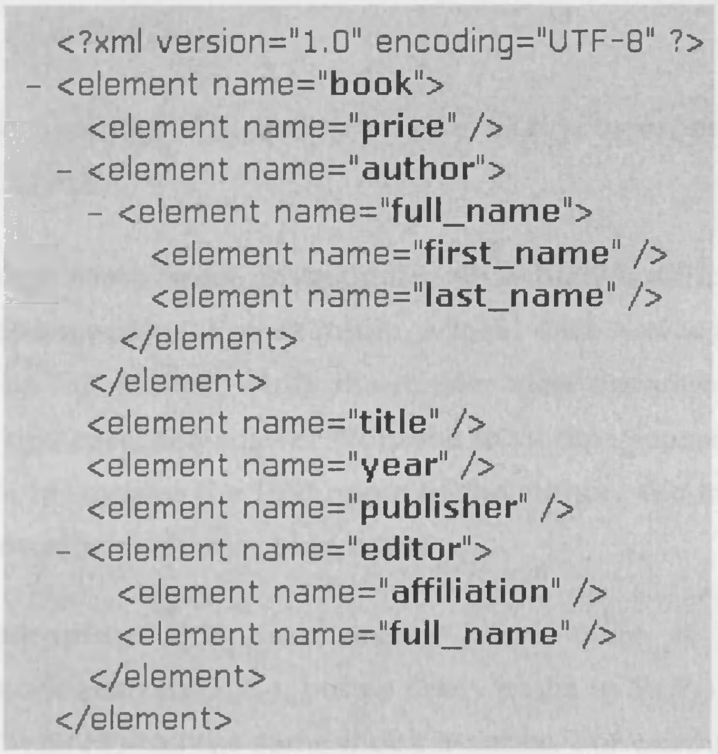


Figure 5.12: The master view tree structure with index numbers.

Figure 5.12 shows the tree structure with the index numbers of the master view shown in Figure 5.13.



```
  <?xml version="1.0" encoding="UTF-8" ?>
- <element name="book">
    <element name="price" />
  - <element name="author">
    - <element name="full_name">
        <element name="first_name" />
        <element name="last_name" />
      </element>
    </element>
    <element name="title" />
    <element name="year" />
    <element name="publisher" />
  - <element name="editor">
      <element name="affiliation" />
      <element name="full_name" />
    </element>
  </element>
```

Figure 5.13: The Master View.

5.4.4 Mapping cases between elements

We can classify the cardinality of mapping cases as the number of paths that correspond to each other, i.e. the number of participating paths in each mapping. The mapping between the correspondence paths can be expressed in the following form:

$$G(P_i) \rightarrow S_j(P_k);$$

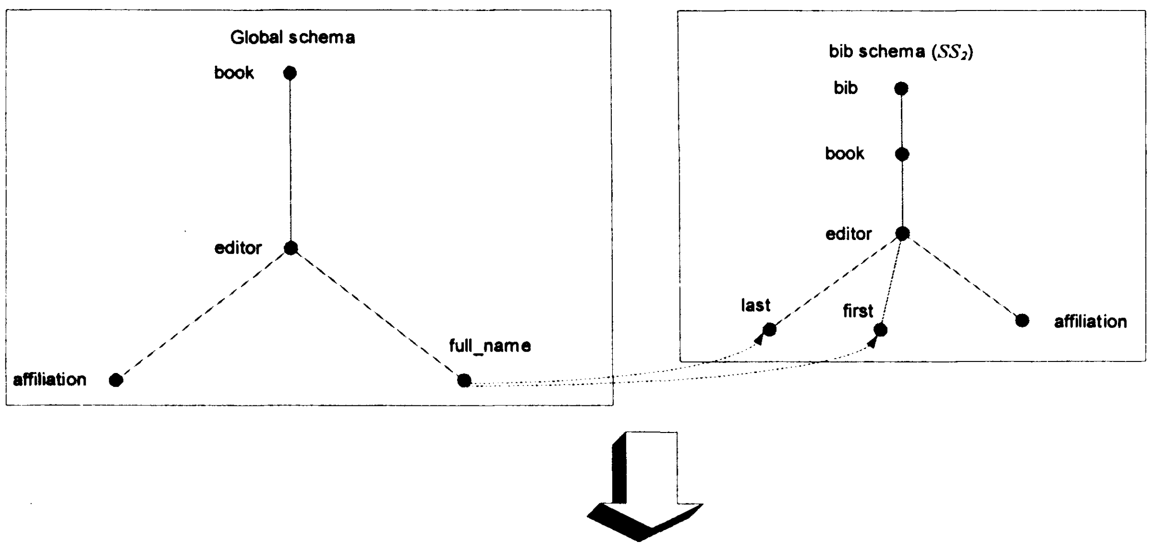
Where $G(P_i)$ is a set of global paths, S_j is a set of local sources, P_k is a set of local paths, such that: $i, j, k \in \{1, \dots n\}$.

The mapping cardinality can then be expressed as follows:

- One to one mapping: if there $\exists!\ P_i \in G(P_i)$ corresponding to $\exists!\ P_k \in S_j(P_k)$.
- One to N mapping: if there $\exists!\ P_i \in G(P_i)$ corresponding to $\exists\ (P_k > 1) \in S_j(P_k)$.
- N to one mapping: if there $\exists\ (P_i > 1) \in G(P_i)$ corresponding to $\exists!\ P_k \in S_j(P_k)$.

Several mapping cases were investigated in which conflicts may occur between the schema paths. For example, a local data source may represent author names as full names, while the master view separates the first and last names. In this case, the answer from the local data source must be split up if a query is to retrieve the first name of the author. We introduce some examples to describe these mapping cases.

One to N mapping: this case occurs when there is a component represented as one path in $G(P_i)$, but as many paths in $S_j(P_k)$. Hence, more than one path in $S_j(P_k)$ has the same index number. For example,



```
<source path="/book/editor/full_name">
   <target name="books.xml" fun="Null">Null</target>
   <target name="bib.xml" fun="Merge">/bib/book/editor/last,/bib/book/editor/first</target>
   <target name="SCMFMA" fun="Null">Null</target>
   <target name="bookdata.xml" fun="Null">Null</target>
</source>
```

Figure 5.14: One to N mapping example.

the master view (global schema) may represent an editor's name as a full name, while the local data source separates an editor's first and last names. To resolve this conflict a *UDF* is needed to combine the editor's first and the last name elements to get the full name. The editor full_name node in our example global schema tree is an example of this case. Figure 5.14 shows such a mapping. Here, in the bib schema tree (*SS_2*) the editor full name information is represented by two separated paths *SS_2(/bib/book/editor/last)* for the last name of the editor and *SS_2(/bib/book/editor/first)* for the first name of the editor. At the same time, this information is represented by one element in the global schema. Hence, a UDF is provided, e.g. *Merge()*, which concatenates the first and last name elements to get the full name. The number of arguments of this function is equal to the number of paths that appear in the bib schema mapping path which correspond to *G(book/editor/full_name)*. We note there are two paths here:

SS_2*(/bib/book/editor/last)* and SS_2*(/bib/book/editor/first)*,

these in turn are concatenated to answer a query for the *G*(*P*(*book/editor/full_name*)) information.

N to one mapping: this case occurs if two or more paths in $G(P_i)$ correspond to one path in $S_j(P_k)$. Hence, a path in $S_j(P_k)$ will have more than one index number and more than one function name. For example,

G(*/book/author/full_name/first_name*) and
G(*book/author/full_name/last_name*)

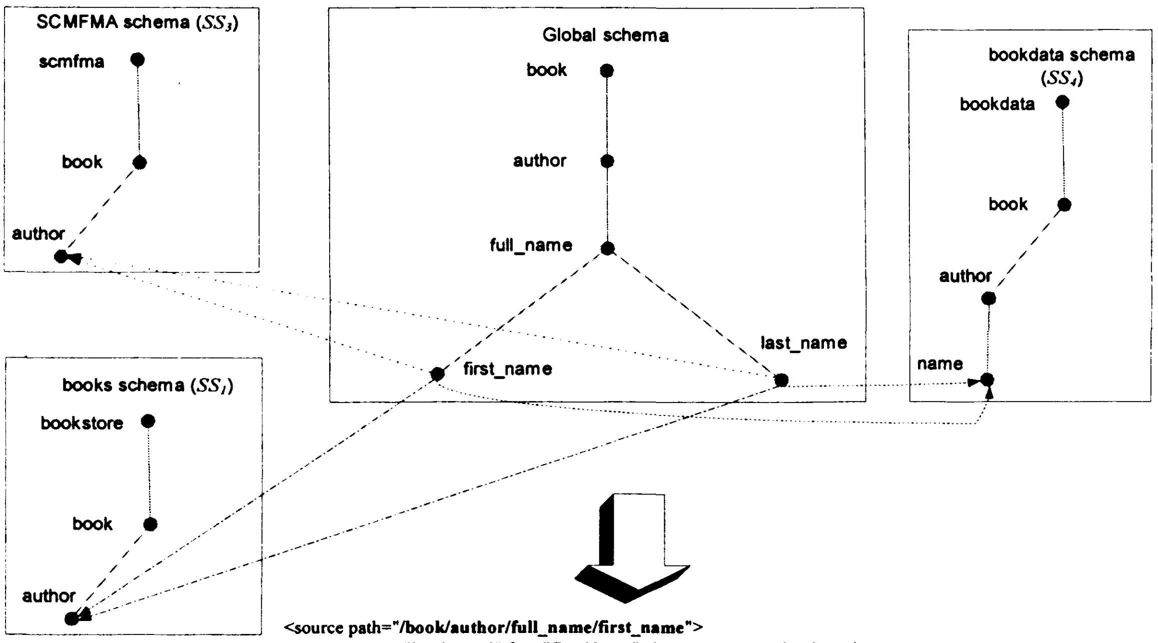
in the global schema correspond to

SS_1*(/bookstore/book/author)* in the books schema (SS_1).

SS_3*(/scmfma/book/author)* in the SCMFMA schema (SS_3).

SS_4*(/bookdata/book/author/name)* in the bookdata schema (SS_4).

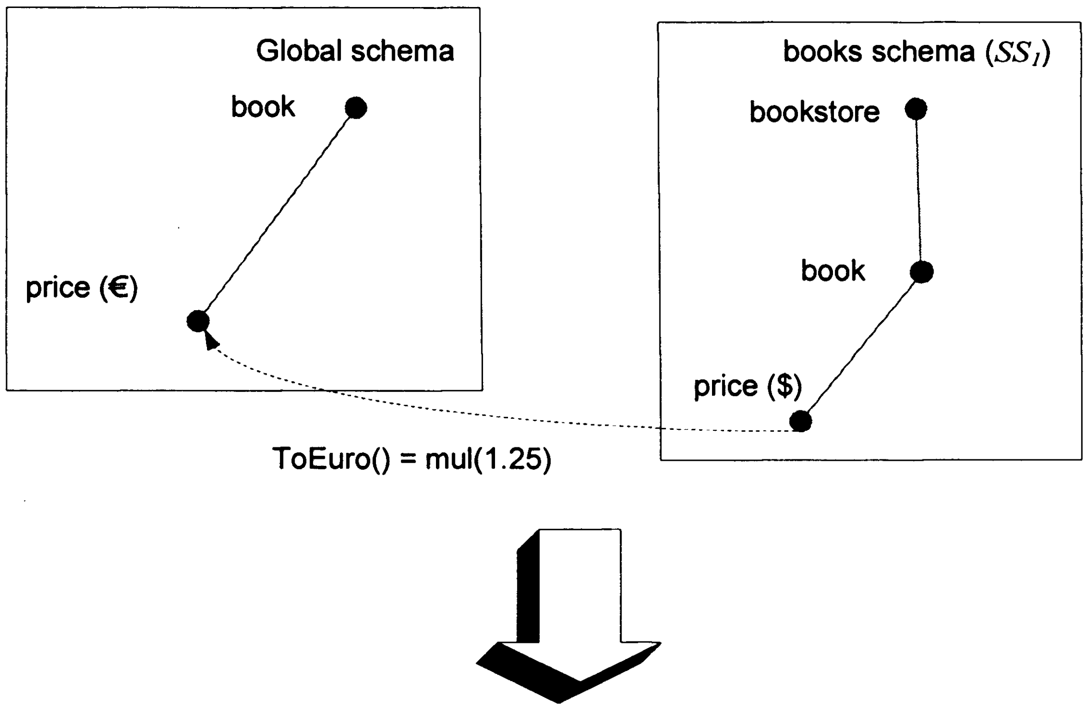
In this case, *UDF* functions are needed to resolve the conflict, e.g., *firstName()* and *lastName()*. The task of these functions are to split the *author* element value in SS_1, the *author* element value in SS_3 and the *name* element value in SS_4, which contain the author full name into separate *first_name* and *last_name*. Figure 5.15 shows that in the SS_1, SS_3 and SS_4 source mapping, the paths */bookstore/book/author*, */scmfma/book/author* and */bookdata/book/author/name* exist twice. Each one corresponds to more than one different global path in *G*(*Pi*). This case is a *2-to-One* mapping case, in that the *firstName()* and *lastName()* functions should be associated with the corresponding global path specified by the designer as an argument for these functions, e.g. the values of the *first_name* and *last_name* elements must be separated from the *author* element value in SS_1, the *author* element value in SS_3 and the *name* element value in SS_4. The returned value of the *firstName()* and *lastName()* functions depends on which global path expression invokes it. The implementation of this function is explicitly coded by the designer.



```
<source path="/book/author/full_name/first_name">
  <target name="books.xml" fun="firstName">/bookstore/book/author</target>
  <target name="bib.xml" fun="Null">/bib/book/author/first</target>
  <target name="SCMFMA" fun="firstName">/scmfma/book/author</target>
  <target name="bookdata.xml" fun="firstName">/bookdata/book/author/name</target>
</source>
<source path="/book/author/full_name/last_name">
  <target name="books.xml" fun="LastName">/bookstore/book/author</target>
  <target name="bib.xml" fun="Null">/bib/book/author/last</target>
  <target name="SCMFMA" fun="LastName">/scmfma/book/author</target>
  <target name="bookdata.xml" fun="LastName">/bookdata/book/author/name</target>
</source>
```

Figure 5.15: N to one mapping example.

One to one mapping with an operation: this case occurs if one path in $G(P_i)$ corresponds to one path in $S_j(P_k)$ but they use different reference systems. This is a granularity conflict, and a specific operation is required to resolve a semantic difference among the two related elements. For example, the price element in SS_1 uses dollar currency, while in the global schema the price element is expressed in euro. To resolve this conflict some conversion mechanism is required which translates between the representations. In this example a *UDF* function is needed to perform an exchange operation in order to get the price in euro, when a query is posed.



```
<source path="/book/price">
  <target name="books.xml" fun="ToEuro">/bookstore/book/price</target>
  ...............................
  ...............................
  ...............................
</source>
```

Figure 5.16: Example of one to one mapping with an operation.

Hence, the mapping between the *G(/book/price)* in the global schema and the SS_1*(/bookstore/book/price)* in the SS_1 schema should be provided with the name of the *UDF* for the exchange currency operation, e.g. *ToEuro()*. The construction of this function is undertaken by the designer. This function should read the price element value in SS_1 and return its equivalent amount in euro. Figure 5.16 shows this mapping.

5.5 Summary

In this chapter we have proposed the mediation of distributed heterogeneous structured and semi-structured data sources as a tool to overcome logical heterogeneity problems, which may occur when integrating data sources. Also we have introduced the mediation process, which has the following steps: (1) generate the Schema Structure Definition (SSD); (2) extract SSD components and generate paths; (3) establish the mappings and generate the mediation information (XMKB).

CHAPTER 6

The query translation process

In this chapter, we deal with the second important aspect of the thesis, which is the Query Processor (QP). The Query Processor (QP) is an integral part of the mediation layer of the SISSD system. A brief introduction concerning the query translation task in data integration systems is followed by a description of the internal architecture of the Query Processor and its components. Then the query translation process is introduced, followed by a brief description of the translation process of XQuery FLWR expressions into SQL queries. Some examples of query translations are given.

6.1 Introduction

The main purpose of building data integration systems is to facilitate access to several data sources. The ability to correctly and efficiently process the queries on the integrated data lies at the heart of the integration system. The integration system must contain a module that uses source descriptions when reformulating user queries which are posed in terms of the composite global schema, into sub-queries that refer directly to the schemas of the component data sources [104, 106]. The user does not pose queries directly to the schema in which the data is stored, since one of the principal goals of a data integration system is to free the user from having to know about the specific data sources and interact with each one separately. Instead, the user poses queries on the *mediated schema*. The mediated schema is a set of *virtual* relations, in the sense that they are not actually stored anywhere [80]. In general, this query processing involves:

1. Reading the user query and parsing it.
2. Using a query optimizer to produce an efficient query execution plan.
3. Executing this plan on the physical data.

We are only concerned with query translation not query optimization. We propose a method for query translation which targets distributed heterogeneous structured data residing in relational databases and semi-structured data held in well-formed XML documents, produced by Internet applications or by human-coding. These XML documents can be XML files on local hard drives or remote documents on Web servers. It is important to develop a technique to seamlessly translate user queries over the master view into sub-queries - called local queries - fitting the appropriate participating data sources. This is achieved by exploiting the mapping information between the master (composite) view and the participating data source Schema Structure Definitions (SSDs) that are defined in the generated XML Metadata Knowledge Base (XMKB) [14].

We have chosen XML to provide a unifying data model in the SISSD data integration system, as this data model is general enough to accommodate hierarchical and relational data sources [91]. We expect, that a data integration query will typically be posed in XQuery, the standard XML query language being developed by the W3C [5]. It is derived from Quilt, an earlier XML query language designed by Jonathan Robie, IBM's Don Chamberlin - co-inventor of SQL - and Daniela Florescu, a well-known database researcher [40]. XQuery is designed to be a language in which queries are concise and easily understood. It is also flexible enough to query a broad spectrum of XML information sources, including both databases and documents. It can be used to query XML data that has no schema at all, or conforms to a W3C standard XML Schema or a Document Type Definition (DTD).

XQuery is centered on the notion of expression; starting with constants and variables, expressions can be nested and combined using arithmetic, logical and list operators, navigation primitives, function calls, higher order operators like sort, conditional expressions, element constructors, etc. For navigating in a document, XQuery uses path expressions, whose syntax is borrowed from the abbreviated syntax of XPath. The evaluation of a path expression on an XML document returns a list of information items, whose order is dictated by the order of elements within the document (also called document order).

Our Query Processor (QP) supports FLWR (short for For-Let-Where-Return) expressions. This subset of XQuery is used because it is powerful and meets the requirements of our approach. The *for-let* clause makes variables iterate over the result of an expression or binds variables to arbitrary expressions, the *where* clause allows specification of restrictions on the variables, and the *return* clause can construct new XML elements as output of the query. In general, an XQuery query consists of an optional list of namespace definitions, followed by a list of function definitions, followed by a single query expression.

6.2 The Query Processor architecture and Components

In this section, we present an overview of the Query Processor (QP) architecture and summarize the functions of the main components. The architecture is shown in Figure 6.1. It consists of five components: XQuery Parser, XQuery Rewriter, Query Execution, XQuery-SQL Translator, and Tagger. The core of the QP and the primary focus of this chapter is the XQuery Rewriter. This component rewrites the user query posed over the master view into sub-queries which fit each local data source, by using the mapping information stored in the XMKB. The main role played by each of the components in Figure 6.1 follows.

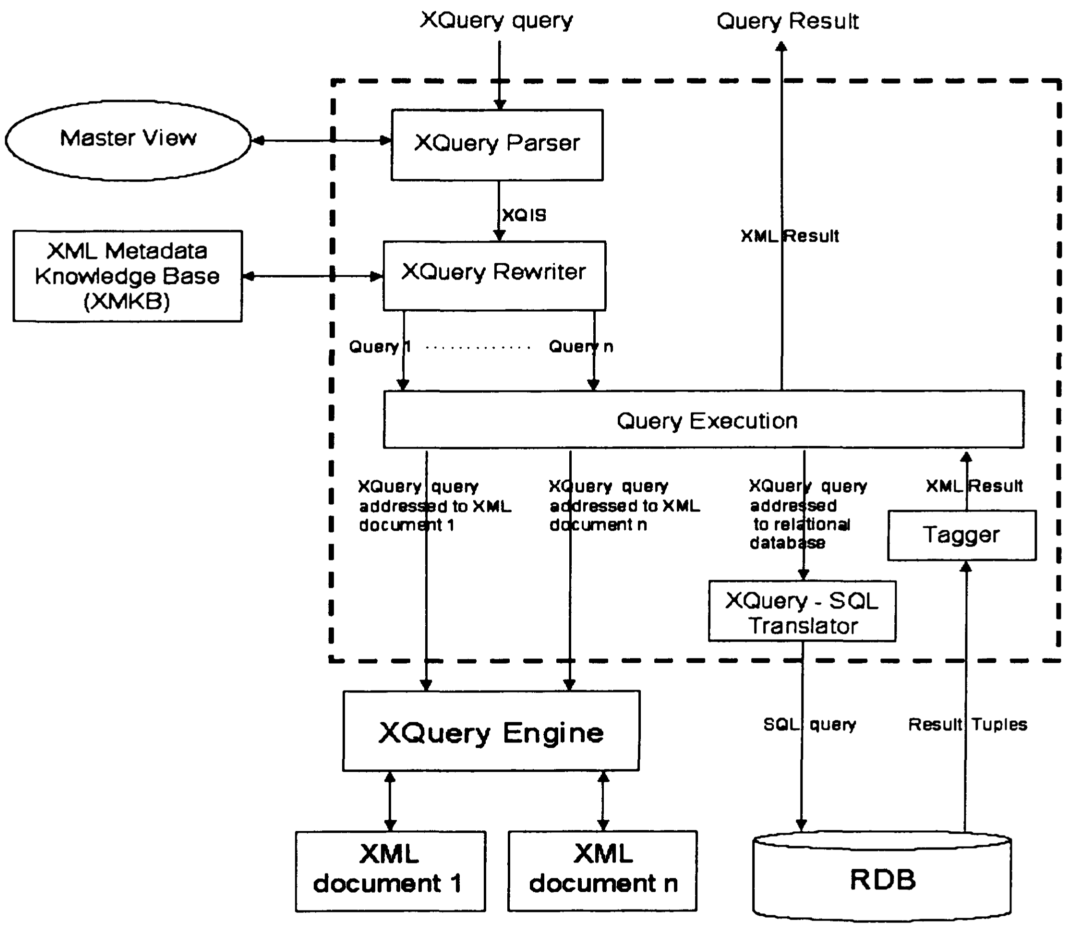


Figure 6.1: The QP Architecture.

- **XQuery Parser:** parses a given XQuery FLWR expression in order to check it for syntactic correctness and ensure that the query is valid and conforms to the relevant master view. Also the parser analyses the query to generate an XQuery Internal Structure (XQIS) which contains the XML paths, variables, conditions and tags present in the query, before passing it to the XQuery Rewriter.

- **XQuery Rewriter:** Takes the XQIS representation of a query, consults the XMKB to obtain the local paths corresponding to the master paths and function names for handling semantic and structural discrepancies, then produces semantically equivalent XQuery queries to fit each local data source. That is, wherever there is a correspondence between the paths in the master view and local Schema Structure Definitions (SSDs) concerned (otherwise the local data source is ignored).

- **Query Execution:** Receives the rewritten XQuery queries, consults the XMKB to determine each data source's location and type (relational database or XML document), then sends each local query to its corresponding query engine, to execute the query and return the results.

- **XQuery-SQL Translator:** Translates the XQIS representation of an XQuery query addressed to a relational database into the SQL query needed to locate the result, then hands the query over to the relational database engine to execute it and return the result in tabular format through the Tagger.

- **Tagger:** Adds the appropriate XML tags to the tabular SQL query result to produce structured XML documents for return to the user.

6.3 The Query Translation process

The Query Processor (QP) component is an important part of the mediation layer of the SISSD system. Its task is the translation of master queries that are posed on the master view into a set of local queries fitting each local data source. The QP gives flexibility to the user to choose the master view that he/she wants to pose his/her query over and then automatically selects the appropriate XMKB that will be used to process any query posed over this master view. The master view provides the user with the elements on which the query can be based. Hence, a user XQuery query written in terms of the master view is rewritten into sub-queries which can be executed locally. We introduce a method for the query translation to produce queries for the distributed heterogeneous structured data residing in relational databases and semi-structured data held in well-formed XML documents. This method is based on the mapping information between the master view and the participating data source Schema Structure Definition (SSD), which are defined in the generated XMKB. Once the XMKB is generated, user queries can be issued on the master view and easily evaluated. Hence, when a query is posed against the master view, the query translation process is accomplished as follows:

First, the given global query is parsed by the XQuery parser module to generate the XQuery Internal Structure (XQIS) which contains the global paths, variables, conditions and tag present in the XQuery query, which is passed to the XQuery Rewriter. XMKB is read and parsed by JDOM to identify the number of local data sources that participate in the integration system, their location and type.

Second, for each element path in the master query, there should be an attribute *path* of element *<source>* in XMKB. If there is a non-empty value for the corresponding local elements (*<target>* elements in XMKB), then the corresponding local paths and the function names (an attribute *fun* of <*target*> element) is obtained from the XMKB. Then the global paths in the master query are replaced by their corresponding local paths (<*target*> elements) obtained from XMKB and the function names are added if they are not null to generate a local query. It may happen that no local query is generated when the content of a local path for a specific local data source is null. This means the query cannot be applied to this local data source. Also, if the content of the function name (an attribute *fun* of <*target*> element) is null, this indicates the translation is straightforward and no function is needed.

Third, the generated local XQuery for a relational database is converted into SQL before passing it to the relational database engine for execution.

Finally, each (generated) local query is sent to the corresponding local data source engine for local execution.

Using the descriptions of the SISSD Query Processor (QP) component architecture (section 6.2) and the XML Metadata Knowledge Base (XMKB) organization and contents (section 5.4.1), we are now in a position to summarize the query translation (rewriting) process carried out at the heart of our system by the QP module. We do so in algorithmic form (see Figure 6.2). The algorithm is both conceptually simple and generally applicable. We have successfully implemented and tested it on a variety of relational and XML data source integration examples in our prototype SISSD system.

6.4 XQuery-to-SQL translation process

The Query Processor (QP) uses XQuery FLWR expressions as the query language of the SISSD data integration system. Using FLWR expressions for querying a master view makes it easy to translate the sub-queries directed at relational databases into SQL queries since syntactically, FLWR expressions are similar to SQL select statements and have similar capabilities, only they use path expressions instead of table and column names.

```
Algorithm: Master query translation process
Input: Master View, Master XQuery query q, and XMKB
Output: local sub-queries q1, q2..., qn
Step1: parse q;
Step2: get global paths g1, g2...., gn from Master View;
Step3: read XMKB;
Step4: identify the number of local data sources participating
in the integration system, their locations and types;
Step5: for each data source Si do
          for each global path ge in q do
             if the corresponding local path le not null then
                get le;
                if the function name fe not null then
                   get fe;
                end if
             else
                  no query generated for this local data source Si ;
             end if
          end for
          replace g1 by l1 with f1, g2 by l2 with f2 ..., gn by ln with
                fn, in qi;
          if data source type is relational database then
             convert qi XQuery into SQL;
       end for
 Step6: execute the generated local query qi by sending it to the
 corresponding local data source engine, and return the result, with
 XML tags added to SQL tables.
```

Figure 6.2: Algorithm for the query translation process.

There is a conceptual difference between an XQuery FLWR expression's concept of iterating in the evaluation of an expression e2 for successive bindings of a variable $v (for $v in e1 return e2) and the set- or table-oriented processing model of SQL. This is resolved by mapping for-bound variables like $v into tables containing all bindings and translating expressions independently of the variable scopes in which they appear. The resulting SQL code implements iteration via equi-joins, a table operation, which RDBMS engines execute efficiently [78].

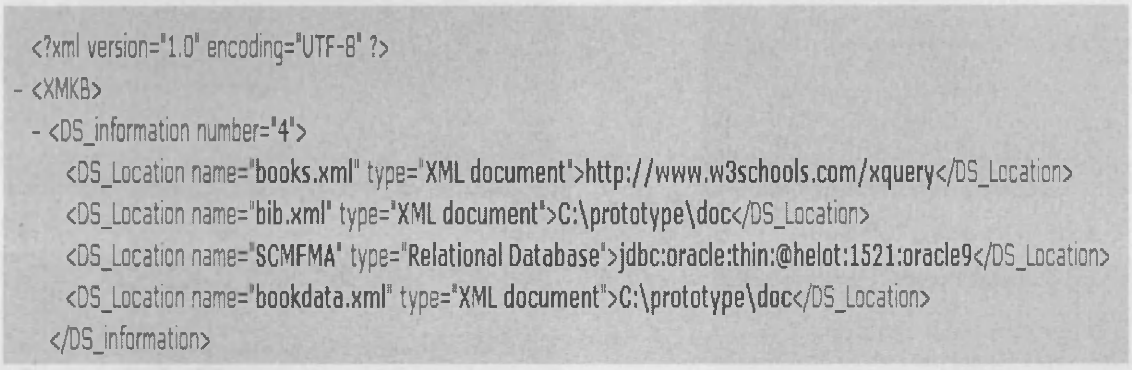
The translation process of XQuery FLWR expression into an SQL query starts by parsing the XQuery FLWR expression to identify its path expressions. The path expression of the *FOR* clause is the root path expression and the others are dependent path expressions. This translation is achieved by:

- First: the relation(s) corresponding to the path expression(s) of the *FOR* clause are identified and put in the *FROM* clause of the SQL query.
- Second: if the XQuery FLWR expression contains a *WHERE* clause then the condition is extracted and the path expression(s) in this condition are identified and replaced by the corresponding attribute(s), which are added to the *WHERE* clause of the SQL query.
- Third: the attribute(s) corresponding to the path expression(s) in the *RETURN* clause in the XQuery FLWR expression are identified and added to the *SELECT* clause of the SQL query.

6.5 Query translation examples

In this section, we introduce some examples of global query translation. These examples will be used in testing the system. We discussed in section 6.3 the technique of the Query Processor to seamlessly translate user queries (XQuery queries) over the master view into sub-queries suited to an appropriate data source, by exploiting the mapping information stored in the XMKB. To illustrate this process, four cases are investigated in the following subsections: one-to-one Mapping, function-involved in a one-to-one Mapping, one-to-many Mapping, and many-to-one Mapping.

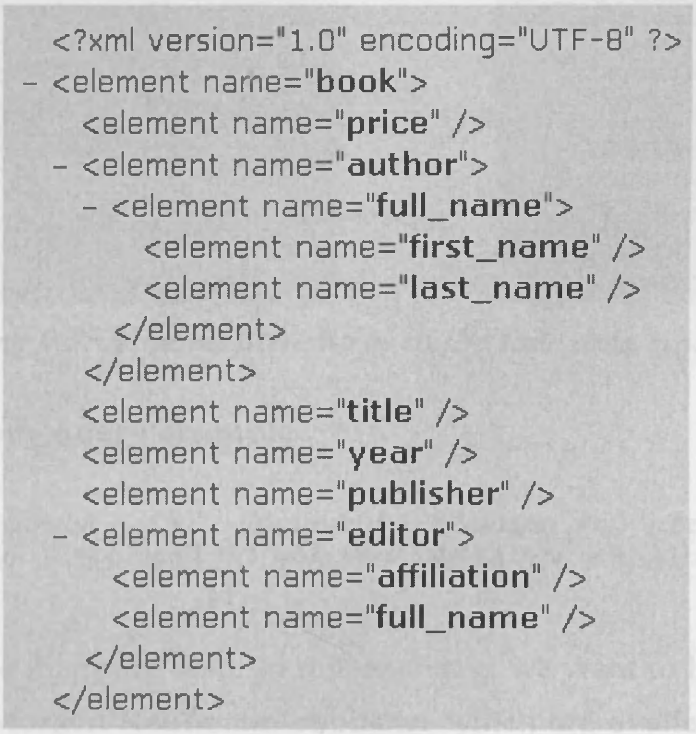
Figure 6.3 shows part of an XMKB which describes the data sources participating in the integration system and their information (names, types and locations). It shows that there are four data sources participating in the integration system, one of them is a relational database and the other three are XML documents (one of these XML documents is a remote document on a web server and the other two are on the local hard drive).



```
  <?xml version="1.0" encoding="UTF-8" ?>
- <XMKB>
  - <DS_information number="4">
      <DS_Location name="books.xml" type="XML document">http://www.w3schools.com/xquery</DS_Location>
      <DS_Location name="bib.xml" type="XML document">C:\prototype\doc</DS_Location>
      <DS_Location name="SCMFMA" type="Relational Database">jdbc:oracle:thin:@helot:1521:oracle9</DS_Location>
      <DS_Location name="bookdata.xml" type="XML document">C:\prototype\doc</DS_Location>
    </DS_information>
```

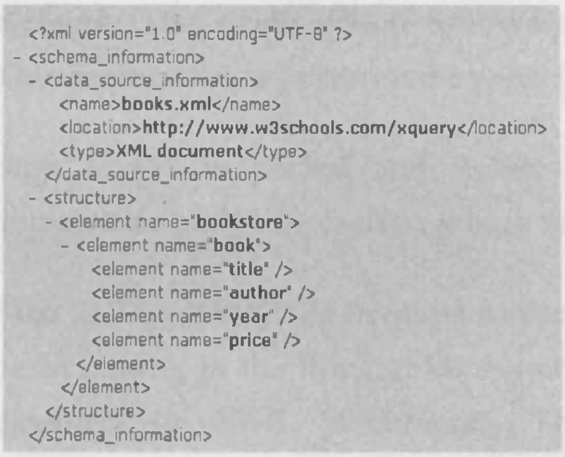
Figure 6.3: The part of XMKB which maintain data sources information.

The Master view and the Schema Structure Definitions (SSDs) of the four data sources (bookstore relational database, bib.xml, bookdata.xml and books.xml) are shown in Figures 6.4 and 6.5.

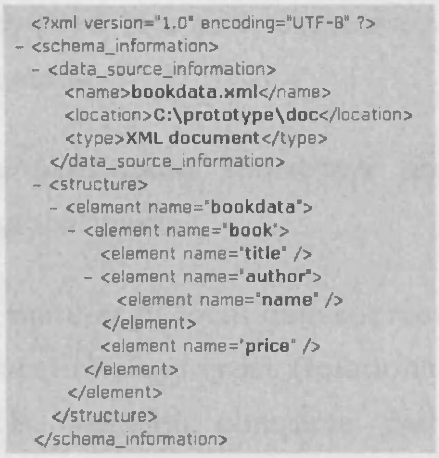


```
  <?xml version="1.0" encoding="UTF-8" ?>
- <element name="book">
    <element name="price" />
  - <element name="author">
    - <element name="full_name">
        <element name="first_name" />
        <element name="last_name" />
      </element>
    </element>
    <element name="title" />
    <element name="year" />
    <element name="publisher" />
  - <element name="editor">
      <element name="affiliation" />
      <element name="full_name" />
    </element>
  </element>
```

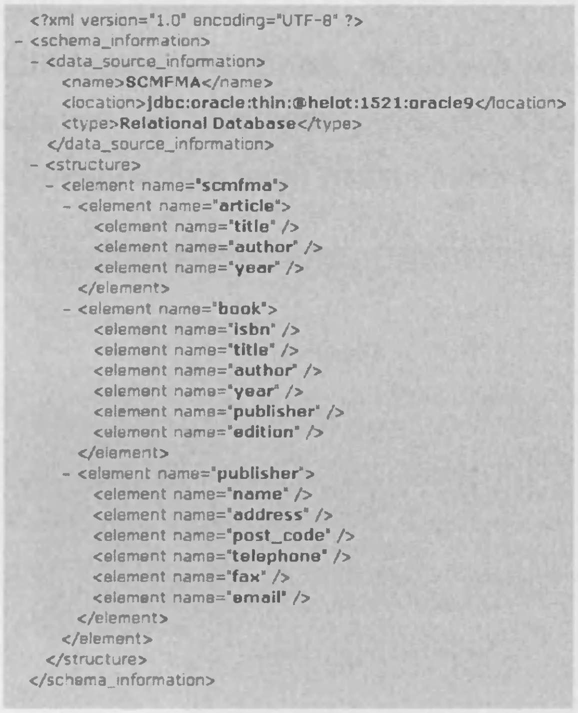
Figure 6.4: The Master View.



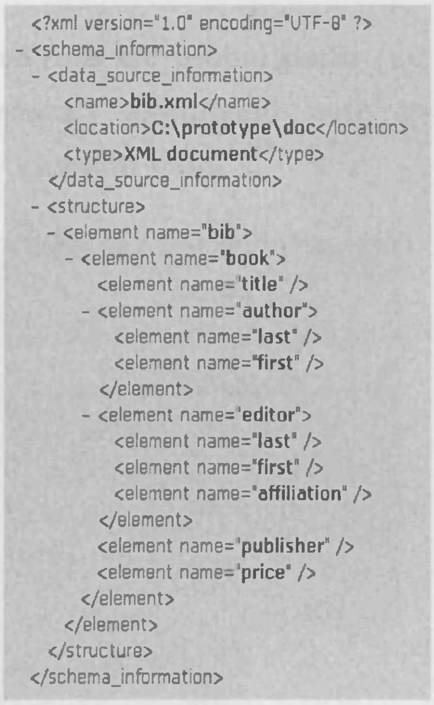
```
<?xml version="1.0" encoding="UTF-8" ?>
- <schema_information>
  - <data_source_information>
      <name>books.xml</name>
      <location>http://www.w3schools.com/xquery</location>
      <type>XML document</type>
    </data_source_information>
  - <structure>
    - <element name="bookstore">
      - <element name="book">
          <element name="title" />
          <element name="author" />
          <element name="year" />
          <element name="price" />
        </element>
      </element>
    </structure>
  </schema_information>
```



```
<?xml version="1.0" encoding="UTF-8" ?>
- <schema_information>
  - <data_source_information>
      <name>bookdata.xml</name>
      <location>C:\prototype\doc</location>
      <type>XML document</type>
    </data_source_information>
  - <structure>
    - <element name="bookdata">
      - <element name="book">
          <element name="title" />
        - <element name="author">
            <element name="name" />
          </element>
          <element name="price" />
        </element>
      </element>
    </structure>
  </schema_information>
```



```
<?xml version="1.0" encoding="UTF-8" ?>
- <schema_information>
  - <data_source_information>
      <name>SCMFMA</name>
      <location>jdbc:oracle:thin:@helot:1521:oracle9</location>
      <type>Relational Database</type>
    </data_source_information>
  - <structure>
    - <element name="scmfma">
      - <element name="article">
          <element name="title" />
          <element name="author" />
          <element name="year" />
        </element>
      - <element name="book">
          <element name="isbn" />
          <element name="title" />
          <element name="author" />
          <element name="year" />
          <element name="publisher" />
          <element name="edition" />
        </element>
      - <element name="publisher">
          <element name="name" />
          <element name="address" />
          <element name="post_code" />
          <element name="telephone" />
          <element name="fax" />
          <element name="email" />
        </element>
      </element>
    </structure>
  </schema_information>
```



```
<?xml version="1.0" encoding="UTF-8" ?>
- <schema_information>
  - <data_source_information>
      <name>bib.xml</name>
      <location>C:\prototype\doc</location>
      <type>XML document</type>
    </data_source_information>
  - <structure>
    - <element name="bib">
      - <element name="book">
          <element name="title" />
        - <element name="author">
            <element name="last" />
            <element name="first" />
          </element>
        - <element name="editor">
            <element name="last" />
            <element name="first" />
            <element name="affiliation" />
          </element>
          <element name="publisher" />
          <element name="price" />
        </element>
      </element>
    </structure>
  </schema_information>
```

Figure 6.5: Schema Structures of the four data sources.

6.5.1 One-to-one query example

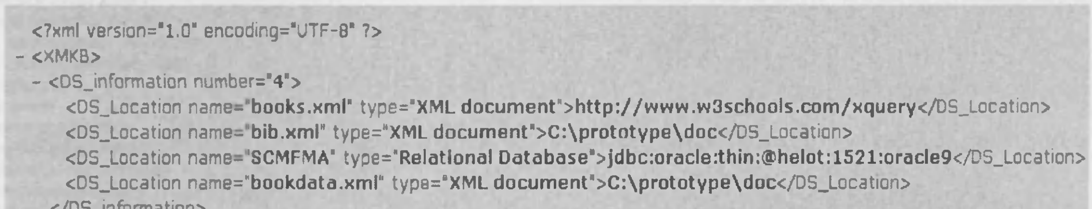
Q1: *FOR $book IN document ("master.xml")/book WHERE $book/publisher ="Morgen Kaufmann" RETURN <book> {$book/title} </book>*

This is a simple mapping case. In this example, we want to list all the titles published by Morgen Kaufmann publisher which are available in the four data sources. The FOR-clause binds the variable $book to the Book XML element. The string which follows the IN keyword is a path expression. This translation is performed by the following steps:

Step 1: Q1 is parsed and */book, $book/publisher* and *$book/title* are detected as path expressions which represent global paths.

Step 2: The XMKB is invoked to identify the number of local data sources participating in the integration system, their locations and types (relational database or XML document). The XMKB contains complete path mappings. Figure 6.6 shows parts of XMKB in which these paths appear.

Obviously, the */book, /book/publisher* and *book/title* are global paths (i.e. content of attribute *path* of *<source>* element) associated with its corresponding local paths values (i.e. *<target>* element value).

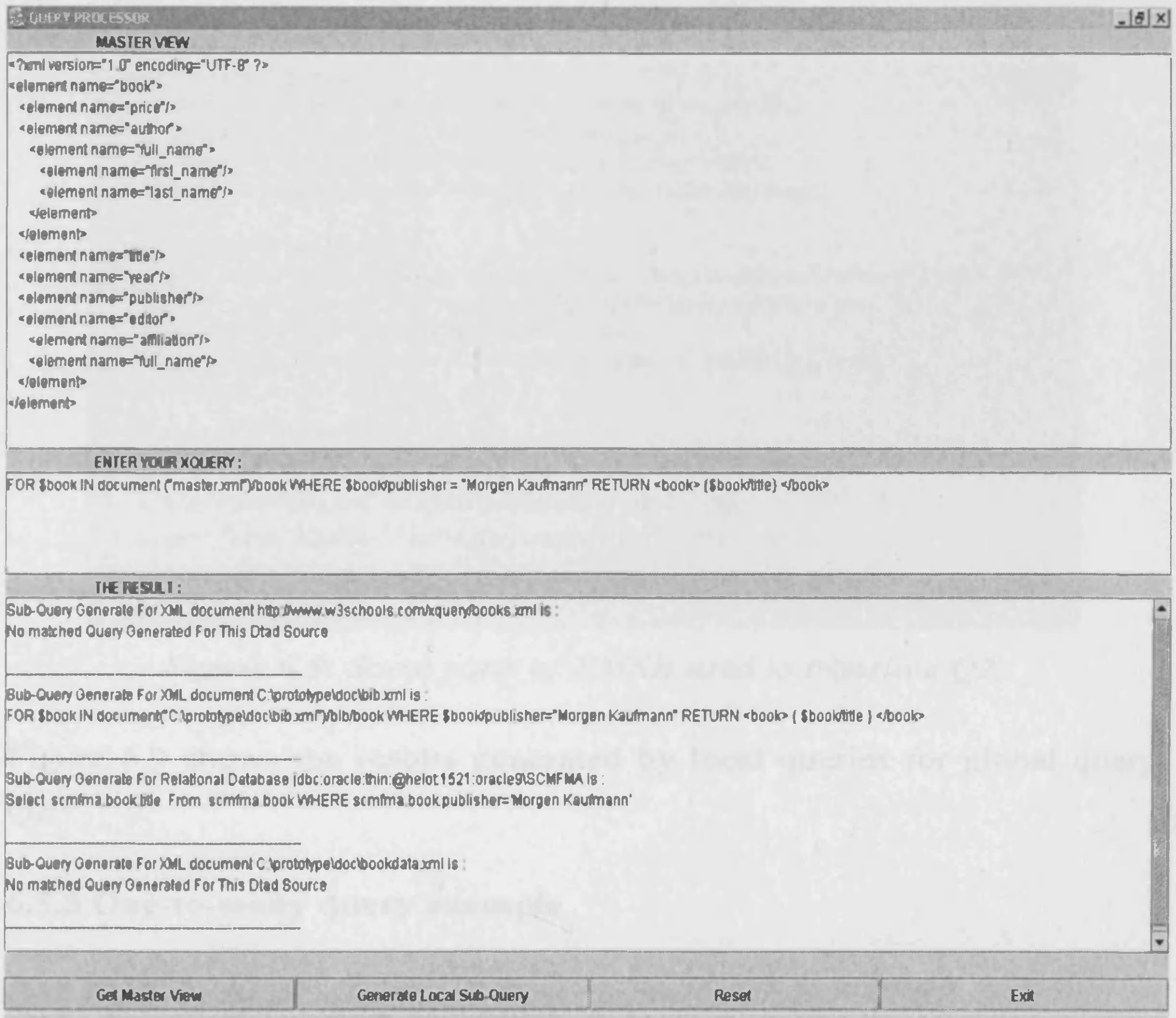


```
<?xml version="1.0" encoding="UTF-8" ?>
- <XMKB>
  - <DS_information number="4">
      <DS_Location name="books.xml" type="XML document">http://www.w3schools.com/xquery</DS_Location>
      <DS_Location name="bib.xml" type="XML document">C:\prototype\doc</DS_Location>
      <DS_Location name="SCMFMA" type="Relational Database">jdbc:oracle:thin:@helot:1521:oracle9</DS_Location>
      <DS_Location name="bookdata.xml" type="XML document">C:\prototype\doc</DS_Location>
    </DS_information>
  - <Med_component>
    - <source path="/book">
        <target name="books.xml" fun="Null">/bookstore/book</target>
        <target name="bib.xml" fun="Null">/bib/book</target>
        <target name="SCMFMA" fun="Null">/scmfma/book</target>
        <target name="bookdata.xml" fun="Null">/bookdata/book</target>
      </source>
    - <source path="/book/title">
        <target name="books.xml" fun="Null">/bookstore/book/title</target>
        <target name="bib.xml" fun="Null">/bib/book/title</target>
        <target name="SCMFMA" fun="Null">/scmfma/book/title</target>
        <target name="bookdata.xml" fun="Null">/bookdata/book/title</target>
      </source>
    - <source path="/book/year">
        <target name="books.xml" fun="Null">/bookstore/book/year</target>
        <target name="bib.xml" fun="Null">Null</target>
        <target name="SCMFMA" fun="Null">/scmfma/book/year</target>
        <target name="bookdata.xml" fun="Null">Null</target>
      </source>
    - <source path="/book/publisher">
        <target name="books.xml" fun="Null">Null</target>
        <target name="bib.xml" fun="Null">/bib/book/publisher</target>
        <target name="SCMFMA" fun="Null">/scmfma/book/publisher</target>
        <target name="bookdata.xml" fun="Null">Null</target>
      </source>
```

Figure 6.6: Some parts of XMKB used to translate Q1.

Step 3: a local query is generated for each <*target*> element whose value is not null. Hence, by navigating the XMKB, for each local query, the global path is replaced by its corresponding local path obtained from the XMKB. If there is no corresponding local path, then no query is generated for this data source, which means this global query cannot be applied to this local data source. On the other hand, if the function representation is null that means there is no function needed for this case.

Step 4: the generated local XQuery query for the relational database is then converted into SQL. Figure 6.7 shows the generated local queries from the global query Q1.



```
<?xml version="1.0" encoding="UTF-8" ?>
<element name="book">
  <element name="price"/>
  <element name="author">
    <element name="full_name">
      <element name="first_name"/>
      <element name="last_name"/>
    </element>
  </element>
  <element name="title"/>
  <element name="year"/>
  <element name="publisher"/>
  <element name="editor">
    <element name="affiliation"/>
    <element name="full_name"/>
  </element>
</element>
```

```
FOR $book IN document ("master.xml")/book WHERE $book/publisher = "Morgen Kaufmann" RETURN <book> {$book/title} </book>
```

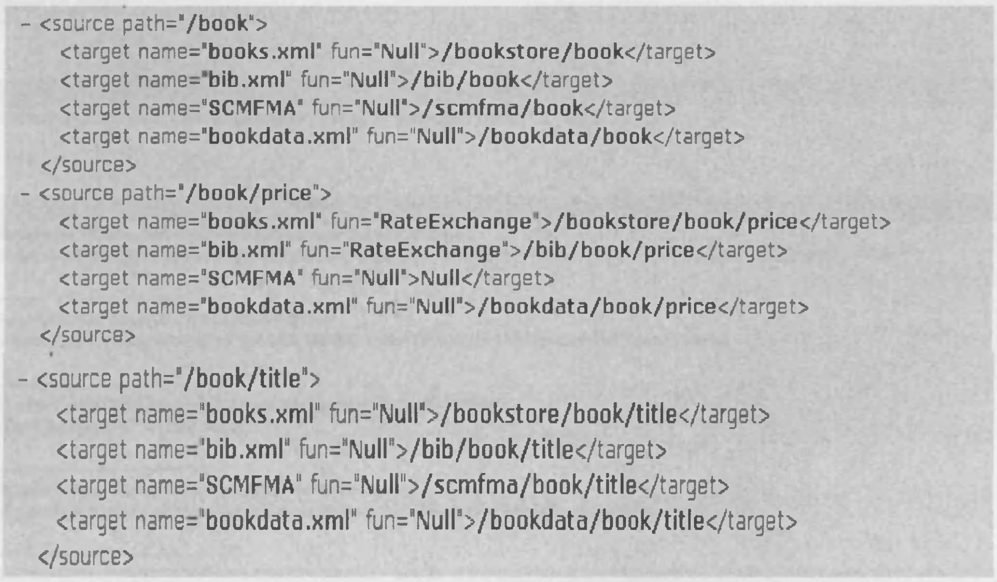
```
Sub-Query Generate For XML document http://www.w3schools.com/xquery/books.xml is :
No matched Query Generated For This Dtad Source
________________________________

Sub-Query Generate For XML document C:\prototype\doc\bib.xml is :
FOR $book IN document("C:\prototype\doc\bib.xml")/bib/book WHERE $book/publisher="Morgen Kaufmann" RETURN <book> { $book/title } </book>
________________________________

Sub-Query Generate For Relational Database jdbc:oracle:thin:@helot:1521:oracle9\SCMFMA is :
Select  scmfma.book.title  From  scmfma.book WHERE scmfma.book.publisher='Morgen Kaufmann'
________________________________

Sub-Query Generate For XML document C:\prototype\doc\bookdata.xml is :
No matched Query Generated For This Dtad Source
________________________________
```

Figure 6.7: The generated local queries from Q1.

6.5.2 Function-involved one-to-one query example

Q2: *FOR $book IN document ("master.xml")/book RETURN <book> {$book/title, $book/price} </book>*

This is a function-involved one-to-one mapping case. The query will list all the titles and prices available at the four data sources. The query is parsed and */book, $book/title* and *$book/price* are detected as path expressions which represent global paths. The XMKB is invoked to obtain the corresponding local path and the function name (if it not null) for each global path. In each local query, the global path is replaced by its corresponding local path with the function name if it is not null. Figure 6.8 shows parts of XMKB in which these paths appear.



```
- <source path="/book">
    <target name="books.xml" fun="Null">/bookstore/book</target>
    <target name="bib.xml" fun="Null">/bib/book</target>
    <target name="SCMFMA" fun="Null">/scmfma/book</target>
    <target name="bookdata.xml" fun="Null">/bookdata/book</target>
  </source>
- <source path="/book/price">
    <target name="books.xml" fun="RateExchange">/bookstore/book/price</target>
    <target name="bib.xml" fun="RateExchange">/bib/book/price</target>
    <target name="SCMFMA" fun="Null">Null</target>
    <target name="bookdata.xml" fun="Null">/bookdata/book/price</target>
  </source>
- <source path="/book/title">
    <target name="books.xml" fun="Null">/bookstore/book/title</target>
    <target name="bib.xml" fun="Null">/bib/book/title</target>
    <target name="SCMFMA" fun="Null">/scmfma/book/title</target>
    <target name="bookdata.xml" fun="Null">/bookdata/book/title</target>
  </source>
```

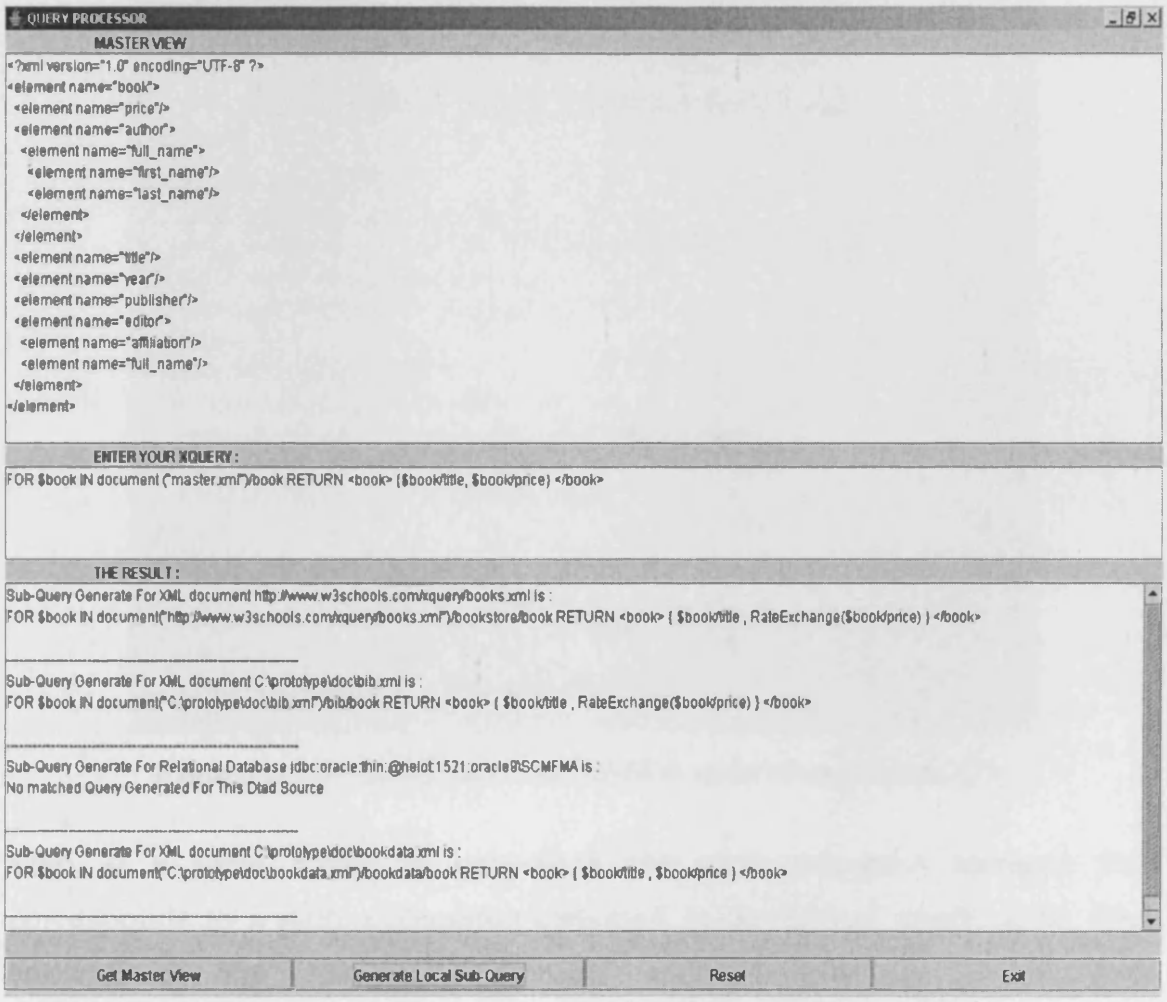
Figure 6.8: Some parts of XMKB used to translate Q2.

Figure 6.9 shows the results generated by local queries for global query Q2.

6.5.3 One-to-many query example

Q3: *FOR $edi IN document ("master.xml")/book WHERE $edi/title = "Database Systems" RETURN <editor> {$edi/editor/full_name} </editor>*

This is a more complex mapping case that can occur, when there is a path in the global schema mapped to many paths in a local schema. Q3 finds the editor's full name for the book titled 'Database Systems'. The translation process is similar to the two previous cases and has the following steps:



```
<?xml version="1.0" encoding="UTF-8" ?>
<element name="book">
 <element name="price"/>
 <element name="author">
  <element name="full_name">
   <element name="first_name"/>
   <element name="last_name"/>
  </element>
 </element>
 <element name="title"/>
 <element name="year"/>
 <element name="publisher"/>
 <element name="editor">
  <element name="affiliation"/>
  <element name="full_name"/>
 </element>
</element>
```

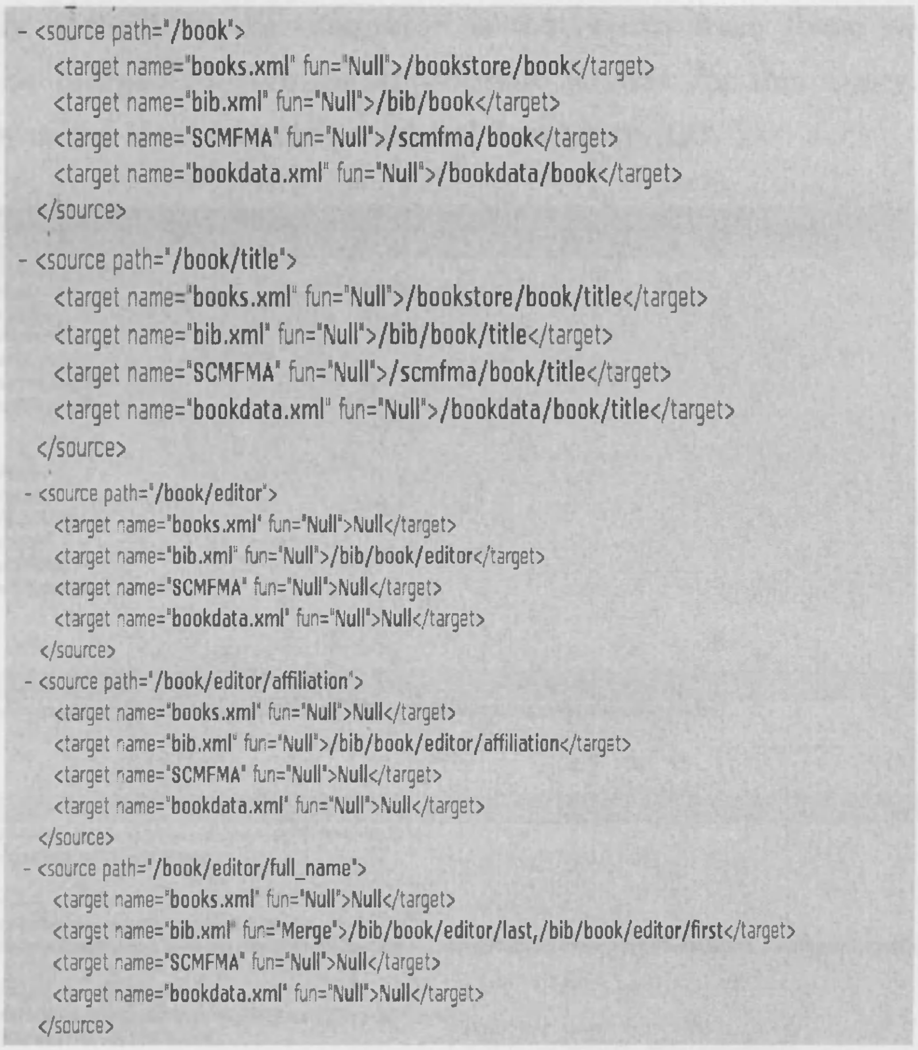
```
FOR $book IN document ("master.xml")/book RETURN <book> {$book/title, $book/price} </book>
```

```
Sub-Query Generate For XML document http://www.w3schools.com/xquery/books.xml is :
FOR $book IN document("http://www.w3schools.com/xquery/books.xml")/bookstore/book RETURN <book> { $book/title , RateExchange($book/price) } </book>

------------------------------------------
Sub-Query Generate For XML document C:\prototype\doc\bib.xml is :
FOR $book IN document("C:\prototype\doc\bib.xml")/bib/book RETURN <book> { $book/title , RateExchange($book/price) } </book>

------------------------------------------
Sub-Query Generate For Relational Database jdbc:oracle:thin:@helot:1521:oracle8\SCMFMA is :
No matched Query Generated For This Dtad Source

------------------------------------------
Sub-Query Generate For XML document C:\prototype\doc\bookdata.xml is :
FOR $book IN document("C:\prototype\doc\bookdata.xml")/bookdata/book RETURN <book> { $book/title , $book/price } </book>

------------------------------------------
```

Figure 6.9: The generated local queries from Q2.

Step 1: Q3 is parsed and */book*, *$edi/title*, and *$edi/editor/full_name* are detected as path expressions which represent global paths.

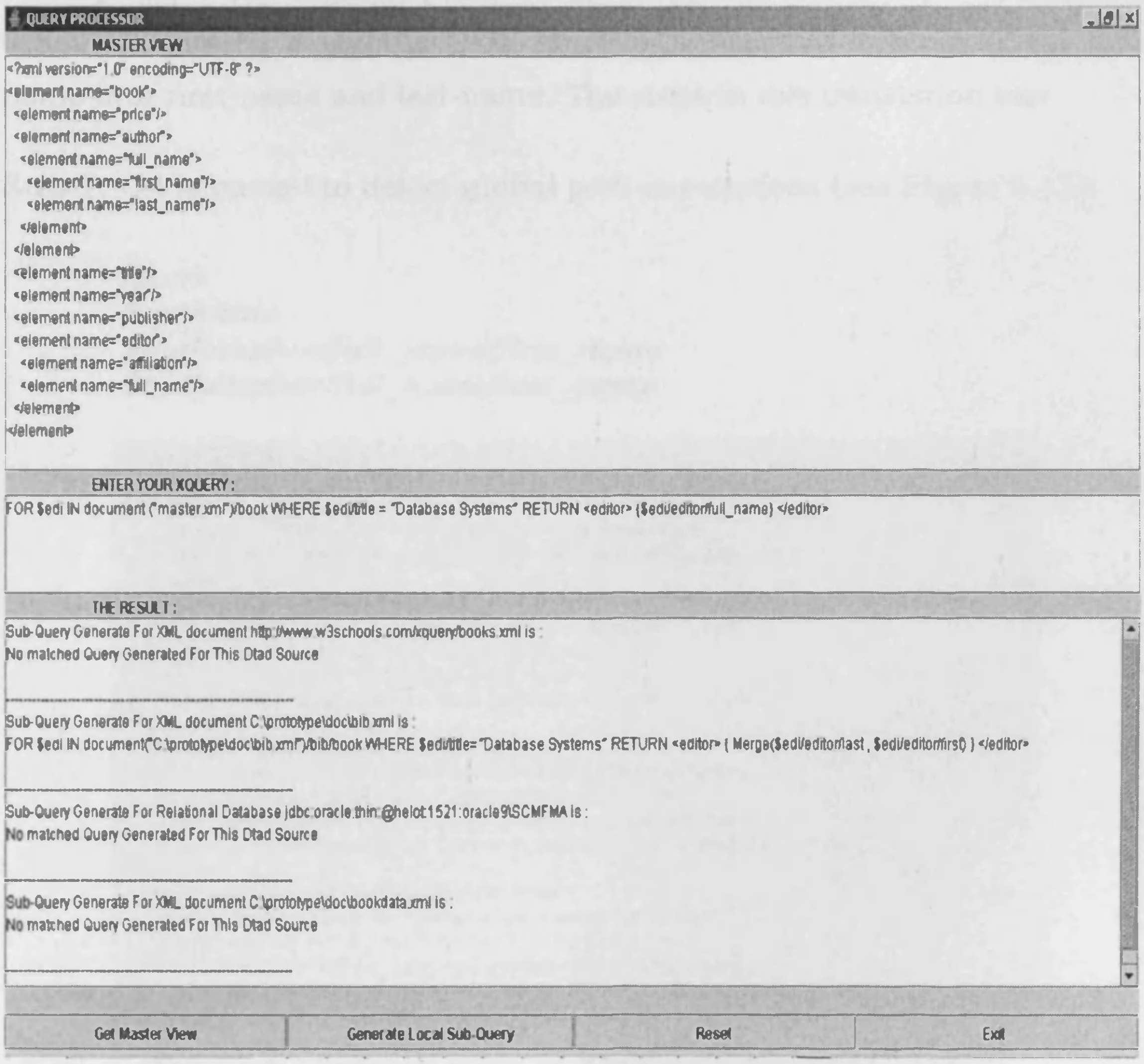
Step 2: The XMKB is invoked to obtain the corresponding local path and function names for each global path. Figure 6.10 shows parts of XMKB in which these paths appear.



```
- <source path="/book">
    <target name="books.xml" fun="Null">/bookstore/book</target>
    <target name="bib.xml" fun="Null">/bib/book</target>
    <target name="SCMFMA" fun="Null">/scmfma/book</target>
    <target name="bookdata.xml" fun="Null">/bookdata/book</target>
  </source>
- <source path="/book/title">
    <target name="books.xml" fun="Null">/bookstore/book/title</target>
    <target name="bib.xml" fun="Null">/bib/book/title</target>
    <target name="SCMFMA" fun="Null">/scmfma/book/title</target>
    <target name="bookdata.xml" fun="Null">/bookdata/book/title</target>
  </source>
- <source path="/book/editor">
    <target name="books.xml" fun="Null">Null</target>
    <target name="bib.xml" fun="Null">/bib/book/editor</target>
    <target name="SCMFMA" fun="Null">Null</target>
    <target name="bookdata.xml" fun="Null">Null</target>
  </source>
- <source path="/book/editor/affiliation">
    <target name="books.xml" fun="Null">Null</target>
    <target name="bib.xml" fun="Null">/bib/book/editor/affiliation</target>
    <target name="SCMFMA" fun="Null">Null</target>
    <target name="bookdata.xml" fun="Null">Null</target>
  </source>
- <source path="/book/editor/full_name">
    <target name="books.xml" fun="Null">Null</target>
    <target name="bib.xml" fun="Merge">/bib/book/editor/last,/bib/book/editor/first</target>
    <target name="SCMFMA" fun="Null">Null</target>
    <target name="bookdata.xml" fun="Null">Null</target>
  </source>
```

Figure 6.10: Some parts of XMKB used to translate Q3.

Step 3: a local query is generated for each *<target>* element that corresponds to a path expression detected in the global query paths. The content of the *<target>* element corresponding to global path *$edi/editor/full_name* in the XMKB is null for three of the data sources. This means this global query cannot be applied to these data sources. However, for the fourth data source, the *$edi/editor/full_name* global path is mapped to the local paths *Bib/book/editor/last* and */bib/book/editor/first*. This means these paths have the same index number and correspond to the path in the global schema. Also, the corresponding user-defined function (UDF) content is *Merge* which is a UDF function name. *Merge* was explicitly defined by the integrator as the results from these two paths should be merged to give the appropriate answer for this query. Figure 6.11 shows the local queries generated from query Q3.



```
<?xml version="1.0" encoding="UTF-8" ?>
<element name="book">
 <element name="price"/>
 <element name="author">
  <element name="full_name">
   <element name="first_name"/>
   <element name="last_name"/>
  </element>
 </element>
 <element name="title"/>
 <element name="year"/>
 <element name="publisher"/>
 <element name="editor">
  <element name="affiliation"/>
  <element name="full_name"/>
 </element>
</element>
```

```
FOR $edi IN document ("master.xml")/book WHERE $edi/title = "Database Systems" RETURN <editor> {$edi/editor/full_name} </editor>
```

```
Sub-Query Generate For XML document http://www.w3schools.com/xquery/books.xml is :
No matched Query Generated For This Dtad Source

--------------------------------------------
Sub-Query Generate For XML document C:\prototype\doc\bib.xml is :
FOR $edi IN document("C:\prototype\doc\bib.xml")/bib/book WHERE $edi/title= "Database Systems" RETURN <editor> { Merge($edi/editor/last , $edi/editor/first) } </editor>

--------------------------------------------
Sub-Query Generate For Relational Database jdbc:oracle:thin:@helot:1521:oracle9\SCMFMA is :
No matched Query Generated For This Dtad Source

--------------------------------------------
Sub-Query Generate For XML document C:\prototype\doc\bookdata.xml is :
No matched Query Generated For This Dtad Source

--------------------------------------------
```

Figure 6.11: The generated local queries from Q3.

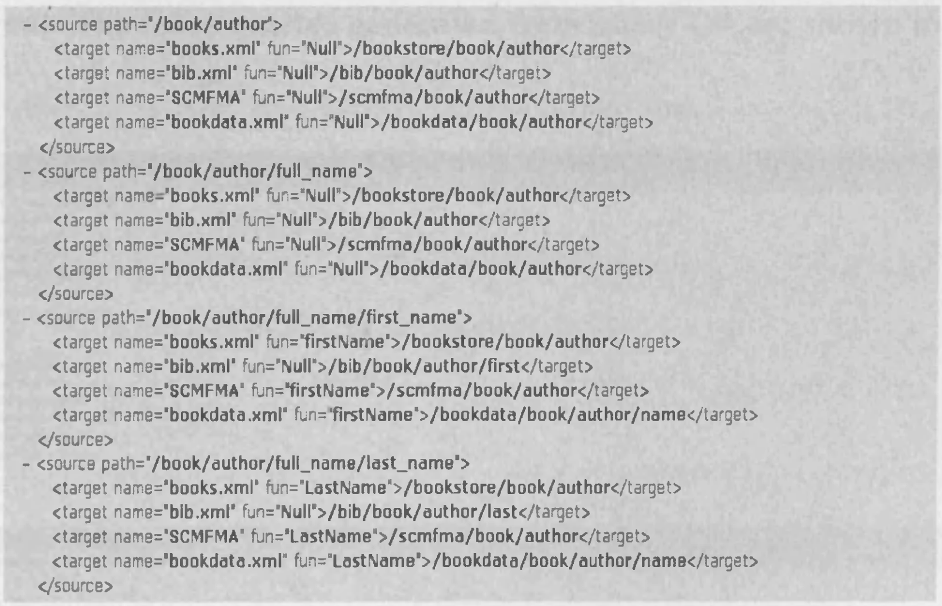
6.5.4 Many-to-one query example

Q4: *FOR $auth IN document ("master.xml")/book where $auth/title = "Data Structures" RETURN <author>{$auth/author/full_name/first_name, $auth/author/full_name/last_name} </author>*

This query shows a case in which two or more paths of the global schema correspond to one path in a local schema. The query lists the author's last and first names for books with a title 'Data Structures'. The *first_name* and *last_name* elements in the global schema are mapped to one element in some of the local schemas (see Figure 6.12), i.e. two element paths correspond to one element path. Thus, to translate this query into terms of the local sources, a specific UDF function is required to separate the full-name into first-name and last-name. The steps in this translation are:

Step 1: Q4 is parsed to detect global path expressions (see Figure 6.12):

/book
$auth/title
$auth/author/full_name/first_name
$auth/author/full_name/last_name



```
- <source path="/book/author">
    <target name="books.xml" fun="Null">/bookstore/book/author</target>
    <target name="bib.xml" fun="Null">/bib/book/author</target>
    <target name="SCMFMA" fun="Null">/scmfma/book/author</target>
    <target name="bookdata.xml" fun="Null">/bookdata/book/author</target>
  </source>
- <source path="/book/author/full_name">
    <target name="books.xml" fun="Null">/bookstore/book/author</target>
    <target name="bib.xml" fun="Null">/bib/book/author</target>
    <target name="SCMFMA" fun="Null">/scmfma/book/author</target>
    <target name="bookdata.xml" fun="Null">/bookdata/book/author</target>
  </source>
- <source path="/book/author/full_name/first_name">
    <target name="books.xml" fun="firstName">/bookstore/book/author</target>
    <target name="bib.xml" fun="Null">/bib/book/author/first</target>
    <target name="SCMFMA" fun="firstName">/scmfma/book/author</target>
    <target name="bookdata.xml" fun="firstName">/bookdata/book/author/name</target>
  </source>
- <source path="/book/author/full_name/last_name">
    <target name="books.xml" fun="LastName">/bookstore/book/author</target>
    <target name="bib.xml" fun="Null">/bib/book/author/last</target>
    <target name="SCMFMA" fun="LastName">/scmfma/book/author</target>
    <target name="bookdata.xml" fun="LastName">/bookdata/book/author/name</target>
  </source>
```

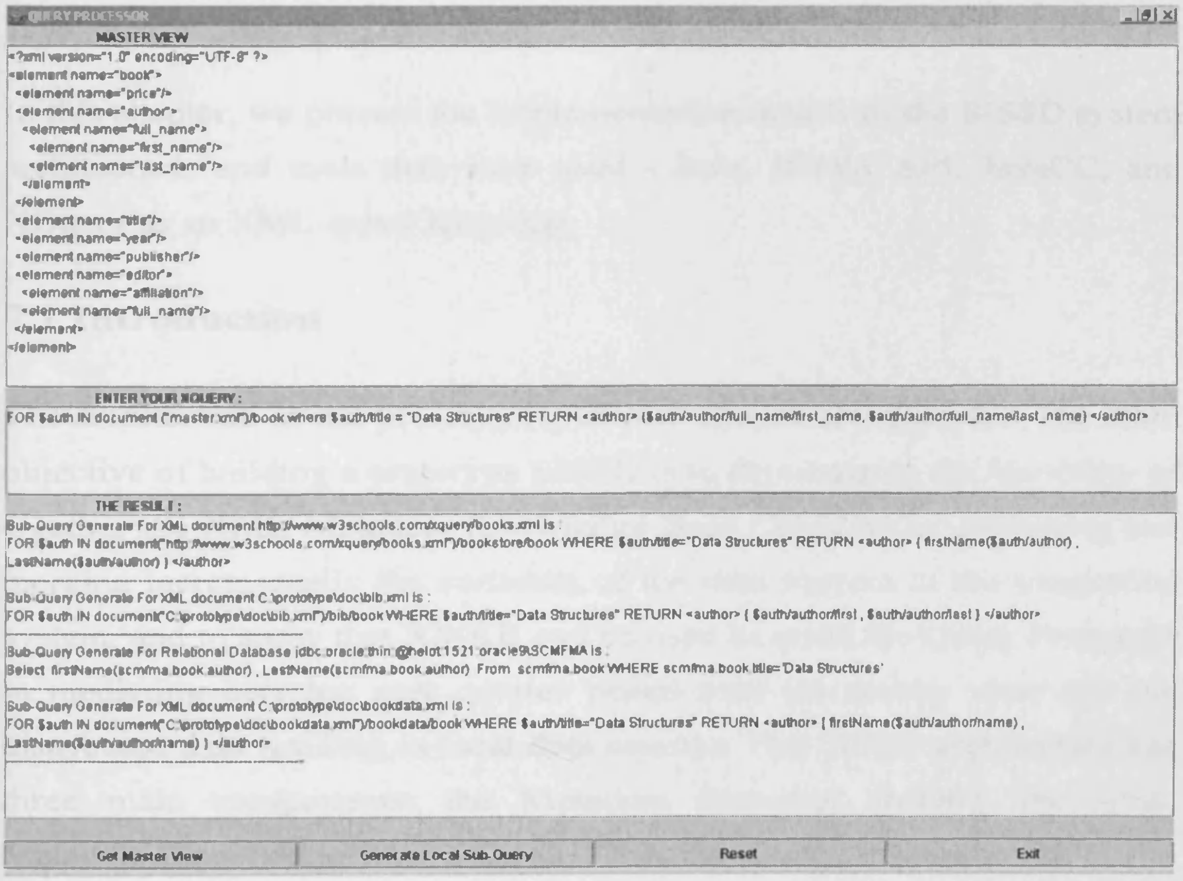
Figure 6.12: Some parts of XMKB used to translate Q3.

Step 2: The XMKB is read to obtain the corresponding local path and function name for each global path.

In this translation, no UDF function is required for the bib.xml data source since its UDF value is null. While for the other three data sources, the corresponding function values are *firstName* and *LastName* indicating two UDF functions are needed. Hence, the global paths

$auth/author/full_name/first_name and *$auth/author/full_name/last_name* are mapped to the paths */bookstore/book/author* in the books.xml data source, */scmfma/book/author* in the SCMFMA data source, and to */bookdata/book/author/name* in the bookdata.xml data source. Thus, one local path holds two different index numbers, i.e. it corresponds to two paths in the global schema. In addition, for each of these paths a UDF function name was explicitly defined by the designer. These functions are *firstName* to return the *first-name* and *LastName* to return the *last-name*. Hence, the *full-name* value is separated into first and last names in order to give the appropriate answer for this query.

Step 3: a local query is generated for each <*target*> element whose value is not null. The local queries generated from query Q4 are shown in Figure 6.13.



```
<?xml version="1.0" encoding="UTF-8" ?>
<element name="book">
 <element name="price"/>
 <element name="author">
  <element name="full_name">
   <element name="first_name"/>
   <element name="last_name"/>
  </element>
 </element>
 <element name="title"/>
 <element name="year"/>
 <element name="publisher"/>
 <element name="editor">
  <element name="affiliation"/>
  <element name="full_name"/>
 </element>
</element>
```

```
FOR $auth IN document ("master.xml")/book where $auth/title = "Data Structures" RETURN <author> {$auth/author/full_name/first_name, $auth/author/full_name/last_name} </author>
```

```
Sub-Query Generate For XML document http://www.w3schools.com/xquery/books.xml is :
FOR $auth IN document("http://www.w3schools.com/xquery/books.xml")/bookstore/book WHERE $auth/title="Data Structures" RETURN <author> { firstName($auth/author) ,
LastName($auth/author) } </author>
----------------------------------------
Sub-Query Generate For XML document C:\prototype\doc\bib.xml is :
FOR $auth IN document("C:\prototype\doc\bib.xml")/bib/book WHERE $auth/title="Data Structures" RETURN <author> { $auth/author/first , $auth/author/last } </author>
----------------------------------------
Sub-Query Generate For Relational Database jdbc:oracle:thin:@helot:1521:oracle9iSCMFMA is :
Select firstName(scmfma.book.author) , LastName(scmfma.book.author) From scmfma.book WHERE scmfma.book.title='Data Structures'
----------------------------------------
Sub-Query Generate For XML document C:\prototype\doc\bookdata.xml is :
FOR $auth IN document("C:\prototype\doc\bookdata.xml")/bookdata/book WHERE $auth/title="Data Structures" RETURN <author> { firstName($auth/author/name) ,
LastName($auth/author/name) } </author>
----------------------------------------
```

Figure 6.13: The generated local queries from Q4.

CHAPTER 7

The SISSD implementation

In this chapter, we present the implementation details of the SISSD system architecture, and tools that were used - Java, JDOM API, JavaCC, and XQuery as an XML query language.

7.1 Introduction

The architecture of our prototype system is shown in Figure 4.1. The main objective of building a prototype SISSD is to demonstrate the feasibility of creating the XML Metadata Knowledge Base (XMKB) by extracting and merging incrementally the metadata of the data sources in the integration system, and to show that XMKB can be used to assist the Query Processor in mediating between user queries posed over the master view and the distributed data residing in local data sources. The SISSD architecture has three main components: the Metadata Extractor (MDE), the XML Metadata Knowledge Base (XMKB) and the Query Processor (QP). The system was created in three stages, one stage for each system component:

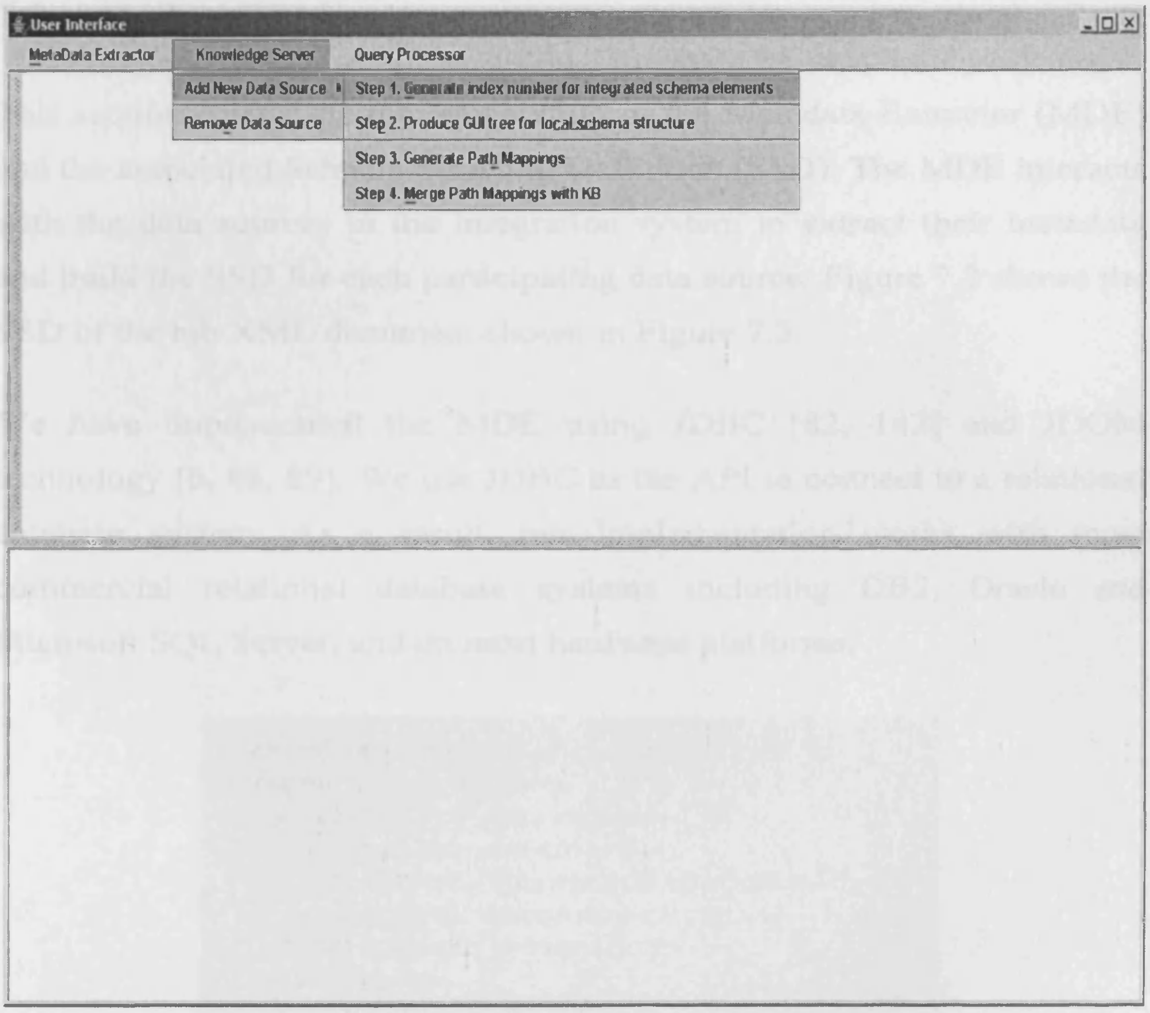

User Interface
MetaData Extractor
Knowledge Server
Query Processor
Add New Data Source
Remove Data Source
Step 1. Generate index number for integrated schema elements
Step 2. Produce GUI tree for local schema structure
Step 3. Generate Path Mappings
Step 4. Merge Path Mappings with KB

Figure 7.1: The main interface of SISSD system.

1. Creating the MDE to extract metadata and build the Schema Structure Definition (SSD) for each data source.

2. Creating the Schema Structure Definition (SSD) parser and the associated mapping process to establish and evolve the XML Metadata Knowledge Base (XMKB).

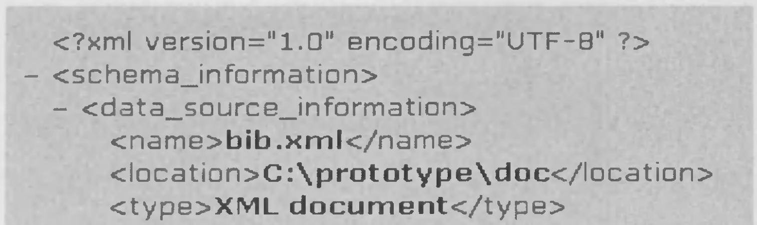
3. Creating the QP to parse and translate user queries into sub-queries which fit each local data source.

Appendix A shows the java code for the Main Interface of our SISSD prototype system (shown in Figure 7.1).

7.2 metadata extracting process

This section covers the implementation of the Metadata Extractor (MDE) and the associated Schema Structure Definition (SSD). The MDE interacts with the data sources in the integration system to extract their metadata and build the SSD for each participating data source. Figure 7.2 shows the SSD of the bib XML document shown in Figure 7.3.

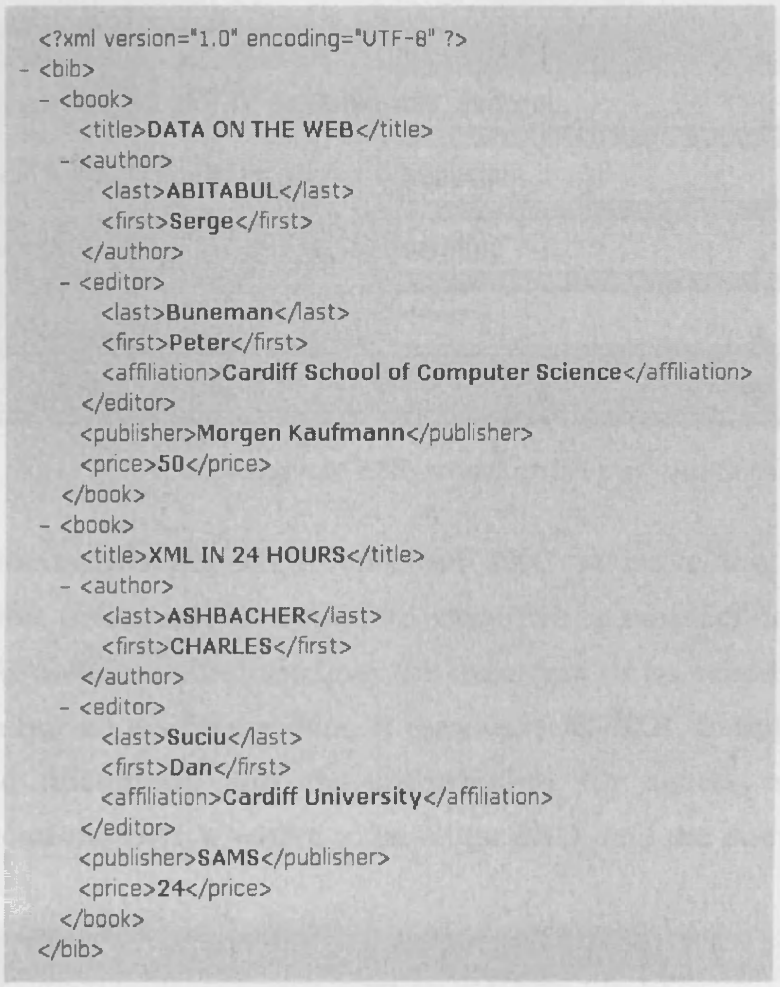
We have implemented the MDE using JDBC [82, 142] and JDOM technology [6, 88, 89]. We use JDBC as the API to connect to a relational database system. As a result, our implementation works with most commercial relational database systems including DB2, Oracle and Microsoft SQL Server, and on most hardware platforms.



```
  <?xml version="1.0" encoding="UTF-8" ?>
- <schema_information>
  - <data_source_information>
      <name>bib.xml</name>
      <location>C:\prototype\doc</location>
      <type>XML document</type>
    </data_source_information>
  - <structure>
    - <element name="bib">
      - <element name="book">
          <element name="title" />
        - <element name="author">
            <element name="last" />
            <element name="first" />
          </element>
        - <element name="editor">
            <element name="last" />
            <element name="first" />
            <element name="affiliation" />
          </element>
          <element name="publisher" />
          <element name="price" />
        </element>
      </element>
    </structure>
  </schema_information>
```

Figure 7.2: SSD of bib XML document.

We have developed JXC (Java XML Connectivity) (see Appendix C for the code) to build the Schema Structure Definition (SSD) of an XML document. This uses a JDOM (Java Document Object Model) interface to connect to the XML document, and detect and extract its metadata buried inside the data.

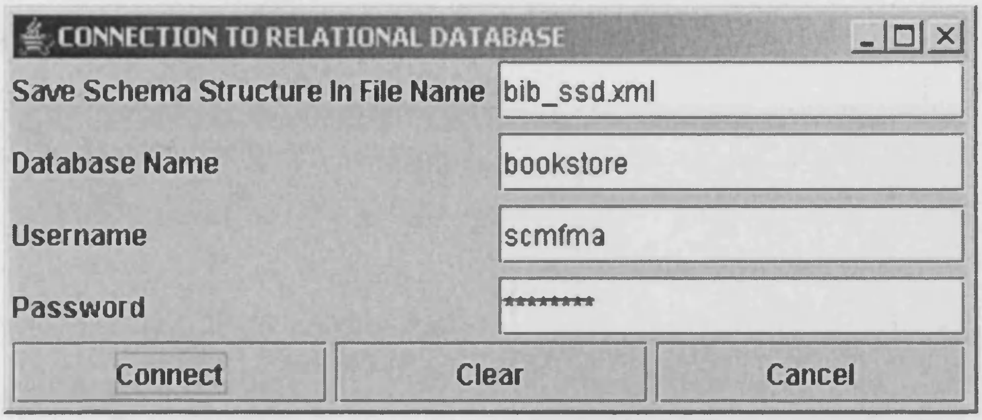


```
  <?xml version="1.0" encoding="UTF-8" ?>
- <bib>
  - <book>
      <title>DATA ON THE WEB</title>
    - <author>
        <last>ABITABUL</last>
        <first>Serge</first>
      </author>
    - <editor>
        <last>Buneman</last>
        <first>Peter</first>
        <affiliation>Cardiff School of Computer Science</affiliation>
      </editor>
      <publisher>Morgen Kaufmann</publisher>
      <price>50</price>
    </book>
  - <book>
      <title>XML IN 24 HOURS</title>
    - <author>
        <last>ASHBACHER</last>
        <first>CHARLES</first>
      </author>
    - <editor>
        <last>Suciu</last>
        <first>Dan</first>
        <affiliation>Cardiff University</affiliation>
      </editor>
      <publisher>SAMS</publisher>
      <price>24</price>
    </book>
  </bib>
```

Figure 7.3: bib XML document.

The MDE accesses data sources without making any changes to them. As the MDE requires no changes to the underlying data sources to access their metadata, it preserves their local autonomy.

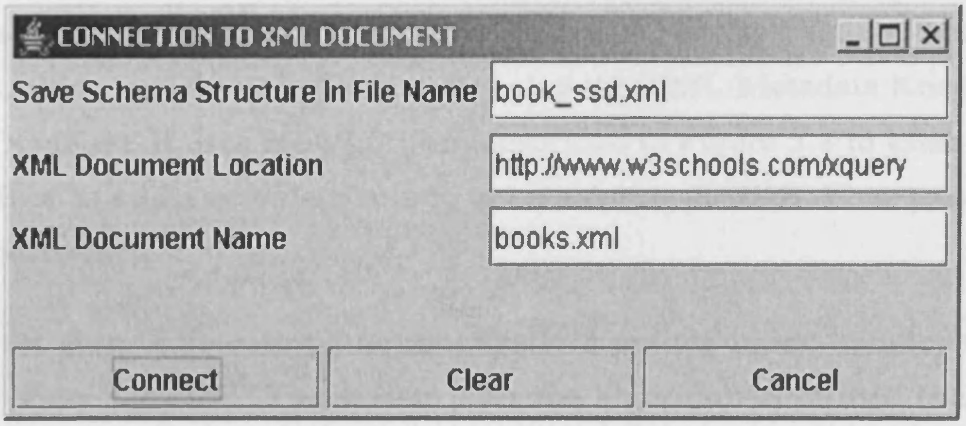
For relational databases the MDE employs JDBC to access the DB. The MDE accepts the information necessary to establish a connection to retrieve the metadata of the DBs schema and uses the XML Data Source Definition Language (XDSDL) (section 5.1) to build the target Schema Structure Definition (SSD) for that DB, and the necessary information for access, such as the DB location (URL), where to save the SSD, and the User ID and Password.


CONNECTION TO RELATIONAL DATABASE

Save Schema Structure In File Name	bib_ssd.xml
Database Name	bookstore
Username	scmfma
Password	********

Figure 7.4: Relational DB connection parameters.

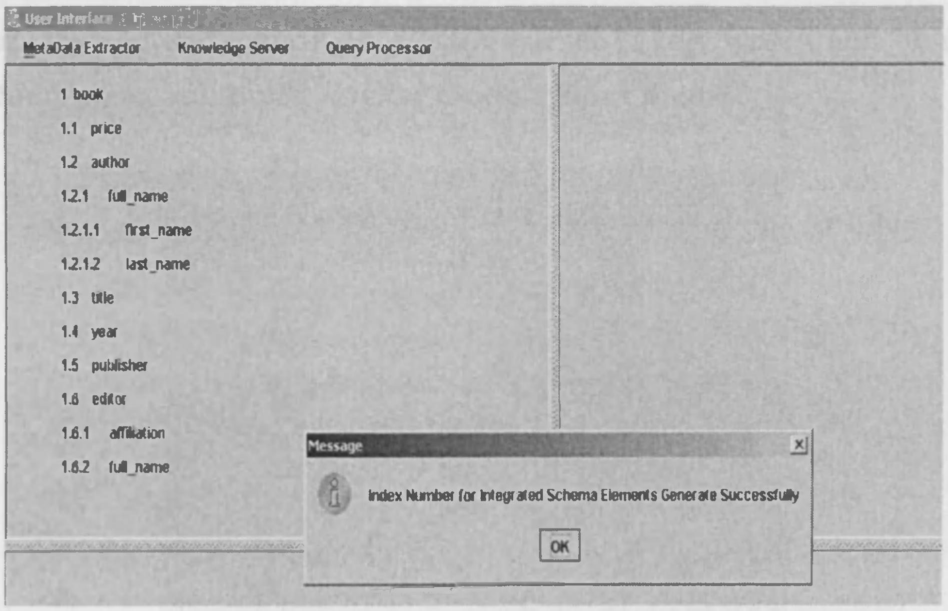
For XML documents the MDE employs JXC to make the access. The MDE gets the information needed to establish a connection to a well-formed XML document and retrieve the metadata of its schema where the metadata are buried inside the data. It then uses XDSDL to build the target SSD for the document, and the information for access, such as the document location (URL), where to save the SSD, and the document name.


CONNECTION TO XML DOCUMENT

Save Schema Structure In File Name	book_ssd.xml
XML Document Location	http://www.w3schools.com/xquery
XML Document Name	books.xml

Figure 7.5: XML document connection parameters.

Once the user has selected the type of data source (relational database or XML document) that is being accessed, the SSD is built using a graphical user interface, which allows the user to submit connection parameters. The interfaces for a relational database and XML document connection parameters are shown in Figure 7.4 and 7.5 respectively. Appendix B and C contain the Java class used to extract and build the SSD for a relational database and an XML document, respectively.



1 book
1.1 price
1.2 author
1.2.1 full_name
1.2.1.1 first_name
1.2.1.2 last_name
1.3 title
1.4 year
1.5 publisher
1.6 editor
1.6.1 affiliation
1.6.2 full_name

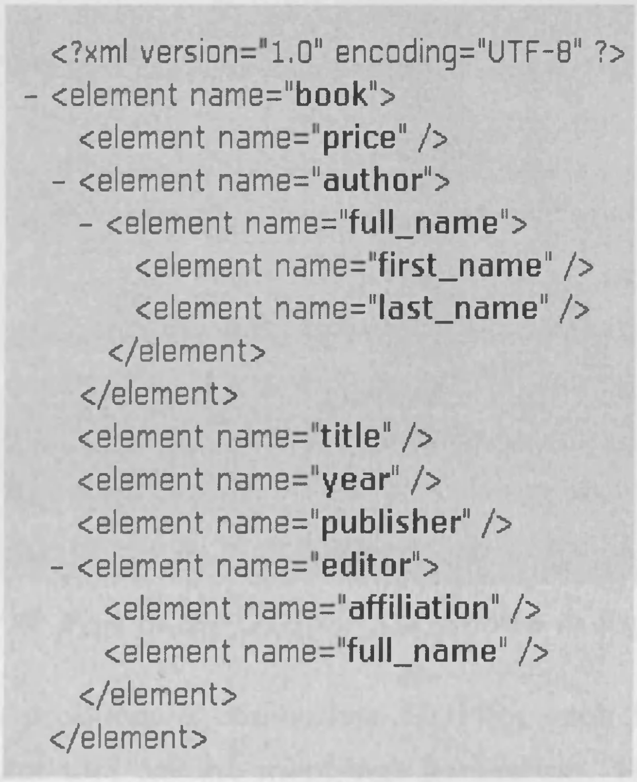
Index Number for Integrated Schema Elements Generate Successfully

Figure 7.6: Index numbers generated for master view shown in Figure 7.7.

7.3 XMKB establishing and mapping process

This section covers implementation of the SSD parsing and mapping process that is used to establish and evolve the XML Metadata Knowledge Base (XMKB). It uses the algorithm described in Figure 5.8 to establish an XMKB or to add a new data source to an existing XMKB. Four steps need to be performed.

The **first step** is generating automatically a unique index number for the master view elements. The system uses the algorithm described in section 5.4.3 to generate these index numbers. The parsing process is performed on the master view to extract and format the XML schema elements. JDOM is used to read and parse the master view document. The JDOM API reads XML documents in memory. JDOM is a tree-based, pure Java API which parses, creates, or manipulates XML documents. It provides a full document view with random access. Once a document has been loaded into memory, whether by creating it from scratch or by parsing it from a stream, it can be easily processed by JDOM. Thus the entire tree of an XML document is available at any time. JDOM itself does not include a parser. Instead it depends on a SAX parser [116], which can be used to parse documents and build JDOM models from them.



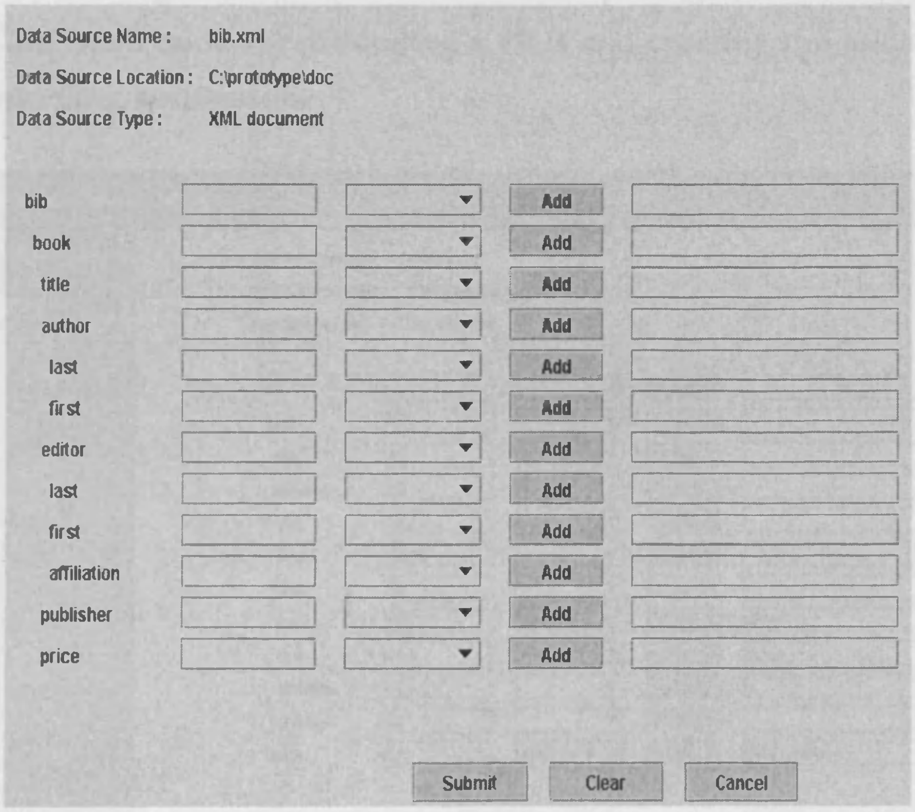
```
  <?xml version="1.0" encoding="UTF-8" ?>
- <element name="book">
    <element name="price" />
  - <element name="author">
    - <element name="full_name">
        <element name="first_name" />
        <element name="last_name" />
      </element>
    </element>
    <element name="title" />
    <element name="year" />
    <element name="publisher" />
  - <element name="editor">
      <element name="affiliation" />
      <element name="full_name" />
    </element>
  </element>
```

Figure 7.7: Master view.

JDOM provides Java specific XML functionality. A new builder is created to build a JDOM tree. In this case, a *SAXBuilder* (SAX class) has been used to build a JDOM tree of the form:

SAXBuilder builder = new SAXBuilder()

JDOM uses the default validating parser; a constructor is available for specifying an alternative validating parser. The JDOM code written to parse the master view, produce its tree structure and then generate the index numbers for its elements can be found in Appendix D. For example, Figure 7.6 shows the index numbers generated for the master view elements shown in Figure 7.7.

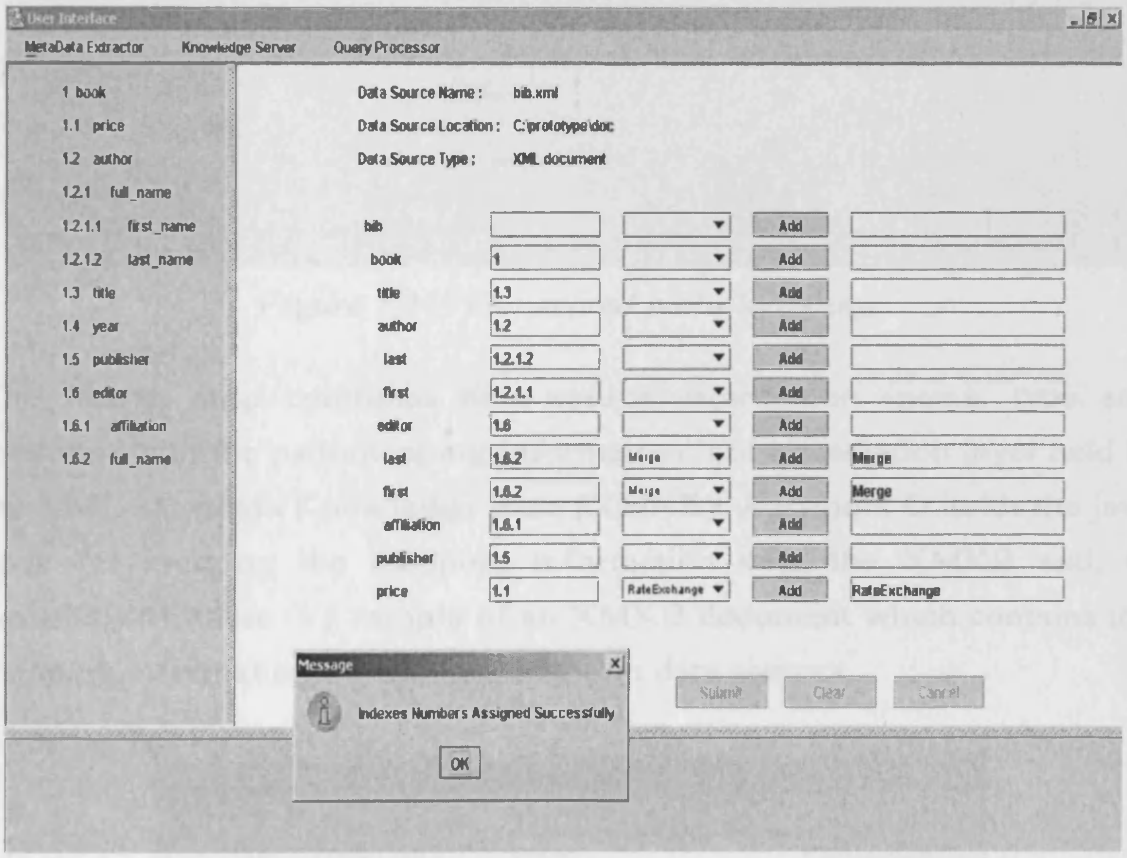


Data Source Name : bib.xml
Data Source Location : C:\prototype\doc
Data Source Type : XML document

bib
book
title
author
last
first
editor
last
first
affiliation
publisher
price

Figure 7.8: Part of the GUI for SSD shown in Figure 7.2.

The **second step** produces a convenient GUI for each local data source SSD as an assistant tool for the mappings generation. The JDOM API is used to read and parse the SSD. Once the SSD is parsed, the tree structure model is formed as a JDOM document object which contains all the components of the SSD. A GUI is generated based on the SSD tree structure model. Part of the GUI is shown in Figure 7.8 for the SSD shown in Figure 7.2. The first column shows the path hierarchy. The second column is used to assign a unique index number for the equivalence paths, while the third column is used to specify the function names which resolve heterogeneity conflicts by performing specific operations. The GUI is generated for each SSD to assign a unique index number to each path that corresponds to an equivalent global path and also a user-defined function name if it is needed. Figure 7.9 shows the interface for submitting index numbers and function names for the mapping between a master view (on the left of the figure) and the SSD shown in Figure 7.2. Appendix E contains the Java code for producing a GUI and creating the assistant tool for the mapping generation.



```
1 book
1.1 price
1.2 author
1.2.1 full_name
1.2.1.1 first_name
1.2.1.2 last_name
1.3 title
1.4 year
1.5 publisher
1.6 editor
1.6.1 affiliation
1.6.2 full_name
```

Data Source Name : bib.xml

Data Source Location : C:\prototype\doc

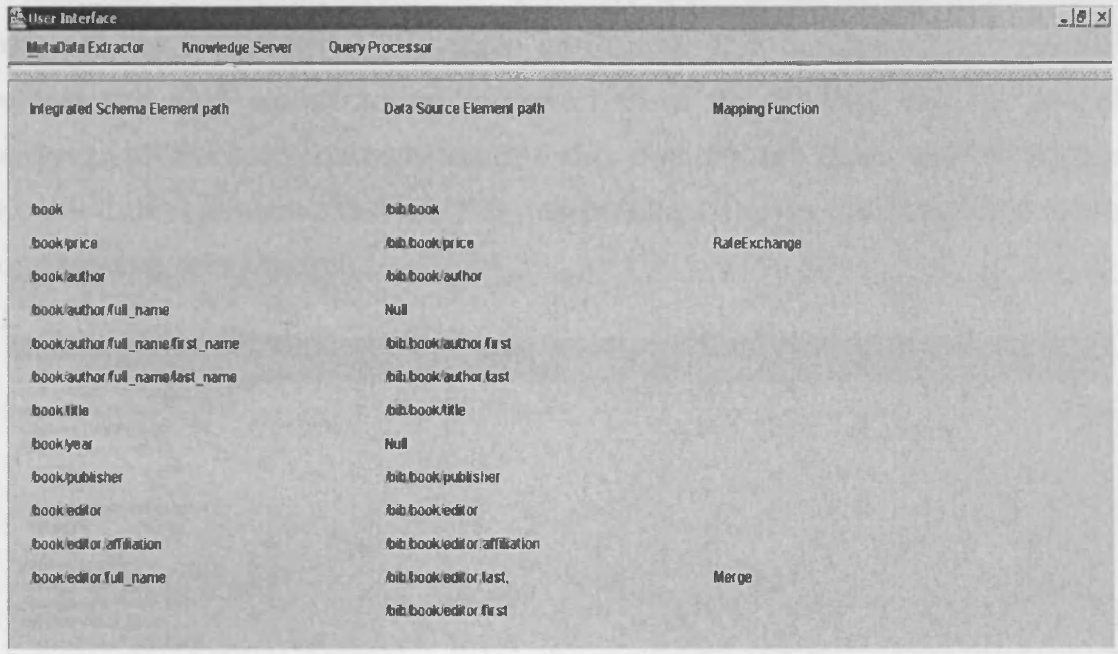
Data Source Type : XML document

bib			
book	1		
title	1.3		
author	1.2		
last	1.2.1.2		
first	1.2.1.1		
editor	1.6		
last	1.6.2	Merge	Merge
first	1.6.2	Merge	Merge
affiliation	1.6.1		
publisher	1.5		
price	1.1	RateExchange	RateExchange

Indexes Numbers Assigned Successfully

Figure 7.9: Interface for submitting index numbers.

The **third step** generates the mappings between the mater view paths and the local SSD paths based on information submitted using the GUI. This is done by collecting paths with the same index numbers which means they are equivalent paths with the same meaning. Figure 7.10 shows a path generated mapping. Appendix F has the Java code for generating the path mapping.



Integrated Schema Element path	Data Source Element path	Mapping Function
/book	/bib/book	
/book/price	/bib/book/price	RateExchange
/book/author	/bib/book/author	
/book/author/full_name	Null	
/book/author/full_name/first_name	/bib/book/author/first	
/book/author/full_name/last_name	/bib/book/author/last	
/book/title	/bib/book/title	
/book/year	Null	
/book/publisher	/bib/book/publisher	
/book/editor	/bib/book/editor	
/book/editor/affiliation	/bib/book/editor/affiliation	
/book/editor/full_name	/bib/book/editor/last, /bib/book/editor/first	Merge

Figure 7.10: Generated paths mapping.

The **fourth step** combines data source information (name, type and location) with the path mapping information into a mediation layer held in the XML Metadata Knowledge Base (XMKB). Appendix G holds the java code for merging the mapping information with the XMKB and, in Appendix H, there is a sample of an XMKB document which contains the mapping information for the four scenario data sources.

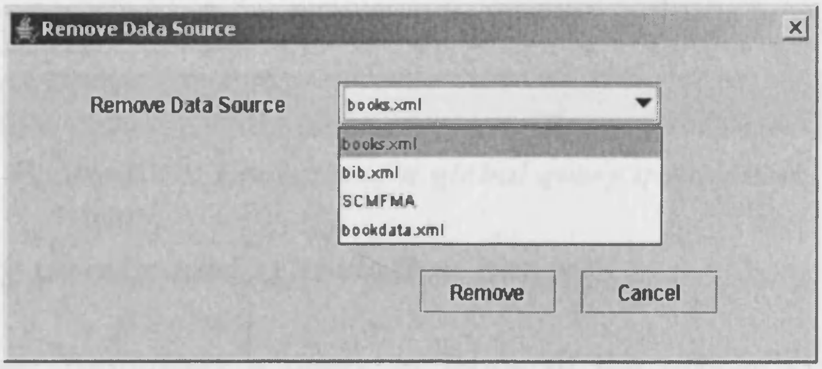
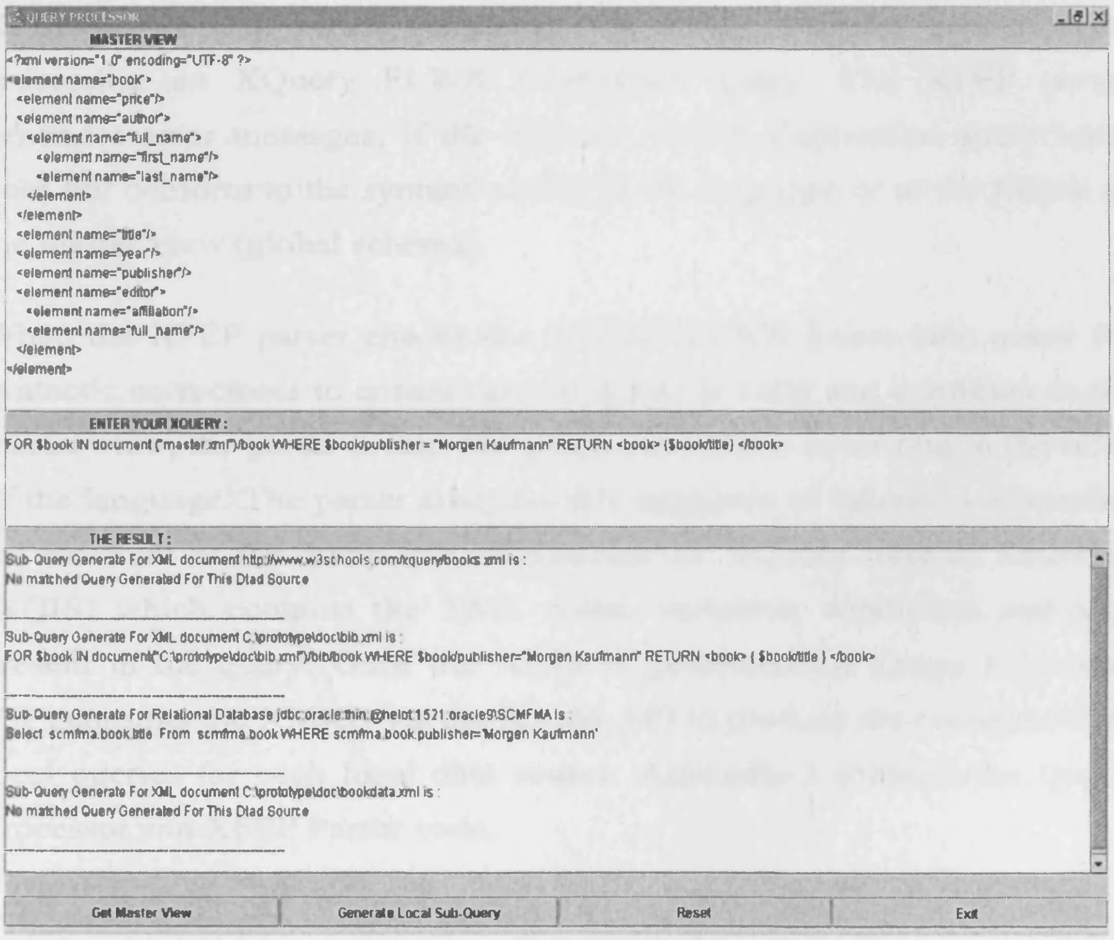


Figure 7.11: Interface for removing data source.

A data source can be removed from the integration system through the JDOM API using the interface shown in Figure 7.11. The API is used to access the XMKB to obtain the numbers and names of the data sources currently in the integration system and create this display. The user then selects the data source to be removed from the XMKB and the system removes all the local paths related to this data source from XMKB without the need to regenerate the XMKB. Appendix I shows the Java class which implements this action.



```
<?xml version="1.0" encoding="UTF-8" ?>
<element name="book">
  <element name="price"/>
  <element name="author">
    <element name="full_name">
      <element name="first_name"/>
      <element name="last_name"/>
    </element>
  </element>
  <element name="title"/>
  <element name="year"/>
  <element name="publisher"/>
  <element name="editor">
    <element name="affiliation"/>
    <element name="full_name"/>
  </element>
</element>
```

```
FOR $book IN document ("master.xml")/book WHERE $book/publisher = "Morgen Kaufmann" RETURN <book> {$book/title} </book>
```

```
Sub-Query Generate For XML document http://www.w3schools.com/xquery/books.xml is :
No matched Query Generated For This Diad Source
_______________________________________
Sub-Query Generate For XML document C:\prototype\doc\bib.xml is :
FOR $book IN document("C:\prototype\doc\bib.xml")/bib/book WHERE $book/publisher="Morgen Kaufmann" RETURN <book> { $book/title } </book>
_______________________________________
Sub-Query Generate For Relational Database jdbc:oracle:thin:@helot:1521:oracle9\SCMFMA is :
Select scmfma.book.title From scmfma.book WHERE scmfma.book.publisher='Morgen Kaufmann'
_______________________________________
Sub-Query Generate For XML document C:\prototype\doc\bookdata.xml is :
No matched Query Generated For This Diad Source
_______________________________________
```

Figure 7.12: Example of a global query translation.

7.4 Query parser and translation process

When a user formulates a query in terms of the master view (global schema) using XQuery FLWR expressions, the query is parsed by the query parser and the Query Processor generates the corresponding local queries. The system uses the algorithm described in section 6.3 to rewrite the user query as appropriate sub-queries for each local data source. For the XML query parser process, we developed a simple parser called **X**Query **F**LWR **E**xpression **P**arser (XFEP) which parses, lexically analyses the query, and breaks the XQuery FLWR Expression query into tokens which are classified. The XFEP parser is implemented in Java. Figure 7.12 shows an example of global query translation.

XFEP parser is a parser generator and lexical analyzer generator for processing an XQuery FLWR Expression query. The XFEP parser generates error messages, if the XQuery FLWR Expression query input does not conform to the syntactic rules of the language or to the format of the master view (global schema).

When the XFEP parser checks the XQuery FLWR Expression query for syntactic correctness to ensure that the query is valid and conforms to the master view, the parser breaks the query into tokens according to the rules of the language. The parser analyzes this sequence of tokens to determine the structure of the query and to generate the XQuery Internal Structure (XQIS) which contains the XML paths, variables, conditions and tags present in the query. Once the XQIS is generated the Query Processor (QP) consults the XMKB via the JDOM API to produce the corresponding local queries for each local data source. Appendix J contains the Query Processor and XFEP Parser code.

CHAPTER 8

Evaluation & Discussion

This chapter is an assessment of the project. We evaluate the functionality and flexibility of the system, and then discuss the suitability of its architecture and design. The XMKB construction process is then evaluated, followed by a discussion of the suitability of XML as the canonical data model in our integration system.

8.1 Evaluation

This evaluation is against the hypothesis and objectives in chapter 1.

The main emphasis of our work was to investigate the feasibility of building by a bottom-up approach an XML Metadata Knowledge Base (XMKB) to assist with the incorporation of heterogeneous distributed structured data residing in relational databases and semi-structured data held in well-formed XML documents into an integration system. This has been achieved by developing:

1) The MDE to extract metadata which is used to build the SSD of the data sources.
2) A tool for a meta-user (the metadata integrator) to describe mappings between the master view and local SSD of data sources.
3) An XML Metadata Knowledge Base (XMKB) to store this mapping information.
4) An architecture of software components which builds this XMKB and exploits its knowledge to assist the Query Processor to mediate between user queries posed over the master view of its heterogeneous data sources, and translate such queries into sub-queries suited to each local data source.

The efficiency and effectiveness of the outcomes of our research are measured in terms of the:

- Functionality of SISSD system with regard to its role as an integration tool for a meta-user and its role in helping users formulate queries and receive output.
- Flexibility of the SISSD system with regard to its suitability to a dynamic environment, where data sources can be added or removed without the need to restructure the master view.
- The architecture of the SISSD system with regard to its design and role as an integration tool.
- Construction of the XMKB with regard to its structure and role as a central repository which stores the mappings information.
- Choice of XML as the data model of our data integration system, and the language to describe the SSD for the participating data sources.

- Capability of SISSD system with regard to handling different types of heterogeneity that may exist between a set of data sources.
- Different uses of the system and the types of user who can use it.

8.1.1 Functionality of SISSD

The SISSD system is a valuable integration tool for a meta-user who does the metadata integration of heterogeneous distributed structured data residing in relational databases and semi-structured data held in well-formed XML documents produced by internet applications – in that it facilitates the efficient production of an XML Metadata Knowledge Base (XMKB) from the extracted metadata of the participating data sources. The generation of an XMKB is simplified in the SISSD system by its graphical interface tool which guides a meta-user step by step through the integration process via system windows that hide low-level and tedious details while enabling the user to concentrate on the parameters that need to be supplied at each stage to describe mappings between the master view and local SSD data sources [16]. The XMKB contains the data source information (name, type and location), meta-information about relationships of paths among data sources, and function names for handling semantic and structural discrepancies.

The SISSD system is valuable to a user at run time, where it allows system users to formulate their queries against the master view. The queries are then transformed into queries against the underlying local data sources. At the heart of our system there is a Query processor (QP) module which mediates user queries posed over the master view of the heterogeneous data sources, by automatically translating such queries into sub-queries, which are suited to each participating data source and which will retrieve information relevant to the query. The QP consults and exploits the mapping information stored in the XMKB at several stages, to obtain the local paths corresponding to the master paths, to find the function names for handling semantic and structural discrepancies, and then to produce semantically equivalent queries to fit each local data source.

In the QP we have created a query translation (rewriting) algorithm which is used by the QP module to achieve this task [14]. The algorithm is both conceptually simple and generally applicable. We have successfully implemented and tested it on a variety of relational and XML data source integration examples in our prototype SISSD system.

8.1.2 Flexibility of SISSD system

The SISSD system is flexible in that its users can assemble virtual master views suited to their requirement. For the same set of data sources users may create different master views, depending on their interest. It also preserves the local autonomy of the participating data sources, thus these data sources can be joined to the system without rebuilding or modification to the local data source to prepare it for the SISSD.

The SISSD uses a local-as-view approach to map between the master view and the local schema structures. This approach is well-suited to a dynamic environment, where data sources can be added or removed from the system without the need for a major restructure of the master view. The information required for the new sources is easily added, and if a source is removed only the information related to it is deleted. Also, the LAV approach provides a more flexible environment to meet users' evolving and changing information requirements across the disparate data sources available over the global information infrastructure (Internet) as time passes. It is better suited and scalable for integrating a large number of autonomous read-only data sources accessible over communication networks than integration systems created in traditional ways. Furthermore the LAV approach provides a flexible environment able to accommodate the continual change and update of data source schemas. This makes it especially suitable for XML documents on Web servers since these remote documents are not static and are often subject to frequent update. When generating the XMKB, the mapping direction is changed from the original local-as-view to global-as-view, to make its use in the query rewriting stage straightforward. A master query from a user is translated into queries to local data sources by looking up the corresponding paths in the XMKB. Hence the SISSD combines both global-as-view and local-as-view approaches taking advantage of the approach best suited to the task.

The SISSD also gives the flexibility to remove any data source schema from the XMKB and then add this data source again with an updated or altered schema without any other impact on the XMKB, or the need to regenerate it from scratch every time.

8.1.3 Architecture of SISSD system

In a typical data integration system, users and component data sources are scattered over a number of nodes of the computer network, and users are provided with front-end interface(s) to access data stored in the different back-end data sources. The design architecture of the SISSD system (section 4.3) is based on a client-server model.

SISSD system has been developed as a collection of software modules. They are:

- JXC (Java XML Connectivity) which detects and extracts the Schema Structure Definition (SSD) of a well-formed XML document.

- MDE (Metadata Extractor) which extracts the metadata of all data sources and builds a Schema Structure Definition (SSD) in XML form for each data source.

- MVP (Master View Parser) which parses the master view to generate a tree structure and then automatically generates unique index numbers for its elements.

- SSDP (Schema Structure Definition Parser) which parses the SSD of the data source to generate a tree structure and then produces the GUI for it.

- KS (Knowledge Server) which establishes, evolves and maintains the XML Metadata Knowledge Base (XMKB).

- QP (Query Processor) which receives a user query over the master view and automatically rewrites it into sub-queries, fitting each local data source, and integrates the results of these sub-quires.

The SISSD system architecture is a collection of modules. This makes it easy to develop and incorporate new modules to enhance the functionality of the prototype. The meta-users (integrators) interact with the software modules in the SISSD system through a GUI provided by the system. It serves as a common front-end for all users. This enables them to interact with the MDE, KS and QP modules.

8.1.4 Construction of the XMKB

The XMKB has been developed as a central repository which stores the data source's information (names, types and locations) and metadata extracted from the data sources, in which the mappings between the master view and Schema Structures Definition (SSD) of the data sources are defined, so that this information can be used to support and improve the integration of distributed heterogeneous structured data residing in relational databases and semi-structured data held in well-formed XML documents. The information stored in this XMKB is available to the Query Processor (QP) to mediate between user queries posed over the master view and the distributed heterogeneous data sources, to automatically rewrite such queries into sub-queries, fitting each local data source. This enables the Query Processor (QP) to reuse the knowledge held in the XMKB for other user queries posed over the master view. Typically, the knowledge held in the XMKB becomes available incrementally, as new data sources join the integration system. This means that the XMKB must be able to evolve. The XMKB has a simple, flexible and easy to understand structure which allows it to be evolved and modified incrementally as new data sources are added or removed from the system, without the need to regenerate it from scratch. Its simple structure makes it easy to construct it automatically. The XMKB is however able to capture the structure and semantics of the schema elements of the data sources so that this information can be used to resolve semantic and structural discrepancies occurring in the data.

We have developed a software module to automatically generate a tool for a meta-user (integrator) to define the semantic relationships between the schema's elements. However, these semantic relationships cannot be determined precisely using an automatic procedure. Thus this task always requires some human intervention and is semi- automatic for this reason.

8.1.5 Choice of XML as the data model

Many data models are based on some form of a labeled directed graph. One of the most popular is the Object Exchange Model or the OEM model. OEM is a simple, self-describing nested object model [124]. However, the eXtensible Markup Language (XML) received significant attention from the database community when the W3C recommended it as a standard for data representation and exchange in the World Wide Web. XML has a strong resemblance to semi-structured data models and could easily represent structured, semi-structured and unstructured data. We consider

XML to be an ideal candidate to provide a unifying data model in data integration systems for several reasons, namely:

1. XML is flexible and powerful enough to represent a wide variety of data models without losing their semantics. This lossless semantic conversion enables XML to represent structured, semi-structured and unstructured sources equally well.
2. Unlike OEM models which lack a well-defined schema, XML can represent schema information.
3. Its recommendation as a standard by W3C and its backing by enterprises has resulted in rich tool support for XML.
4. Standardization efforts have led to XML query languages like XPath, and XQuery appearing.
5. XML is not tied to any particular platform, architecture or organization.

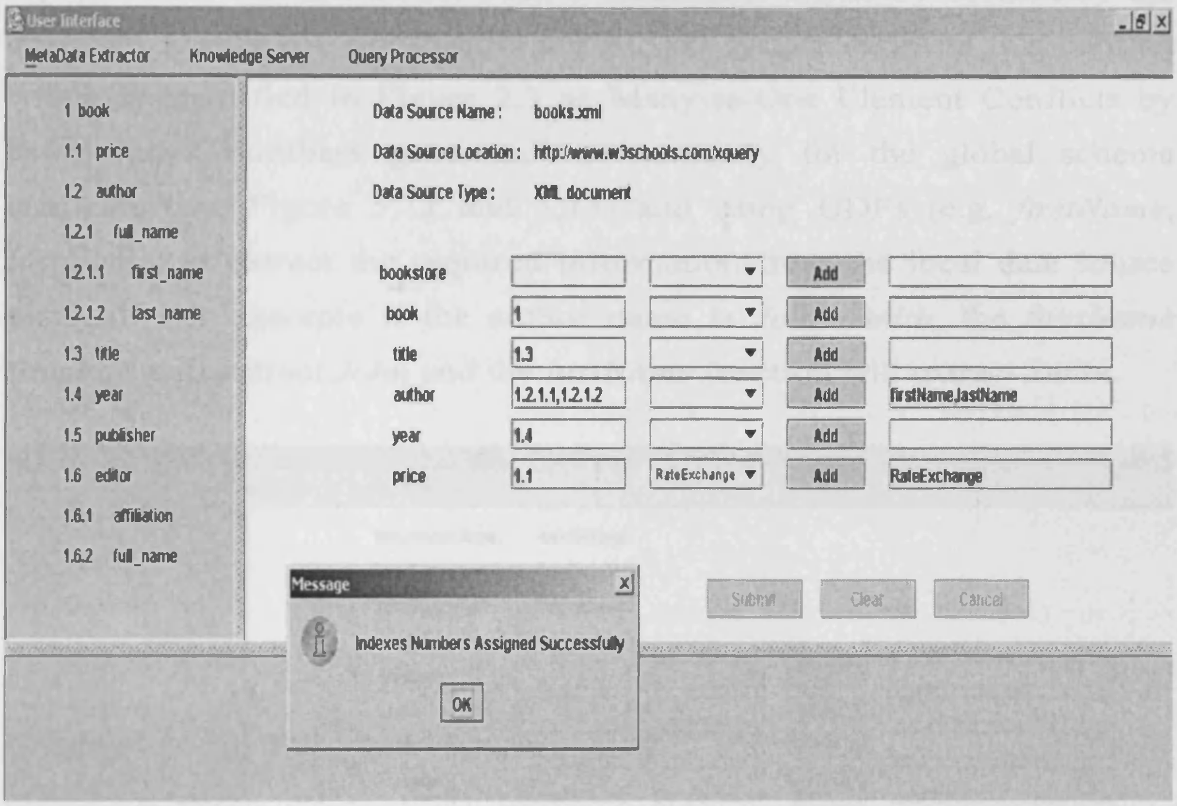
In the SISSD system we want to represent the structure of a data source joining the integration system as this is crucial for data integration. We have therefore defined a simple definition language called XML Data Source Definition Language (XDSDL) which abstracts the structure of schema elements to build the Schema Structure Definition (SSD) of the data source. This language uses a simple grammar similar to the XML Schema Language but omits information such as data types. Furthermore, this language describes the *actual* structure of a data source not the *possible* one as would be defined by a DTD and XML Schema Language definition. We have developed a software module to automate the task of building an SSD. Thus, by using this module a meta-user (integrator) can construct a SSD semi-automatically, which captures the structure of a given data source.

8.1.6 Handling different types of heterogeneity

In the SISSD system we are concerned with the higher level of heterogeneity. This area can be further divided into three levels of heterogeneity: *syntactic heterogeneity*, *structural heterogeneity* and *semantic heterogeneity*. This classification of heterogeneity is one of several classifications of the different types of higher level of heterogeneity that may exist between a set of independently designed data sources. We chose this classification to show that our SISSD system can deal with different levels of heterogeneity (*syntactic heterogeneity*, *structural heterogeneity* and *semantic heterogeneity*) and provide solutions to the problems at these different levels of *interoperability*. The conflict types identified in Figure 2.3 can be classified into one of these three levels of heterogeneity. In this section we show how our SISSD system can handle these levels of heterogeneities.

Syntactic heterogeneity refers to the encoding of the same concept in different data models (or natural languages); in other words using a different data model for storing similar data, examples are systems using a relational and XML model.

Mainly, syntactic heterogeneity addresses the problem of using different data models. Our approach is concerned with data sources that contain relational data and XML data. This type of heterogeneity in our system can be resolved by using a *Common Data Model (CDM)* and translating all data source schemas to this common model using transformation rules that explain how to translate schemas into the target data model. This task is done by the Metadata Extractor (MDE) (see section 4.3) that interacts with the data sources to extract the metadata and map the schemas to this CDM. The chosen CDM must be such that it is expressive enough to capture the meaning of all local data models. The XML data model is a suitable CDM and has been used for this purpose in several projects [72, 113] and was chosen in this project also.



1 book
1.1 price
1.2 author
1.2.1 full_name
1.2.1.1 first_name
1.2.1.2 last_name
1.3 title
1.4 year
1.5 publisher
1.6 editor
1.6.1 affiliation
1.6.2 full_name

Data Source Name : books.xml
Data Source Location : http://www.w3schools.com/xquery
Data Source Type : XML document

bookstore			
book	1		
title	1.3		
author	1.2.1.1,1.2.1.2		firstNameLastName
year	1.4		
price	1.1	RateExchange	RateExchange

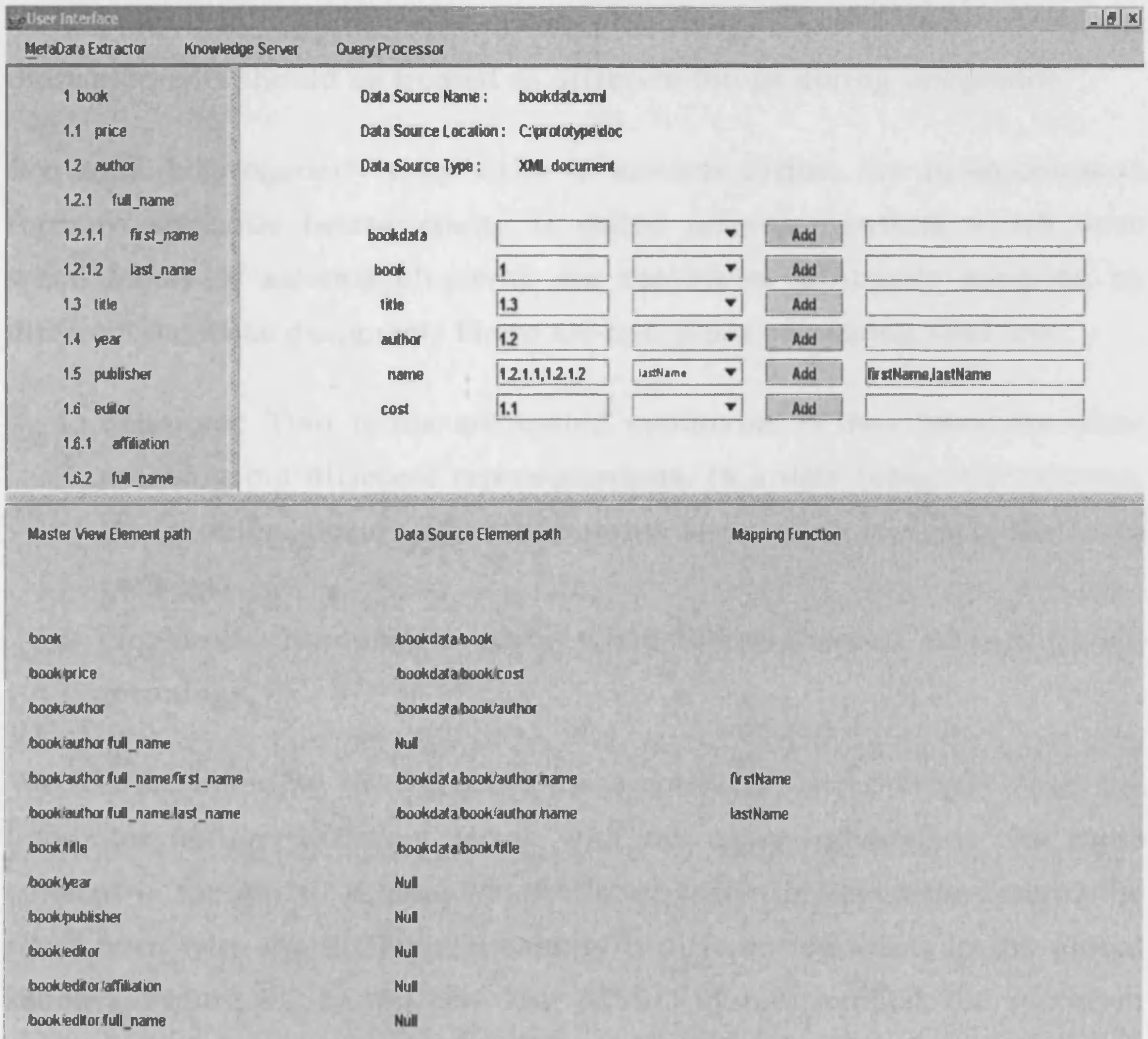
Indexes Numbers Assigned Successfully

Figure 8.1: Example of resolving structural heterogeneity.

Structural heterogeneity arises when the same concept is represented differently, in other words when elements have the same meaning, are modeled with the same data model, but structured and represented in a different way.

In dealing with structural heterogeneity, the main difficulty is that the data in different sources may be represented in different formats and in incompatible ways. Therefore, we have to provide an appropriate mechanism to handle this kind of heterogeneity conflict. It can happen for example, when one source represents authors' names as full names, while the global schema separates the first and last names. In this case, the name from the local source must be separated into its parts, if a query is to retrieve the first name of the author. Therefore, user-defined functions

(UDFs) are needed to perform the required operation for resolving this case. The tasks of these functions are to split the author full name into separate first and last name. Such a function is explicitly defined by the designer. Figure 8.1 shows how our SISSD system resolved this conflict which is identified in Figure 2.3 as Many-to-One Element Conflicts by using index numbers generated automatically for the global schema elements (see Figure 5.12 and 5.13) and using UDFs (e.g. *firstName*, *lastName*) to extract the required information from the local data source element. For example if the author name is *John Smith,* the *firstName* function will extract *John* and the *lastName* function will extract *Smith*.



1 book
1.1 price
1.2 author
1.2.1 full_name
1.2.1.1 first_name
1.2.1.2 last_name
1.3 title
1.4 year
1.5 publisher
1.6 editor
1.6.1 affiliation
1.6.2 full_name

Data Source Name : bookdata.xml
Data Source Location : C:\prototype\doc
Data Source Type : XML document

bookdata			
book	1		
title	1.3		
author	1.2		
name	1.2.1.1,1.2.1.2	lastName	firstName,lastName
cost	1.1		

Master View Element path	Data Source Element path	Mapping Function
/book	/bookdata/book	
/book/price	/bookdata/book/cost	
/book/author	/bookdata/book/author	
/book/author/full_name	Null	
/book/author/full_name/first_name	/bookdata/book/author/name	firstName
/book/author/full_name/last_name	/bookdata/book/author/name	lastName
/book/title	/bookdata/book/title	
/book/year	Null	
/book/publisher	Null	
/book/editor	Null	
/book/editor/affiliation	Null	
/book/editor/full_name	Null	

Figure 8.2: Example of handling synonym conflict.

The distinction between semantic and structural heterogeneity is not always clear-cut. Structural heterogeneity refers basically to the structure of the data, while semantic heterogeneity refers to the represented concepts' interpretation.

Semantic heterogeneity refers to the fact that data represented in different systems in similar ways may be subject to different interpretation. For example, two schema elements in two local data sources can have the same intended meaning, but different names. Thus, during integration, it should be realized that these two elements actually refer to the same concept. Alternatively, two schema elements in two data sources might be named identically, while their intended meanings are incompatible. Hence, these elements should be treated as different things during integration.

Semantic heterogeneity may exist in several forms; the most common form of semantic heterogeneity is called *naming conflicts* which arise when labels of schema elements are somewhat arbitrarily assigned by different database designers. There are two types of naming conflicts:

1. *Synonym*: Two terms are called synonyms if they have the same meaning but different representations. In a data integration system, this problem occurs when two terms are used to represent the same concept.
2. *Homonym*: homonyms occur when identical terms have different meanings.

We use the mapping to overcome these conflicts. In the former case, the integrator assigns different terms with the same meaning to the same concept in the global schema. In the latter case, the integrator assigns the same term with the different meaning to different concepts in the global schema. Figure 8.2 shows how our SISSD system handles the synonym conflict which is identified in Figure 2.3 by using index numbers generated automatically for the global schema elements (see Figure 5.12 and 5.13) and assign these index numbers to the elements that are synonyms in the local schema structures. For example, in Figure 8.2 the index number (1.1) of element *price* in the global schema is assigned to the element *cost* in the local schema.

8.1.7 Ways of using the system

Different users have different reasons for integrating data sources, and even the same user might need to integrate the same data in a variety of ways and/or include different services to satisfy different tasks in an organization. Thus a tool that supports the flexible integration of pre-existing structured and semi-structured data sources needs to allow different views and reasons for the integration to be handled. The primary motivation for most of the work in this area is that many applications require processing of data that belongs to structured and semi-structured data sources. For instance, an order processing application might need to handle inventory information in a relational database as well as purchase orders received as (semi-structured) XML documents [126].

Our system enables the users to link data from different structured and semi-structured data sources flexibly. It provides a tool that can be used by the integrator or the end user to do the metadata integration. Furthermore, it gives the user who does the metadata integration the option to choose which master view to use so that his/her current requirements are met. It also allows choice of the data sources that will be integrated and in which order the integration will be performed. This also gives this user the possibility to change and edit a master view.

The system can be used in two different ways:

1. In a centralized approach, when one person is the integrator (skilled in IT) constructs the master view that characterizes the underlying

data sources, then integrates the participating schema structures as they are presented for integration and creates the UDF to resolve the heterogeneity conflicts by performing specific operations. This approach is tightly-coupled in that data is accessed using a global view(s) created and managed by the integrator(s). It is appropriate when there are a small number of data sources which are permanent and their schema structures do not change frequently.

2. In a customized approach, when several users are integrators, each chooses a view as the initial master view that meets the requirements and decides on which schemas to integrate and in what order. However, the user in this case will provide a library of functions to locate the appropriate UDF to resolve conflicts. This approach is loosely-coupled, in that it is the user's responsibility to create and maintain the integration regime. This approach provides a more flexible environment which meets the users' evolving and changing information requirements across the disparate data sources available over the global information infrastructure (Internet). It is better suited to the integration of a large number of autonomous read only data sources accessible over communication networks, especially when these data sources are subject to continual change.

8.2 Discussion

One of several trends that have significant impact on the use of database technology is XML. The power of XML as a description language lies in the fact that an XML document contains a self-description of hierarchically structured data, and it has the ability to associate markup terms with data elements (see section 8.1.5). These markup terms serve as metadata allowing a formalized description of the content and structure of the accompanying data. XML can subsume HTML and its successor

XHTML as the communication language for the Web and it provides a structure in a widely accepted format.

As the importance of XML has increased, the W3C has introduced the XML Schema language to replace the DTD (Document Type Definition) grammar language. The DTD mechanism has numerous limitations. A basic and major limitation is that a DTD is not a valid XML document. Therefore it must be handled by XML parsing tools in a special way. Furthermore, DTDs have a very limited capability for specifying data types. Also, DTDs are quite limited in their ability to constrain the structure and content of XML documents. In addition, they cannot handle namespace conflicts within XML structures or describe complex relationships among documents or elements. They also are not modular, and can not derive new type definitions based on an existing definition.

An XML Schema Definition (XSD) is an XML-based grammar declaration for XML documents. The motivation for using and developing an XML Schema was dissatisfaction with DTDs. It was developed in response to the limitations of the DTD mechanism, and was a tremendous advance over DTDs. XML Schema allows very precise definition for both simple and complex data types, and allows derivation of new type definitions.

The definition language that the SISSD system used to build the Schema Structure Definition (SSD) is similar to the XML Schema Language but omits information such as data types. This language is the XDSDL, which is used in our system to abstract the schema structure of the data sources joining the integration system. The XDSDL avoids the complexity of the XML Schema Language by using a simple notation to describe the structure of the schema elements.

For the foreseeable future, a great quantity of data will continue to be stored in relational database systems because of the reliability, scalability, tools and performance associated with these systems [68, 133]. Additionally, much interesting and useful data can be published in well-formed XML documents by Web-based applications and Web services or by human-coding [102].

While the availability of data in XML format reduces the need to focus on wrappers to make them interoperable, the challenges of integrating distributed heterogeneous structured data residing in relational databases and semi-structured data held in well-formed XML documents produced by internet applications still remains. Querying such heterogeneous distributed data sources is not easy for several reasons. The first difficulty comes from the distribution of the data. The second difficulty is associated with its heterogeneity, which occurs at different levels. The problem of the discrepancies between data sources is important. Usually, when the contents of data sources are related in some way, they will show diversity in many aspects. Resolving the differences between the data sources in these situations is a crucial issue. The logical heterogeneity is one of the more complicated issues that should be taken into consideration in building a data integration system. It comes from different understanding and modeling of the same concept. Thus, the construction of a data integration system must handle mechanisms for resolving conflicts when attributing meaning to the data (semantic conflicts), referencing data (naming conflicts), and storing data (structural conflicts). Hence, distinct data sources may use different names to refer to the same concept and may use the same name to refer to different concepts in these conflicts.

Since finding the correspondences between the schemas' elements often depend on the application context this is a basic issue. The matching of two elements requires a decision as to whether they correspond to each other in some way, i.e. are they logically equivalent? Therefore, any decision about the semantic correspondence of sets of elements requires an in-depth analysis by an integrator. In this area we use paths instead of elements, because the SSDs are trees and each element is identified uniquely by its path in the tree.

Consequently, as the base step in constructing the XMKB, we matched a set of paths of the schemas if they were related to each other in some way. To express a correspondence between a global path and a set of paths in a data source schema structure, we conducted an in-depth study of the semantics of the paths.

This work builds on the concept of a mediated system. The first contribution of the thesis is a mechanism for the mediation of heterogeneous distributed structured and semi-structured data sources. A mediation layer was introduced to maintain the mappings among global and local schemas. Such a layer was developed as an assistant tool to facilitate the detection, analysis and resolution of schema discrepancies and to improve the solution of relevant data integration issues. It can be used as an assisting tool to minimize the designer effort in building structured and semi-structured data integration systems. We argue that our approach can be used as a semi-automatic tool for mediation of heterogeneous distributed structured and semi-structured data sources.

Another difficulty which impedes data integration systems is the query translation process. This is an important problem in the design of a data integration system, in that the system should be able to reformulate the query posed in terms of the global schema into a set of queries suited to the data sources. Thus, the second contribution of the thesis was the provision of a mechanism that allows a user to transparently query structured and semi-structured data sources in a conceptual way (semantic name) instead of by knowledge of its local structure. This reduces the semantic problem for a user during query formulation, and significantly simplifies the task of querying multiple heterogeneous structured and semi-structured data sources. In this way, the system becomes responsible for translating global user queries into local queries. The thesis demonstrated an algorithm for the query translation process which was capable of generating a local query for each data source corresponding to part of the global query. During the process of generating local queries for the participating data sources, many structural and semantic conflicts are resolved by our system.

With regard to the mapping specification, there are two basic approaches that have been used to specify the mapping between the data sources and the global schema. These are the GAV and LAV approaches. Our approach is an attempt to combine features from both these approaches. The GAV approach requires that the global schema is expressed in terms of the data sources. This means, that for every element of the global schema, a view over the data sources is associated, which is specified in terms of data residing in the data sources. In other words, the global schema is defined as a view over the local data sources' schemas.

The LAV approach requires the global schema to be specified independently from the data sources. In turn, the data sources are defined as views over the global schema. Thus each data source is described in terms of the global schema elements. The LAV approach gives better support to a dynamic environment than GAV, where data sources can be added to the integration system without the need to restructure the global schema.

We classify our approach as a structural approach that can be used as a tool for structured and semi-structured data sources mediation and querying. It follows LAV in its way of describing the data sources, i.e. all the data sources' elements are mapped by mediation. In other words, it is not restricted to a subset of data sources involved, as is the case in GAV. Thus, the resulting LAV description is translated into GAV when generating the mappings between the global paths and local schemas' paths by the query translation process. Hence our approach combines the virtues of both GAV and LAV approaches. It follows the GAV approach in respect of query reformulation. This advantage facilitates the query translation task, in that it usually does nothing more than change and formula substitution. The biggest problem in the GAV approach is that it makes it complicated to implement changes in the global schema when there are changes in the schemas of the data sources. The LAV approach is better able to support a dynamic environment, where data sources can be added or removed from the integration system without the need to restructure the global schema.

As a final word, the benefit of our approach is that we can automate the process of construction of an XML Metadata Knowledge Base (XMKB) which can assist the Query Processor (QP) in querying a multiplicity of distributed heterogeneous structured data residing in relational databases and semi-structured data held in well-formed XML documents produced by internet applications or by human code. We have developed a prototype system to demonstrate that the ideas explored in the thesis are sound and practical, and convenient from a user standpoint. Our approach should be generic enough to easily incorporate a large number of relational databases and XML data sources from the same domain. We have shown our approach is feasible and is successful with real data in different domains and have shown that the approach is domain independent. This domain independency is one of the key points of our approach. A limitation of our approach is that it is not scalable with large schemas since they will involve considerable effort to do mappings by assigning a unique index number to each element and specifying conversion function names to resolve structural and semantic conflicts. However this is not a major limitation for our target domain as most of the data sources have small or medium schemas which are compatible with our approach.

CHAPTER 9

Summary, conclusion and future work

This chapter concludes the thesis by briefly summarizing the work, presenting the conclusions of the thesis, and addressing future directions for further development.

9.1 Thesis summary

We have presented an approach to integrate and query distributed heterogeneous structured data residing in relational databases and semi-structured data held in well-formed XML documents. A general overview of the field of distributed database systems was given with an overview of the types of heterogeneous distributed databases. The basic issues concerning data integration systems and their architectures were presented using a classification of the different aspects, concepts and approaches. After that we presented an overview of XML and its related technologies followed by a description of our approach to achieving a distributed system. Two important problems were addressed in this work. The first was establishing a Knowledge Base to hold descriptions of the mappings between the integrated view (master view) and the participating data sources which are used to resolve the logical heterogeneity present in the distributed local data sources' schemas. The second was the query translation process. These problems were concerned with building a structured and semi- structured data integration systems, in which a global schema was provided over the heterogonous data sources.

The integration architecture we adopted is based on a mediator architecture. The prototype system, called SISSD, performed mappings between the global schema and local data source schemas, by creating an XML Metadata Knowledge Base (XMKB), which is used to generate local queries. The data sources are described in XDSDL, a language created in the project. The mediation layer was developed to:

1. Establish appropriate mappings between the global schema and the schemas of the local data sources.
2. Enable querying of local data sources in terms of the global schema.

The challenge was to generate a mapping for the correspondence between schema elements. This was addressed by developing a methodology for extracting and formalizing element paths of the global and local schemas. A mapping process was then developed to generate the correspondences between paths. This was achieved through a semi-automatic process that generated local and global paths and their relationships. This created the XMKB module used in mediation to overcome heterogeneity problems between data sources. The XMKB module was used to hold the correspondence between schema paths. For each path of the global schema, the objective was to link it with the set of local paths that have the same meaning and with a user-defined, function if needed, to perform specific operations that are defined explicitly by the designer. These user-defined functions are used to overcome differences in representation and granularity.

The query translator, which is an integral part of the mediation layer, was developed to translate a user query posed over the global schema into local queries. It uses the mapping information defined in the XMKB, to obtain local queries corresponding to the query issued against the global schema. The basic idea was that a query posed to the integration system, called a global query, would be automatically rewritten to sub-queries called local queries, appropriate to each local data source's required format, using the information stored in XMKB. This task was accomplished by the query translator module. The XMKB contains the schema paths and functions to be applied when creating a query for a local data source. The paths in a global query are parsed by the query parser and replaced by the corresponding paths for each target local data source, by consulting the XMKB to see if there are such paths for the user query. If not, a null query is generated for the corresponding path in the local query, which means that this query cannot be applied to that local data source. Each local query generated is sent to its corresponding local data source, which executes the query and returns its result. The set of results are processed to get the full answer to the global query.

A simple prototype implementation of the system architecture was created using: Java 2, JDOM API, and the JavaCC. We also used FLWR expressions (For-Let-Where-Return) as the XML query language. This is a subset of XQuery which supports the basic requirements of our approach, particularly the uniform querying of heterogeneous distributed structured (relational database) and semi-structured (well-formed XML document) data sources.

9.2 Conclusions

This work has identified a new approach to structured and semi-structured data integration. We have addressed the logical heterogeneity problem which occurs between the schemas. This problem was solved by creating a mechanism in which the correspondence among schema elements is expressed as a set of mappings and by using UDFs to overcome conflicts where a transformation is required. This is described in section 2.4 and 4.4. These mappings are a powerful tool for expressing the correspondences between schema elements and capturing the heterogeneity of the various data sources. However, finding the correspondences between the schema elements will depend on the application context. Hence, matching two elements is a basic issue and requires a decision as to whether they correspond to each other in some way, e.g. are they logically equivalent. Any decision about the semantic correspondence of sets of element requires a deep analysis by a skilled integrator.

We have introduced an approach for heterogeneous structured and semi-structured data source mediation. This approach produced a system capable of processing queries across a set of heterogeneous distributed structured and semi-structured data sources. We developed a prototype system to demonstrate that the ideas explored in the thesis are sound and practical, and convenient from a user standpoint. The resulting system can easily incorporate a reasonable number of relational databases and XML data sources from the same domain. Most of the existing data integration systems in this area work with XML documents that use DTD (Document Type Definition) or XML Schema language to describe the schemas of the participating heterogeneous XML data sources in the data integration system. We have investigated and used XML documents which have no referenced DTD or XML schema, instead the schema metadata are buried inside the document data. However, XML documents which have a referenced DTD or XML schema can also be handling by bypassing the

DTD or the XML schema. This thesis has shown that querying a set of distributed heterogeneous structured and semi-structured data sources of this form is possible using our approach.

Thus, this work has developed a method of interoperation between structured and semi-structured data sources. This interoperation is achieved by generating mappings between global and local schemas, and resolving naming, structural and semantic conflicts which may occur between the schemas. Also we have developed a method for translating queries in terms of a global schema into sub-queries in terms of local schemas by exploiting the mapping information stored in the XMKB. The novelty of this research compared with the work done previously in this area and reviewed in chapter 2 is the use of a knowledge base approach and the use of UDFs to overcome naming, structural and semantic conflicts, also, the use of an incremental tool to build this knowledge base.

9.3 The future work

The work presented in this thesis can be extended in several ways. There are both practical and theoretical issues that need to be addressed to provide a complete framework for creating structured and semi-structured data integration systems. We suggest the following for future work:

- In data integration systems, a very important task is the integration of the results of the local queries. In our work, this task was not addressed other than at a basic level. For example, there may be duplicated information retrieved from the local data sources which should be removed when the results are presented.

- More features of Schema Structure Definition (SSD) can be involved in the process. For example, if some elements in the local data sources' SSD contain attributes and these attributes correspond to elements in the global schema a mapping between these elements would be needed. This is not yet implemented, but should not be a difficult extension to our current system.

- In this work, the global schema is specified by the integrator, or by choosing one of the data source's SSD that meet the requirements of the users to be a global schema. It should be possible to semi-automate the process of constructing the global unified schema that characterizes the underlying data sources.

- The major difficulty of connecting the global schema elements with the local schema elements when there are a large number of data sources, large size of schemas, and there is a high degree of logical heterogeneity between the schemas is the manual linkage. It should be possible to achieve scalability by generating mappings between the schemas elements automatically while reducing the manual integrator interaction to ensuring the semantic consistency of such mappings. However this needs further investigation.

Bibliography

[1] "Homepage. http://www.w3.org, 2001."

[2] "HyperText Markup Language Home Page. http://www.w3.org/MarkUp/, 2001."

[3] "W3C Consortium: XML Schema Part 0: Primer. http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/."

[4] "World Wide Web Consortium, http://www.w3.org/TR/2004/REC-xml-20040204/. Extensible Markup Language (XML) 1.0 W3C Recommendation, third edition, February 2004."

[5] "World Wide Web Consortium, http://www.w3.org/TR/xquery/. XQuery 1.0: An XML Query Language, W3C Working Draft, November 2003."

[6] "World Wide Web Consortium. Document object model (DOM) level 1 specification, version 1.0, w3c recommendation. See http://www.w3c.org/TR/1998/REC-DOM-Level-1-19981001 and see http://www.w3c.org/DOM/DOMTR., 1 October 1998."

[7] "XQL. http://www.w3.org/TandS/QL/QL98/PP/XQL.html."

[8] *International Organization for Standardization. ISO 8879: Information Processing -Text and Office Systems-Standard Generalized Markup Language (SGML)*, October 1986.

[9] S. Abiteboul, "Querying Semi-Structured Data," *Proceedings of the 6th International Conference on Database Theory, ICDT '97*, pp. 1-18, January 8-10, 1997.

[10] S. Abiteboul, P. Buneman, and D. Suciu, *Data on the Web: From Relational to Semistructured Data and XML.* San Francisco: Morgan Kaufmann, 2000.

[11] S. Abiteboul and O. M. Duschka, "Complexity of Answering Queries Using Materialized Views," in *Proceedings of the 17th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'98)*. Seattle, Washington, June 1998, pp. 254-263.

[12] R. Ahmed, P. D. Smedt, W. Du, W. Kent, M. A. Ketabchi, W. A. Litwin, A. Rafii, and M.-C. Shan, "The Pegasus Heterogeneous Multidatabase System," *IEEE Computer*, vol. 24(12), pp. 19-27, December 1991.

[13] M. B. Al-Mourad, W. A. Gray, and N. J. Fiddian, "Multiple Views with Multiple Behaviours for Interoperable Object-Oriented Database Systems," in *Proceedings of the 14th International Conference on Database and Expert Systems Applications, DEXA 2003*. Prague, Czech Republic, September 1-5, 2003, pp. 713-723.

[14] F. M. Al-Wasil, N. J. Fiddian, and W. A. Gray, "Query Translation for Distributed Heterogeneous Structured and Semi-structured Databases," in *Proceedings of the 23rd British National Conference on Databases (BNCOD2006)*. Belfast, Northern Ireland, 18 - 20 July, 2006.

[15] F. M. Al-Wasil and W. A. Gray, "Loosely-Coupled Linkage of Data from Structured and Semi-Structured Databases," in *Proceedings of the 6th International Conference on Information Integration and Web-based Applications Services (iiWAS2004)*. Jakarta, Indonesia: Austrian Computer Society, 27-29 September 2004.

[16] F. M. Al-Wasil, W. A. Gray, and N. J. Fiddian, "Establishing an XML Metadata Knowledge Base to Assist Integration of Structured and Semi-structured Databases," in *ADC '2006: Proceedings of the 17th Australasian Database Conference.* Tasmania, Australia, January 16-19, 2006.

[17] A. M. Alashqur, S. Su, and H. Lam, "OQL: A Query Language for Manipulating Object-oriented Databases," in *Proceedings of the 15th International Conference on Very Large Data Bases (VLDB)*. Amsterdam, The Netherlands, August 22-25, 1989.

[18] A. Almarimi and J. Pokorny, " A Mediation Layer for Heterogeneous XML Schemas," in *Proceedings of the 6th International Conference on Information Integration and Web Based Applications & Services (iiWAS2004)*. Jakarta, Indonesia: Austrian Computer Society, 27-29 September 2004.

[19] J. L. Ambite, N. Ashish, G. Barish, C. A. Knoblock, S. Minton, P. J. Modi, I. Muslea, A. Philpot, and S. Tejada, "ARIADNE: A System for Constructing Mediators for Internet Sources," in *Proceedings of ACM SIGMOD International Conference on Management of Data, SIGMOD98*. Seattle, Washington, USA, June 2-4, 1998, pp. 561-563.

[20] C. Baru, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, P. Velikhov, and V. Chu, "XML-based information mediation with MIX," in *SIGMOD '99: Proceedings of ACM SIGMOD International Conference on Management of Data*: ACM Press, 1999, pp. 597-599.

[21] C. Batini, M. Lenzerini, and S. B. Navathe, "A comparative analysis of methodologies for database schema integration," *ACM Computing Surveys*, vol. 18(4), pp. 323-364, 1986.

[22] A. Behm, A. Geppert, and K. R. Dittrich, "On the Migration of Relational Schemas and Data to Object-Oriented Database Systems," in *Proceedings of the 5th International Conference on Re-Technologies for Information Systems*. Klagenfurt, Austria, December 1997.

[23] D. Bell and J. Grimson, *Distributed Database systems*: Addison Wesley, 1992.

[24] L. Bellatreche, G. Pierra, D. N. Xuan, D. Hondjack, and Y. Ait-Ameur, "An a Priori Approach for Automatic Integration of Heterogeneous and Autonomous Databases," in *Proceedings of the15th International Conference on Database and Expert Systems Applications, DEXA 2004*. Zaragoza, Spain, August 30-September 3, 2004, pp. 475-485.

[25] S. Bergamaschi, S. Castano, and M. Vincini, "Semantic integration of semistructured and structured data sources," *ACM SIGMOD Record*, vol. 28(1), pp. 54-59, March 1999.

[26] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernandez, M. Kay, J. Robie, and J. Simeon., "XML path language (XPath) 2.0, W3C Working Draft. http://www.w3.org/TR/xpath20/." 2005.

[27] P. A. Bernstein and E. Rahm, "Data Warehouse Scenarios for Model Management," in *Proceedings of the 19th International Conference on Conceptual Modeling (ER 2000)*. Salt Lake City, Utah, USA: number 1920 in LNCS, Springer-Verlag, October 2000., pp. 1-15.

[28] J. M. Blanco, A. Illarramendi, and A. Goñi, "Building A Federated Relational Database System: An Approach Using A Knowledge-Based System," *International Journal of Intelligent and Cooperative Information Systems*, vol. 3(4), pp. 415-455, December 1994.

[29] S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, and J. Simeon, "XQuery 1.0: An XML query language, 2005. W3C Working Draft. http://www.w3.org/TR/xquery/."

[30] T. Bray, J. Paoli, and C. M. Sperberg-McQueen, "Extensible Markup Language (XML) 1.0 (2nd Edition)," W3C Recommendation, Oct. 2000. http://www.w3.org/TR/2000/REC-xml-20001006.

[31] A. Brown, "XML in serial publishing: past, present and future," *OCLC Systems & Services*, vol. 19(4), pp. 149-154, 2003.

[32] O. A. Bukhers, A. K. Elmagarmid, F. F. Gherfal, and X. Liu, *The Integration of Database Systems*: Prentice-Hall, 1996.

[33] D. Burnell, A. Al-Zobaidie, and G. Windall, "Bridging the gap between the data warehouse and XML," in *Proceedings of 14th International Workshop on Database and Expert Systems Applications (DEXA'03)*. Prague, Czech Republic, 1-5 Sept 2003, pp. 241- 246.

[34] S. Busse, R.-D. Kutsche, U. Leser, and H. Weber, "Federated Information Systems: Concepts, Terminology and Architectures," Technische Universität Berlin 1999.

[35] A. Calì, D. Calvanese, G. D. Giacomo, and M. Lenzerini, "On the Role of Integrity Constraints in Data Integration," *IEEE Data Engineering Bulletin*, vol. 25(3), pp. 39-45, 2002.

[36] D. Calvanese, G. D. Giacomo, M. Lenzerini, D. Nardi, and R. Rosati, "Data Integration in Data Warehousing," *International Journal of Cooperative Information Systems (IJCIS)*, vol. 10(3), pp. 237-271, 2001.

[37] M. J. Carey, D. Petkovic, J. Thomas, J. H. Williams, E. L. Wimmers, L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, M. Flickner, A. W. Luniewski, and W. Niblack, "Towards heterogeneous multimedia information systems: the Garlic approach," in *RIDE '95: Proceedings of the 5th International Workshop on Research Issues in Data Engineering-Distributed Object Management (RIDE-DOM'95)*: IEEE Computer Society, 1995, pp. 124-131.

[38] S. Ceri and G. Pelagatti, *Distributed databases principles and systems. Computer Science Series*: McGraw-Hill, Inc., 1984.

[39] D. Chamberlin, J. Robie, and D. Florescu, "Quilt: An XML Query Language for Heterogeneous Data Sources," in *Proceedings of International Workshop on the Web and Databases (WebDB).* Dallas, TX, USA, 2000, pp. 53–62.

[40] D. D. Chamberlin, "XQuery: An XML query language," *IBM Systems Journal*, vol. 41(4), pp. 597-615, 2002.

[41] S. Chaudhuri and U. Dayal, "An overview of data warehousing and OLAP technology," *ACM SIGMOD Record*, vol. 26(1), pp. 65-74, 1997.

[42] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom, "The TSIMMIS Project: Integration of Heterogeneous Information Sources," in *Proceedings of the 10th Anniversary Meeting of the Information Processing Society of Japan.* Tokyo, Japan, 1994, pp. 7-18.

[43] V. Christophides, S. Cluet, and J. Simèon, "On wrapping query languages and efficient XML integration," in *Proceedings of ACM SIGMOD Conference on Management of Data.* Dallas, Texas, USA, May 2000.

[44] J. Clark and S. DeRose, "XML Path Language (XPath), Version 1.0, W3C Recommendation, http://www.w3.org/TR/xpath," November 1999.

[45] E. F. Codd, *The Relational Model for Database Management: Version 2*: Addison-Wesley Longman Publishing Co., Inc., 1990.

[46] W. W. Cohen, "Integration of heterogeneous databases without common domains using queries based on textual similarity," in *Proceedings of the ACM SIGMOD international conference on Management of data, SIGMOD '98*. Seattle, Washington, USA, June 2-4, 1998, pp. 201-212.

[47] W. W. Cohen, "The WHIRL Approach to Information Integration," *IEEE Intelligent Systems*, pp. 20-23., Sept/Oct 1998.

[48] C. J. Date, *An Introduction to Database Systems*, 7th ed: Addison-Wesley, 2000.

[49] A. Deusch, M. Fernadez, D. Florscu, A. Levy, and D. Suciu, "XML-QL: A query language for XML. http://www.w3.org/tr/note-xml-ql. Technical report, April 1998."

[50] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu, "A Query Language for XML," in *Proceedings of the 8th International World Wide Web Conference (WWW8)*. Toronto, Canada, 1999.

[51] A. Deutsch, M. F. Fernandez, D. Florescu, A. Y. Levy, and D. Suciu, "XML-QL: A Query Language for XML," presented at WWW The Query Language Workshop (QL98), Cambridge, MA, 1998.

[52] A. Doan and R. McCann, "Building Data Integration Systems: A Mass Collaboration Approach," in *Proceedings of Information Integration on the Web (IIWeb-03)*. Acapulco, Mexico, August 9 - 10, 2003.

[53] R. Domenig, "A Query Based Approach for Integrating Heterogeneous Data Sources," *PhD thesis, Department of Information Technology, University of Zurich, Switzerland*, 2002.

[54] R. Domenig and K. R. Dittrich, "An Overview and Classification of Mediated Query System," *SIGMOD Record*, vol. 28(3), pp. 63-72, 1999.

[55] D. Draper, A. Y. HaLevy, and D. S. Weld, "The Nimble XML data integration system," in *Proceedings of the 17th International Conference on Data Engineering (ICDE'01)*, 2001, pp. 155-160.

[56] D. Draper, A. Y. Halevy, and D. S. Weld, "The nimble integration engine," in *Proceedings of the ACM SIGMOD international conference on Management of data (SIGMOD '01)*. Santa Barbara, California, United States, May 21 - 24, 2001, pp. 567-568.

[57] D. Dreilinger and A. E. Howe, "Experiences with selecting search engines using metasearch," *ACM Transactions on Information Systems (TOIS)*, vol. 15(3), pp. 195-222, 1997.

[58] O. M. Duschka and M. R. Genesereth, "Query planning in infomaster," in *Proceedings of the ACM symposium on Applied computing (SAC '97)*. San Jose, California, United States, 1997, pp. 109-111.

[59] R. M. Duwairi, "Views for Interoperability in a Heterogeneous Object-Oriented Multidatabase System," *PhD thesis, Department of Computer Science, University of Wales College of Cardiff*, April 1997.

[60] R. Eckstein and M. Casabianca, *XML Pocket Reference*, Second ed: O'Reilly & Associates, Inc., April 2001.

[61] A. Elmagarmid, M. Rusinkiewicz, and A. Sheth, *Management of Heterogeneous and Autonomous Database Systems*: Morgan Kufmann, 1999.

[62] A. K. Elmagarmid and C. Pu, "Guest Editors' Introduction to the Special Issue on Heterogeneous Databases," *ACM Computing Surveys*, vol. 22(3), pp. 175-178, September 1990.

[63] R. Elmasri and S. Navathe, *Fundamentals of Database Systems*, vol. 1, 3rd ed: Addison-Wesley, 2000.

[64] R. Elmasri and S. Navathe, *Fundamentals of Database Systems*, 3rd ed: Addison-Wesley, 2000.

[65] P. Fankhauser, W. Litwin, E. J. Neuhold, and M. Schrefl, "Global view definition and multidatabase languages - two approaches to database integration," in *Proceedings of the European Teleinformatics Conference (EUTECO 88)*. Vienna, Austria, April 1988, pp. 1069-1082.

[66] L. Feng, E. Chang, and T. Dillon, "A semantic network-based design methodology for XML documents," *ACM Transactions on Information Systems*, vol. 20(4), pp. 390–421, October 2002.

[67] D. Florescu, A. Levy, and A. Mendelzon, "Database techniques for the World-Wide Web: a survey," *ACM SIGMOD Record.*, vol. 27(3), pp. 59-74, 1998.

[68] J. E. Funderburk, G. Kiernan., J. Shanmugasundaram, E. Shekita, and C. Wei, "XTABLES: Bridging Relational Technology and XML," *IBM Systems Journal*, vol. 41(4), pp. 616-641, 2002.

[69] H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom, "Integrating and Accessing Heterogeneous Information Sources in TSIMMIS," in *Proceedings of the AAAI Symposium on Information Gathering*. Stanford, California, March 1995.

[70] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y.Sagiv, J. Ullman, V. Vassalos, and J. Widom, "The TSIMMIS Approach to Mediation: Data Models and Languages," *Journal of Intelligent Information Systems (JIIS)*, vol. 8(2), pp. 117-132, 1997.

[71] M. Garcia-Solaco, F. Saltor, and M. Castellanos, "Semantic heterogeneity in multidatabase systems," in *Object-Oriented Multidatabase Systems: A Solution for Advanced Applications*, O. A. Bukhres and A. K. Elmagarmid, Eds.: Prentice Hall International (UK) Ltd, 1995, pp. 129-202.

[72] G. Gardarin, A. Mensch, and A. Tomasic, "An Introduction to the e-XML Data Integration Suite," in *EDBT '02: Proceedings of the 8th International Conference on Extending Database Technology*: Springer-Verlag, 2002, pp. 297-306.

[73] G. Gardarin, F. Sha, and T. Dang-Ngoc, "XML-based Components for Federating Multiple Heterogeneous Data Sources," in *ER '99: Proceedings of the 18th International Conference on Conceptual Modeling*: Springer-Verlag, 1999, pp. 506-519.

[74] M. R. Genesereth, A. M. Keller, and O. M. Duschka, "Infomaster: An Information Integration System," in *Proceedings the ACM SIGMOD International Conference on Management of Data, SIGMOD97*. Tucson, Arizona, USA, May 13-15, 1997, pp. 539-542.

[75] C. H. Goh, S. Bressan, S. Madnick, and M. Siegel, "Context interchange: new features and formalisms for the intelligent integration of information," *ACM Transactions on Information Systems (TOIS)*, vol. 17(3), pp. 270-293, 1999.

[76] C. Goldfarb, *The SGML Handbook*: Clarendon Press, 1990.

[77] M. Goodchild, M. Egenhofer, R. Fegeas, and C. Kottman, "Interoperating Geographic Information Systems.," *Kluwer*, 1999.

[78] T. Grust, S. Sakr, and J. Teubner, "XQuery on SQL Hosts," in *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB 2004)*. Toronto, Canada, 29 August - 3 September 2004.

[79] A. Gupta, "Integration of Information Systems: Bridging Heterogeneous Databases," *IEEE Press*, 1989.

[80] A. Y. Halevy, "Answering queries using views: A survey," *The VLDB Journal The International Journal on Very Large Data Bases*, vol. 10(4), pp. 270-294, 2001.

[81] M. Haller, B. Pröll, W. Retschitzegger, A. M. Tjoa, and R. R. Wagner, "Integrating Heterogeneous Tourism Information in TIScover - The MIRO-Web Approach," in *Proceedings of the International Conference on Information and Communication Technologies in Tourism (ENTER 2000)*. Barcelona, Spain, April 26-28, 2000.

[82] G. Hamilton, R. Cattell, and M. Fisher, *JDBC Database Access With Java: A Tutorial and Annotated Reference*, 2nd ed: Addison-Wesley Pub Co, September 1997.

[83] T. Härder, G. Sauter, and J. Thomas, "The Intrinsic Problems of Structural Heterogeneity and an Approach to Their Solution," *The VLDB Journal*, vol. 8(1), pp. 25-43, 1999.

[84] J. Heflin and J. Hendler, "Semantic Interoperability on the Web," in *Proceedings of Extreme Markup Languages 2000. Graphic Communications Association*, 2000, pp. 111-120.

[85] T. Hernandez and S. Kambhampati, "Integration of Biological Sources: Current Systems and Challenges Ahead," *ACM SIGMOD Record*, vol. 33(3), pp. 51-60, 2004.

[86] G. Hu and H. Fernandes, "Integration and querying of distributed databases," in *Proceedings of the IEEE International Conference on Information Reuse and Integration (IRI 2003)*. Las Vegas, NV, USA, October 27-29,2003, pp. 167-174.

[87] R. Hull, "Managing semantic heterogeneity in databases: a theoretical prospective," in *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, PODS '97*. Tucson, Arizona, United States, May 11 - 15, 1997, pp. 51-61.

[88] J. Hunter and B. McLaughlin, "Easy Java/XML integration with JDOM, Part 2: Use JDOM to create and mutate XML," http://www.javaworld.com/javaworld/jw-07-2000/jw-0728-jdom2.html, Technical report, July 2000.

[89] J. Hunter and B. McLaughlin, "Easy Java/XML integration with JDOM, Part 1: Learn about a new open source API for working with XML," http://www.javaworld.com/javaworld/jw-05-2000/jw-0518-jdom.html, Technical report, May 2000.

[90] A. Hurson, M. Bright, and S. Pakzad, *Multidatabase systems: an advanced solution for global information sharing*: IEEE Computer Society Press, 1994.

[91] Z. G. Ives, "Efficient query processing for data integration," PhD thesis, University of Washington, Seattle, 2002.

[92] V. Josifovski, P. Schwarz, L. Haas, and E. Lin, "Garlic: a new flavor of federated query processing for DB2," in *Proceedings of the ACM SIGMOD international conference on Management of data, (SIGMOD '02)*. Madison, Wisconsin, 2002, pp. 524-532.

[93] D. D. Karunaratna, W. A. Gray, and N. J. Fiddian, "Establishing a Knowledge Base to Assist Integration of Heterogeneous Databases," in *BNCOD 16: Proceedings of the 16th British National Conferenc on Databases*: Springer-Verlag, 1998, pp. 103-118.

[94] W. Kelley, S. Gala, W. Kim, T. Reyes, and B. Graham, "Schema architecture of the UniSQL/M multidatabase system," *Modern database systems: the object model, interoperability, and beyond*, pp. 621-648, 1995.

[95] W. Kim, "Introduction to part 2: technology for interoperating legacy databases," *Modern database systems: the object model, interoperability, and beyond*, pp. 515-520, 1995.

[96] W. Kim, I. Choi, S. Gala, and M. Scheevel, "On Resolving Schematic Heterogeneity in Multidatabase Systems," *Modern Database Systems*, pp. 512-550, 1995.

[97] W. Kim, I. Choi, S. Gala, and M. Scheevel, "On Resolving Schematic Heterogeneity in Multidatabase Systems," *Distributed and Parallel Databases*, vol. 1(3), pp. 251-279, July, 1993.

[98] W. Kim and J. Seo, "Classifying schematic and data heterogeneity in multidatabase systems," *IEEE Computer*, vol. 24(12), pp. 12-18, December, 1991.

[99] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava, "The Information Manifold," in *Proceedings of the AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments, pp. 85-91*. Stanford University, Stanford, CA, March 1995.

[100] L. Kurgan, W. Swiercz, and K. Cios, "Semantic Mapping of XML Tags using Inductive Machine Learning," in *Proceedings of the International Conference on Machine Learning and Applications - ICMLA '02*. Las Vegas, Nevada, USA, 2002.

[101] K. Lee, J. Min, and K. Park, "A Design and Implementation of XML-Based Mediation Framework (XMF) for Integration of Internet Information Resources," in *HICSS '02: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS'02)-Volume 7*: IEEE Computer Society, 2002, pp. 202-210.

[102] P. Lehti and P. Fankhauser, "XML data integration with OWL: Experiences & challenges," in *Proceedings of the International Symposium on Applications and the Internet (SAINT 2004)*. Tokyo, Japan, 2004, pp. 160-170.

[103] M. Lenzerini, "Data integration: a theoretical perspective," in *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. Madison, Wisconsin, 2002.

[104] A. Levy, "The Information Manifold Approach to Data Integration," *IEEE Intelligent Systems*, vol. 13, pp. 12-16, 1998.

[105] A. Levy, A. Mendelzon, Y. Sagiv, and D. Srivastava, "Answering queries using views," in *Proceedings of the 14th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. San Jose, CA, USA, 1995.

[106] A. Y. Levy, "Combining Artificial Intelligence and Databases for Data Integration," *In Special issue of LNAI: Artificial Intelligence Today; Recent Trends and Developments. Lecture Notes in Computer Science*, vol. 1600, pp. 249-268, 1999.

[107] A. Y. Levy, A. Rajaraman, and J. J. Ordille, "Querying Heterogeneous Information Sources Using Source Descriptions," in

Proceedings of the 22th International Conference on Very Large Data Bases, VLDB'96. Mumbai (Bombay), India, September 3-6, 1996, pp. 251-262.

[108] C. Li, R. Yerneni, V. Vassalos, H. Garcia-Molina, Y. Papakonstantinou, J. Ullman, and M. Valiveti, "Capability based mediation in TSIMMIS," in *Proceedings of the ACM SIGMOD international conference on Management of data (SIGMOD '98)*. Seattle, Washington, United States, 1998, pp. 564-566.

[109] R. Li, Z. Lu, W. Xiao, B. Li, and W. Wu, "Schema Mapping for Interoperability in XML-Based Multidatabase Systems," in *DEXA '03: Proceedings of the 14th International Workshop on Database and Expert Systems Applications*: IEEE Computer Society, 2003, pp. 235.

[110] W. Litwin and A. Abdellatif, "Multidatabase interoperability," *IEEE Computer*, vol. 19(12), pp. 10-18, 1986.

[111] W. Litwin, L. Mark, and N. Roussopoulos, "Interoperability of multiple autonomous databases," *ACM Computing Surveys*, vol. 22(3), pp. 267-293, 1990.

[112] B. Ludäscher, A. Gupta, and M. E. Martone, "Model-Based Mediation with Domain Maps," in *Proceedings of the 17th International Conference on Data Engineering (ICDE)*. Heidelberg, Germany, April 2-6, 2001, pp. 81-90.

[113] I. Manolescu, D. Florescu, and D. Kossmann, "Answering XML Queries over Heterogeneous Data Sources," in *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB)*. Rome, Italy, September 2001.

[114] I. Manolescu, D. Florescu, D. Kossmann, F. Xhumari, and D. Olteanu, "Agora: Living with XML and Relational," in *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB00)*. Cairo, Egypt, September 10-14, 2000, pp. 623-626.

[115] W. May, "A Rule-Based Querying and Updating Language for XML," in *Proceedings of the 8th International Workshop on Database Programming Languages (DBPL '01)*. Frascati, Italy, September 8-10, 2001, pp. 165-181.

[116] D. Megginson, "SAX 2.0: The Simple API for XML," available at http://www.megginson.com/SAX/index.html, October 2000.

[117] R. J. Miller, L. M. Haas, and M. A. Hernández, "Schema Mapping as Query Discovery," in *Proceedings of the 26th International Conference on Very Large Data Bases (VLDB '00)*. Cairo, Egypt, September 10-14, 2000, pp. 77-88.

[118]R. J. Miller, M. A. Hernández, L. M. Haas, L. Yan, C. T. Howard, R. Fagin, and L. Popa, "The Clio project: managing heterogeneity," *ACM SIGMOD Record*, vol. 30(1), pp. 78-83, March 2001.

[119] T. Millstein, A. Levy, and M. Friedman, "Query containment for data integration systems," in *PODS '00: Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. Dallas, Texas, United States: ACM Press, 2000, pp. 67-75.

[120] Y.-K. Nam, J. Goguen, and G. Wang, "A Metadata Tool for Retrieval from Heterogeneous Distributed XML Documents," in *Proceedings of the International Conference on Computational Science, LNCS 2660, Springer, pp. 1020-1029.*, 2003.

[121] M. T. Ozsu and P. Valduriez, "Distributed database systems: Where are we now?" *IEEE Computer*, vol. 24(8), pp. 68-78, August 1991.

[122] M. T. Özsu and P. Valduriez, *Principles of Distributed Database Systems*, 2nd ed: Prentice Hall, San Ysidro, CA, 1999.

[123] Y. Papakonstantinou, H. Garcia-Molina, and J. D. Ullman, "MedMaker: A Mediation System Based on Declarative Specifications," in *ICDE '96: Proceedings of the 12th International Conference on Data Engineering*: IEEE Computer Society, 1996, pp. 132-141.

[124] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom, "Object Exchange Across Heterogeneous Information Sources," in *Proceedings of the 11th International Conference on Data Engineering (ICDE '95)*. Taipei, Taiwan, March 06 - 10, 1995, pp. 251-260.

[125] Y. Papakonstantinou and P. Velikhov, "Enhancing Semistructured Data Mediators with Document Type Definitions," in *Proceeding of Data Engineering (ICDE)*. Syndey, Australia, 1999.

[126] S. Raghavan and H. Garcia-Molina, "Integrating diverse information management systems: A brief survey," *IEEE Data Engineering Bulletin*, vol. 24(4), pp. 44-52, 2001.

[127] J. Robie, J. Lapp, and D. Schach., "XML query language (XQL)," *In QL'98 - The Query Languages Workshop*, 1998.

[128] F. Saltor, M. Castellanos, and M. García-Solaco, "Suitability of data models as canonical models for federated databases," *ACM SIGMOD Record*, vol. 20(4), pp. 44-48, December 1991.

[129] P. Schauble., *Multimedia Information Retrieval*: Kluwer Academic Publishers, 1997.

[130] A. Segev and A. Chatterjee, "Data manipulation in heterogeneous databases," *Sigmod Record*, vol. 20(4), pp. 64-68, December 1991.

[131] E. Selberg and O. Etzioni, "Multi-Service Search and Comparison Using the MetaCrawler," in *Proceedings of the 4th International World-Wide Web Conference*. Boston, Massachusetts, USA, December 11-14, 1995.

[132] E. Selberg and O. Etzioni, "The MetaCrawler architecture for resource aggregation on the Web," *IEEE Expert*, January-February 1997.

[133] J. Shanmugasundaram, E. J. Shekita, R. Barr, M. J. Carey, B. G. Lindsay, H. Pirahesh, and B. Reinwald, "Efficiently Publishing Relational Data as XML Documents," in *Proceedings of the 26th International Conference on Very Large Databases, (VLDB2000)*. Cairo, Egypt, September 2000, pp. 65-76.

[134] A. Sheth, "Semantic issues in Multidatabase Systems," *SIGMOD Record*, vol. 20(4), December 1991.

[135] A. P. Sheth and V. Kashyap, "So Far (Schematically) yet So Near (Semantically)," *Proceedings of the IFIP WG 2.6 Database Semantics Conference on Interoperable Database Systems (DS-5)*, pp. 283-312, November 16 - 20, 1992.

[136] A. P. Sheth and J. A. Larson, "Federated database systems for managing distributed, heterogeneous, and autonomous databases," *ACM Computing Surveys*, vol. 22(3), pp. 183-236, 1990.

[137] S. Spaccapietra and C. Parent, "Conflicts and correspondence assertions in interoperable databases," *ACM SIGMOD Record*, vol. 20(4), pp. 49-54, December 1991.

[138] V. S. Subrahmanian, S. Adali, A. Brink, R. Emery, J. J. Lu, A. Rajput, T. J. Rogers, R. Ross, and C. Ward, "HERMES: A heterogeneous Reasoning and Mediator System," *ARPA*, 1995.

[139] R. Sudha and P. Jinsoo, "Semantic conflict resolution ontology (SCROL): an ontology for detecting and resolving data and schema-level semantic conflicts," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 16, pp. 189, 2004.

[140] G. Thomas, G. R. Thompson, C.-W. Chung, E. Barkmeyer, F. Carter, M. Templeton, S. Fox, and B. Hartman, "Heterogeneous distributed database systems for production use," *ACM Computing Surveys*, vol. 22(3), pp. 237-266, 1990.

[141] A. Tomasic, L. Raschid, and P. Valduriez, "Scaling access to heterogeneous data sources with DISCO," *IEEE Transactions on Knowledge and Data Engineering*, vol. 10(5), pp. 808-823, 1998.

[142] S. White, M. Fisher, R. Cattell, G. Hamilton, and M. Hapner, *JDBC(TM) API Tutorial and Reference: Universal Data Access for the Java(TM) 2 Platform*, 2nd ed: Addison-Wesley Pub Co, 1999.

[143] J. Widom, "Integrating heterogeneous databases: lazy or eager?" *ACM Computing Surveys.*, vol. 28(4), pp. 91, 1996.

[144] J. Widom, "Research Problems in Data Warehousing," in *Proceedings of the 4th International Conference on Information and Knowledge Management*. Baltimore, Maryland, November 1995, pp. 25-30.

[145] G. Wiederhold, "Mediators in the Architecture of Future Information System," *IEEE Computer*, vol. 25(3), pp. 38-49, March 1992.

[146] L. Wood, "Programming the Web: the W3C DOM specification," *IEEE Internet Computing*, vol. 3(1), pp. 48-54, Jan/Feb 1999.

[147] L. Xu and D. W. Embley, "Combining the Best of Global-as-View and Local-as-View for Data Integration.," in *Information Systems Technology and its Applications, 3rd International Conference ISTA'2004*. Salt Lake City, Utah, USA, June 15-17, 2004.

[148] N. Young-Kwang, G. Joseph, and W. Guilian, "A Metadata Integration Assistant Generator for Heterogeneous Distributed Databases," in *Proceedings of the Confederated International*

Conferences DOA, CoopIS and ODBASE. Irvine CA: LNCS 2519, Springer, pp. 1332-1344., October 2002.

[149] C. Yu and L. Popa, "Constraint-based XML query rewriting for data integration," in *Proceedings of the ACM SIGMOD international conference on Management of data (SIGMOD '04)*. Paris, France, June 13 - 18, 2004, pp. 371-382.

[150] G. Zhou, R. Hull, R. King, and J.-C. Franchitti, "Data Integration and Warehousing Using H2O," *Bulletin of the Technical Committee on Data Engineering*, vol. 18(2), pp. 29-40, 1995.

APPENDIX A

Java code for the Main Interface of SISSD system

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

class MainInterFaceF extends JFrame implements ActionListener
{
private final int ITEM_PLAIN = 0;
private final int ITEM_CHECK = 1;
private final int ITEM_RADIO = 2;
public String integratedfile;
public String kbName="";
public Vector MappingPaths= new Vector();
public Vector SourceMetadata = new Vector();
public JPanel topPanel;
public JMenuBar menuBar;
public JMenu menuExtractor;
public JMenu menuQuery;
public JMenu menuKServer;
public JMenuItem menuItemRel;
public JMenuItem menuItemXML;
public JMenuItem menuItemIntegrated;
public JMenuItem menuItemlocal;
public JMenuItem menuItemMappings;
public JMenuItem menuItemKB,menuItemremove;
public JMenuItem menuItemQuery;
public JMenu submenu;
CreateXmlView listDialog;
JXC listDialog1;
JPanel mainPanel = new JPanel();
XYLayout xYLayout1 = new XYLayout();
JSplitPane hsplitPane = new JSplitPane();
JSplitPane vsplitPane = new JSplitPane();
public MainInterFaceF()
```

```
{
setTitle("User Interface");
setSize(1100, 1000);
topPanel = new JPanel();
topPanel.setLayout(new BorderLayout());
getContentPane().add(topPanel);
menuBar = new JMenuBar();
setJMenuBar(menuBar);
menuExtractor = new JMenu("    MetaData Extractor  ");
menuExtractor.setMnemonic('M');
menuBar.add(menuExtractor);
menuItemRel = CreateMenuItem(menuExtractor, ITEM_PLAIN,
"Bulid Schema Structure for Relational Database ", null, 'R',"");
menuExtractor.addSeparator();
menuItemXML = CreateMenuItem(menuExtractor, ITEM_PLAIN,
  "Bulid Schema Structure for XML Document",null, 'X', "");
menuKServer = new JMenu("    Knowledge Server  ");
menuKServer.setMnemonic('K');
menuBar.add(menuKServer);
submenu = new JMenu("Add New Data Source");
menuKServer.add(submenu);
menuItemIntegrated = CreateMenuItem(submenu, ITEM_PLAIN,"Step 1.
Generate index number for integrated  schema elements ", null,'G',"");
submenu.addSeparator();
menuItemlocal = CreateMenuItem(submenu, ITEM_PLAIN, "Step 2. Produce GUI
tree for local schema structure", null, 'P', "");
submenu.addSeparator();
menuItemMappings = CreateMenuItem(submenu, ITEM_PLAIN,
        "Step 3. Generate Path Mappings", null, 'H',"");
submenu.addSeparator();
menuItemKB = CreateMenuItem(submenu, ITEM_PLAIN,
           "Step 4. Merge Path Mappings with KB", null, 'M',"");
menuItemKB.setEnabled(true);
menuItemlocal.setEnabled(false);
menuItemMappings.setEnabled(false);
menuItemKB.setEnabled(false);
menuKServer.addSeparator();
menuItemremove = CreateMenuItem(menuKServer, ITEM_PLAIN,
            "Remove Data Source", null, 'V', "");
menuQuery = new JMenu("    Query Processor  ");
menuKServer.setMnemonic('Q');
menuBar.add(menuQuery);
menuItemQuery = CreateMenuItem(menuQuery, ITEM_PLAIN,
              "Process User Query ", null, 'U', "");
listDialog = new CreateXmlView();
listDialog1 = new JXC();
try
{ jbInit(); }
catch (Exception e)
{ e.printStackTrace();
}}
public JMenuItem CreateMenuItem(JMenu menu, int iType, String sText,
           ImageIcon image, int acceleratorKey, String sToolTip)
{ JMenuItem menuItem;
switch (iType)
{ case ITEM_RADIO:
menuItem = new JRadioButtonMenuItem();
```

```
break;
case ITEM_CHECK:
menuItem = new JCheckBoxMenuItem();
break;
default:
menuItem = new JMenuItem();
break;
}
menuItem.setText(sText);
if (image != null)
{ menuItem.setIcon(image); }
if (acceleratorKey > 0)
{ menuItem.setMnemonic(acceleratorKey); }
if (sToolTip != null)
{ menuItem.setToolTipText(sToolTip); }
menuItem.addActionListener(this);
menu.add(menuItem);
return menuItem;
}
public void actionPerformed(ActionEvent event)
{ if (event.getSource() == menuItemRel)
{ listDialog.setVisible(true); }
else if (event.getSource() == menuItemXML)
{ listDialog1.setVisible(true); }
else if (event.getSource() == menuItemIntegrated)
{ MappingPaths= new Vector();
final JFileChooser vc = new JFileChooser();
int returnVal = vc.showOpenDialog(this);
if (returnVal == JFileChooser.APPROVE_OPTION)
{ File file1 = vc.getSelectedFile();
integratedfile = file1.getAbsolutePath();
int ln=integratedfile.length();
kbName=integratedfile.substring(0,ln-4)+"_kb.xml";
JPanel rightPanel1 = new JPanel();
hsplitPane.setBottomComponent(rightPanel1);
hsplitPane.setDividerLocation(350);
JPanel rightPanel2 = new JPanel();
vsplitPane.setRightComponent(rightPanel2);
vsplitPane.setDividerLocation(200);
JPanel leftPanel = new GlobalSchemaPanel(integratedfile,this);
vsplitPane.setLeftComponent(leftPanel);
vsplitPane.setDividerLocation(500);
JOptionPane.showMessageDialog(this, "Index Number for Integrated Schema
Elements Generate Successfully");
}}
else if (event.getSource() == menuItemlocal)
{ final JFileChooser fc = new JFileChooser();
int returnVal = fc.showOpenDialog(this);
if (returnVal == JFileChooser.APPROVE_OPTION)
{ File file = fc.getSelectedFile();
String myfilename = file.getAbsolutePath();
JPanel rightPanel1 = new JPanel();
hsplitPane.setBottomComponent(rightPanel1);
hsplitPane.setDividerLocation(350);
JPanel rightPanel = new SourceSchemaPanel(myfilename,this);
vsplitPane.setRightComponent(rightPanel);
vsplitPane.setDividerLocation(200);
rightPanel.setBackground(Color.white);
}}
else if (event.getSource() == menuItemMappings)
{ JPanel rightPanel = new MappingPanel(this);
hsplitPane.setBottomComponent(rightPanel);
hsplitPane.setDividerLocation(350); }
else if (event.getSource() == menuItemKB)
{ KBmerge kb = new KBmerge();
kb.mergeMapping(this, kbName);
JOptionPane.showMessageDialog(this, "Path Mappings Merged Successfully
with The Knowledge Base");
menuItemMappings.setEnabled(false);
menuItemKB.setEnabled(false);
}
else if (event.getSource() == menuItemremove)
{ final JFileChooser vc = new JFileChooser();
int returnVal = vc.showOpenDialog(this);
if (returnVal == JFileChooser.APPROVE_OPTION)
{ File file1 = vc.getSelectedFile();
integratedfile = file1.getAbsolutePath();
JDialog a = new RemoveSources(integratedfile);
a.show();
}}
else if (event.getSource() == menuItemQuery)
{ QueryProcessor application = new QueryProcessor();
application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
application.show();
}}
public static void main(String args[])
{ MainInterFaceF a = new MainInterFaceF();
a.addWindowListener(new WindowAdapter()
{ public void windowClosing(WindowEvent e)
{ System.exit(0);
}});
a.setSize(1250, 1000);
a.setVisible(true);
a.show();
}
private void jbInit() throws Exception
{ mainPanel.setLayout(xYLayout1);
JPanel panel = new JPanel();
panel.setBackground(Color.white);
panel.setLayout( new BorderLayout() );
hsplitPane.setOrientation(JSplitPane.VERTICAL_SPLIT);
vsplitPane.setDividerSize(10);
vsplitPane.setLeftComponent(panel);
vsplitPane.setRightComponent(panel);
vsplitPane.setContinuousLayout(true);
vsplitPane.setBackground(Color.white);
this.getContentPane().add(mainPanel, BorderLayout.CENTER);
hsplitPane.setDividerSize(10);
hsplitPane.setTopComponent(vsplitPane);
hsplitPane.setBottomComponent(panel);
hsplitPane.setContinuousLayout(true);
mainPanel.add(hsplitPane, new XYConstraints(0, 0, 1250, 1000));
hsplitPane.setDividerLocation(330);
}}
```

APPENDIX B

Java code for extracting and building SSD for relational database

```
import javax.swing.JFrame;
import java.awt.Dimension;
import javax.swing.JLabel;
import java.awt.Rectangle;
import java.awt.Font;
import javax.swing.JTextField;
import javax.swing.JPanel;
import java.awt.GridLayout;
import java.awt.*;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.io.IOException;
import java.awt.Container;
import java.sql.*;
import java.awt.event.*;
import java.sql.*;
import java.io.*;

class CreateXmlView extends JFrame implements ActionListener
{
static BufferedWriter t;
String ch, chk;
private JLabel name,pass,status,dbname,xmlfile;
private JTextField user,stat,textdb,textfile;
private JPasswordField passbox;
private JButton connect,clear,cancel;
private JPanel pane,cent,input,connectx;
Color c1 = new Color(204,125,205);
Color c2 = new Color(108,153,204);
public CreateXmlView()
{
```

```
super("CONNECTION TO RELATIONAL DATABASE ");
int inset = 299;
Dimension scr =Toolkit.getDefaultToolkit().getScreenSize();
setBounds(inset,inset,scr.width-inset*2,scr.height-inset*2);
name = new JLabel("Username");
pass = new JLabel("Password");
dbname = new JLabel("Database Name");
xmlfile= new JLabel("Save Schema Structure In File Name");
connect = new JButton("Connect");
connect.addActionListener(this);
clear = new JButton("Clear");
clear.addActionListener(this);
cancel = new JButton("Cancel");
cancel.addActionListener(this);
cent = new JPanel();
input = new JPanel();
connectx =new JPanel();
textfile = new JTextField(10);
textdb = new JTextField(10);
user = new JTextField(10);
passbox = new JPasswordField(10);
cent.setLayout(new BorderLayout());
cent.add(input,"Center");
cent.add(connectx,"South");
input.setLayout(new GridLayout(4,4,5,5));
connectx.setLayout(new GridLayout(1,2,3,3));
input.add(xmlfile);
input.add(textfile);
input.add(dbname);
input.add(textdb);
input.add(name);
input.add(user);
input.add(pass);
input.add(passbox);
connectx.add(connect);
connectx.add(clear);
connectx.add(cancel);
setContentPane(cent);
}
public void actionPerformed( ActionEvent w )
{
Connection conn =null;
if (w.getSource() == connect )
{
String filename,filename1;
filename1=textfile.getText();
filename= "C:\\prototype\\schema_structure\\"+textfile.getText()+".xml";
                  File db=new File(filename);
if (db.exists())
{
JOptionPane.showMessageDialog(null,"The file "+filename1+ ".xml already
exists ","Error Message", JOptionPane.ERROR_MESSAGE);
textfile.setText("");
}
else
{ ch=user.getText();
char [] a = passbox.getPassword();
```

```
chk =String.valueOf(a);
String schema;
schema=textdb.getText().toUpperCase();
try
{ t = new BufferedWriter(new FileWriter(filename)); }
catch(Exception e)
{ System.out.println(e); }
try
{ Class.forName ("oracle.jdbc.driver.OracleDriver");
System.out.println("Driver loaded");
}
catch(Exception exe)
{
JOptionPane.showMessageDialog(null,"Driver error","Error Message",
JOptionPane.ERROR_MESSAGE);
}
if ( filename1 .equals (""))
{
JOptionPane.showMessageDialog(null,"Please enter file name of schema
structure ","Error Message", JOptionPane.ERROR_MESSAGE);
File file = new File(filename);
try
{ t.close(); }
catch(Exception excp)
{ System.out.println("File cannot be closed!"); }
boolean success = file.delete();
if (!success)
{ System.out.println("File cannot be deleted!");
}}
else
{ try
{
conn =DriverManager.getConnection("jdbc:oracle:thin:@helot:1521:oracle9
" , ch, chk);
System.out.println("Connection made");
ResultSet rset,rset3,rset4;
String tablename[]=new String[10];
DatabaseMetaData dbmd = conn.getMetaData();
rset3 = dbmd.getTables("",schema,"%",null);
int k=0,e;
String b=null;
while (rset3.next())
{ tablename[k]=rset3.getString(3);
k++; }
t.write("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");
t.write("<schema_information>");
t.write("<data_source_information>");
t.write("<name>"+schema+"</name>");
t.write("<location>jdbc:oracle:thin:@helot:1521:oracle9</location>");
t.write("<type>Relational Database</type>");
t.write("</data_source_information>");
t.write("<structure>");
t.write("<element name=\""+schema.toLowerCase()+"\">");
for (e=0;e<k;e++)
{ t.write("<element name=\""+tablename[e].toLowerCase()+"\">");
rset4=dbmd.getColumns("",schema,tablename[e],"%");
while (rset4.next())
{ t.write("<element name=\""+rset4.getString(4).toLowerCase()+"\"/>");
b=rset4.getString(3);
}
t.write("</element>");
}
t.write("</element>");
t.write("</structure>");
t.write("</schema_information>");
t.close();
if ( k == 0)
{ JOptionPane.showMessageDialog(null,"Invalid Database name","Error
Message", JOptionPane.ERROR_MESSAGE);
File file = new File(filename);
try
{ t.close(); }
catch(Exception excp)
{ System.out.println("File cannot be closed!"); }
boolean success = file.delete();
if (!success)
{ System.out.println("File cannot be deleted!");
}}
else
{ JOptionPane.showMessageDialog(this, "Schema Structure Built
Successfully for "+schema+" Database");
dispose();
user.setText("");
passbox.setText("");
textdb.setText("");
textfile.setText("");
}}
catch(Exception e)
{
JOptionPane.showMessageDialog(null,"Invalid Username or Password","Error
Message", JOptionPane.ERROR_MESSAGE);
File file = new File(filename);
try
{
t.close();
}
catch(Exception excp)
{
System.out.println("File cannot be closed!");
}
boolean success = file.delete();
if (!success)
{ System.out.println("File cannot be deleted!");
}}}}}
else if (w.getSource() == clear )
{ user.setText("");
passbox.setText("");
textdb.setText("");
textfile.setText(""); }
else
{ dispose();
}}}
```

APPENDIX C

Java code for extracting and building SSD for XML document

```
import javax.swing.JFrame;
import java.awt.Dimension;
import javax.swing.JLabel;
import java.awt.Rectangle;
import java.awt.Font;
import javax.swing.JTextField;
import javax.swing.JPanel;
import java.awt.GridLayout;
import java.awt.*;
import java.util.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.io.IOException;
import java.awt.Container;
import java.sql.*;
import java.awt.event.*;
import java.sql.*;
import org.jdom.*;
import org.jdom.input.SAXBuilder;
import java.io.IOException;
import java.util.*;
import java.io.*;

class JXC extends JFrame implements ActionListener
{
String ch, chk;
private JLabel doclocation,docname,xmlfile;
private JTextField textlocation,textdoc,textfile;
private JPasswordField passbox;
private JButton connect,clear,cancel;
private JPanel pane,cent,input,connectx;
```

```
Color c1 = new Color(204,125,205);
Color c2 = new Color(108,153,204);
static BufferedWriter out;
public JXC()
{
super("CONNECTION TO XML DOCUMENT");
//setDefaultCloseOperation(EXIT_ON_CLOSE);
int inset = 299;
Dimension scr =Toolkit.getDefaultToolkit().getScreenSize();
setBounds(inset,inset,scr.width-inset*2,scr.height-inset*2);
doclocation = new JLabel("XML Document Location");
docname = new JLabel("XML Document Name");
xmlfile= new JLabel("Save Schema Structure In File Name");
connect = new JButton("Connect");
connect.addActionListener(this);
clear = new JButton("Clear");
clear.addActionListener(this);
cancel = new JButton("Cancel");
cancel.addActionListener(this);
cent = new JPanel();
input = new JPanel();
connectx =new JPanel();
textlocation = new JTextField(10);
textdoc = new JTextField(10);
textfile = new JTextField(10);
cent.setLayout(new BorderLayout());
cent.add(input,"Center");
cent.add(connectx,"South");
input.setLayout(new GridLayout(4,4,5,5));
connectx.setLayout(new GridLayout(1,2,3,3));
input.add(xmlfile);
input.add(textfile);
input.add(doclocation);
input.add(textlocation);
input.add(docname);
input.add(textdoc);
connectx.add(connect);
connectx.add(clear);
connectx.add(cancel);
setContentPane(cent);
}
public void actionPerformed( ActionEvent w )
{
if (w.getSource() == connect )
{
String filename,filename1;
filename1 = textfile.getText();
filename = "C:\\prototype\\schema_structure\\"+textfile.getText()+
".xml";
File db=new File(filename);
if (db.exists())
{
JOptionPane.showMessageDialog(null,"The file "+filename1+ ".xml already
exists ","Error Message", JOptionPane.ERROR_MESSAGE);
textfile.setText("");
}
else
```

```
{
String location,documentname,documentname1;
location = textlocation.getText();
documentname=textdoc.getText();
documentname1=textdoc.getText();
if (!(documentname.endsWith(".xml")))
{
documentname = documentname + ".xml";
}
try
{ out = new BufferedWriter(new FileWriter(filename)); }
catch(Exception e)
{
System.out.println(e);
}
if ( filename1 .equals (""))
{
JOptionPane.showMessageDialog(null,"Please enter file name of schema
structure ","Error Message", JOptionPane.ERROR_MESSAGE);
File file = new File(filename);
try
{
out.close();
}
catch(Exception excp)
{ System.out.println("File cannot be closed!"); }
boolean success = file.delete();
if (!success)
{
System.out.println("File cannot be deleted!");
}}
else
{
SAXBuilder builder = new SAXBuilder();
try
{
out.write("<?xml version=\"1.0\" encoding=\"UTF-8\" ?>");
out.write("<schema_information>");
out.write("<data_source_information>");
out.write("<name>"+documentname + "</name>");
out.write("<location>"+location+"</location>");
out.write("<type>XML document</type>");
out.write("</data_source_information>");
out.write("<structure>");
Document doc = builder.build(location+"\\"+documentname);
Element root = doc.getRootElement();
listChildren(root, 0);
out.write("</structure>");
out.write("</schema_information>");
out.close();
JOptionPane.showMessageDialog(this, "Schema Structure Built Successfully
for "+documentname+" Document");
textfile.setText("");
textlocation.setText("");
textdoc.setText("");
dispose();
}
```

```
// indicates a well-formedness error
catch (JDOMException e)
{
JOptionPane.showMessageDialog(null,documentname +".xml is not well-
formed.","Error Message", JOptionPane.ERROR_MESSAGE);
textfile.setText("");
textlocation.setText("");
textdoc.setText("");
File file = new File(filename);
try
{ out.close(); }
catch(Exception excp)
{
System.out.println("File cannot be closed!");
}
System.out.println("my file path is: " + file.getAbsolutePath());
boolean success = file.delete();
if (!success)
{ System.out.println("File cannot be deleted!"); }
System.out.println(documentname + " is not well-formed.");
System.out.println(e.getMessage());
}
catch (IOException e)
{
System.out.println(e);
if (location .equals (""))
{
JOptionPane.showMessageDialog(null,"Please specify the XML document
location","Error Message", JOptionPane.ERROR_MESSAGE);
textfile.setText("");
textlocation.setText("");
textdoc.setText("");
File file = new File(filename);
try
{ out.close(); }
catch(Exception excp)
{ System.out.println("File cannot be closed!"); }
System.out.println("my file path is: " + file.getAbsolutePath());
boolean success = file.delete();
if (!success)
{
System.out.println("File cannot be deleted!");
}}
else if ( documentname .equals (""))
{
JOptionPane.showMessageDialog(null,"Please specify the XML document
name","Error Message", JOptionPane.ERROR_MESSAGE);
textfile.setText("");
textlocation.setText("");
textdoc.setText("");
File file = new File(filename);
try
{ out.close(); }
catch(Exception excp)
{
System.out.println("File cannot be closed!");
}
System.out.println("my file path is: " + file.getAbsolutePath());
boolean success = file.delete();
if (!success)
{
System.out.println("File cannot be deleted!");
}}
else
{
JOptionPane.showMessageDialog(null,"Please verify the XML document name
and location","Error Message", JOptionPane.ERROR_MESSAGE);
textfile.setText("");
textlocation.setText("");
textdoc.setText("");
File file = new File(filename);
try
{ out.close(); }
catch(Exception excp)
{
System.out.println("File cannot be closed!");
}
System.out.println("my file path is: " + file.getAbsolutePath());
boolean success = file.delete();
if (!success)
{ System.out.println("File cannot be deleted!");
}}}}}}
else if (w.getSource() == clear )
{ textfile.setText("");
textlocation.setText("");
textdoc.setText("");
}
else
{ dispose();
}}
public static void listChildren(Element current, int depth) throws
IOException
{ java.util.List children = current.getChildren();
Iterator iterator = children.iterator();
if (iterator.hasNext())
out.write("<element name=\""+current.getName()+"\">");
else out.write("<element name=\""+current.getName()+"\"/>");
String st="";
while (iterator.hasNext())
{ Element child = (Element) iterator.next();
if (!(child.getName().toString().equalsIgnoreCase(st)))
listChildren(child, depth+1);
st = child.getName();
if (!(iterator.hasNext()))
{ out.write("</element>");
}}}}
```

APPENDIX D

JDOM code for parsing master view to generate index numbers

```
import org.jdom.*;
import org.jdom.input.SAXBuilder;
import java.io.IOException;
import java.util.*;
public class GenerateIndex
{
static String x = "1";
static int y;
static int index = 1;
static int previousLevel = 0;
static String levels = "";
static String lastnode = "";
static String lastSign = "";
static TreeMap paths = new TreeMap();
static TreeMap elements = new TreeMap();
static Vector SourceMetadata = new Vector();
public GenerateIndex()
{
x = "1";
index = 1;
previousLevel = 0;
levels = "";
lastnode = "";
String lastSign = "";
paths = new TreeMap();
elements = new TreeMap();
}
public void GenerateIndex(String filename)
{
SAXBuilder builder = new SAXBuilder();
try
{
Element root1;
Document doc = builder.build(filename);
```

```
Element root = doc.getRootElement();
if (root.getName() .equals ("schema_information"))
{
Element information = root.getChild( "data_source_information" );
Element docname = information.getChild( "name" );
Element location = information.getChild( "location" );
Element type = information.getChild( "type" );
String con,con1,con2;
con = docname.getText();
con1 = location.getText();
con2 = type.getText();
SourceMetadata = new Vector();
SourceMetadata.add(con);
SourceMetadata.add(con1);
SourceMetadata.add(con2);
Element structure = root.getChild( "structure" );
root1 = structure.getChild("element");
}
else
{
root1 = root;
}
listChildren(root1, 1);
}
// indicates a well-formedness error
catch (JDOMException e)
{
System.out.println(" is not well-formed.");
System.out.println(e.getMessage());
}
catch (IOException e)
{
System.out.println(e);
}
}
public static void main(String[] args)
{
GenerateIndex gi = new GenerateIndex();
gi.GenerateIndex("bib_schema.xml");
Iterator e = paths.keySet().iterator();
while (e.hasNext())
{
String v = (String) e.next().toString();
String s = (String) paths.get(v);
}
}
public static void listChildren(Element current, int depth)
{
String previousPath = "";
String completePath = "";
String Space = "";
Space = getSpaces(depth);
printSpaces(depth);
String att = current.getAttributeValue("name");
if (depth == previousLevel)
{
levels = getTree(levels, depth);
```

```
int m = levels.lastIndexOf(".");
String a = levels.substring(m + 1);
levels = levels.substring(0, m + 1);
int o = Integer.parseInt(a);
o = o + 1;
levels = levels + o;
previousPath = getParents(levels);
completePath = previousPath  + "/"+ att;
paths.put(levels, completePath);
elements.put(levels, Space + att);
}
else if (depth > previousLevel)
{
if (levels.equalsIgnoreCase(""))
{
levels = levels + "1";
}
else
{
levels = levels + ".1";
}
previousPath = getParents(levels);
completePath = previousPath  + "/"+ att;
paths.put(levels, completePath);
elements.put(levels, Space + att);
previousLevel = depth;
}
else if (depth < previousLevel)
{
levels = getTree(levels, depth);
int m = levels.lastIndexOf(".");
String a = levels.substring(m + 1);
levels = levels.substring(0, m + 1);
int o = Integer.parseInt(a);
o = o + 1;
levels = levels + o;
previousLevel = depth;
previousPath = getParents(levels);
completePath = previousPath  + "/"+ att;
paths.put(levels, completePath);
elements.put(levels, Space + att);
}
List children = current.getChildren();
ListIterator iterator = children.listIterator();
while (iterator.hasNext())
{
Element child = (Element) iterator.next();
listChildren(child, depth + 1);
}
}
private static String getTree(String level, int depth)
{
int n = 0;
String s = "";
for (int i = 0; i < depth; i++)
{
n = level.indexOf(".", n + 1);
```

```
if (n == -1)
{ break;
}}
if (n == -1)
{ return level;
}
else
{ s = level.substring(0, n);
return s;
}}
private static void printSpaces(int n)
{
for (int i = 0; i < n; i++)
{
System.out.print("  ");
}}
private static String getSpaces(int n)
{
String space = "";
for (int i = 0; i < n; i++)
{
space = space + "  ";
}
return space;
}
private static String getParents(String indx)
{
String parentPath = "";
String previousIndex = "";
int p = indx.lastIndexOf(".");
if (p > -1)
{
previousIndex = indx.substring(0, p);
parentPath = (String) paths.get(previousIndex);
}
else
{ parentPath = "";
}
return parentPath;
}}
```

APPENDIX E

Java code for producing GUI and generating assistant tool for mapping

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import java.util.*;
import java.text.*;
import java.io.*;
import java.lang.*;
import java.net.*;
import javax.swing.filechooser.*;

public class GlobalSchemaPanel extends JPanel
{
BorderLayout borderLayout1 = new BorderLayout();
private JPanel pnl_txt = null;
JTextField txt_Filed = new JTextField();
JComboBox txt_Schm = new JComboBox();
JComboBox txt_Func = new JComboBox();
private JButton btn_ok = null;
private JButton btn_cancel = null;
private JLabel lbl_show = null;
private JLabel lbl_element = null;
private JLabel GlobalElement_lbl = null;
private JPanel JFrameContentPane = null;
public Vector IndexVector = new Vector();
private Vector textFieldsVector = new Vector();
private Vector FuncFieldsVector = new Vector();
private Vector labelsVector = new Vector();
public static String GlobalSchema = "";
public static String SourceSchema = "";
public static GenerateIndex gi;
public static GenerateIndex si;
public Vector SchemaElements = new Vector();
```

```
JComboBox CB = new JComboBox();
static TreeMap Globalpaths = new TreeMap();
static TreeMap Globalelements = new TreeMap();
public Frame fram;
private int y2 = 152, y3 = 12;
private int height = 5;
MainInterFaceF mycaller=null;
public GlobalSchemaPanel(String filename,MainInterFaceF caller)
{
GlobalSchema = filename;
mycaller=caller;
try
{
jbInit();
}
catch (Exception ex)
{
ex.printStackTrace();
}
}
void jbInit() throws Exception
SchemaElements.add("");
UIManager.put("Label.font", new Font("SansSerif", Font.BOLD, 12));
UIManager.put("Button.font", new Font("SansSerif", Font.BOLD, 12));
UIManager.put("TextField.font", new Font("SansSerif", Font.BOLD, 12));
UIManager.put("ComboBox.font", new Font("SansSerif", Font.PLAIN, 10));
UIManager.put("TextArea.font", new Font("SansSerif", Font.BOLD, 12));
try
{
this.setLayout(null);
JTextField textField = null;
JLabel Element_lbl = null;
gi = new GenerateIndex();
gi.GenerateIndex(GlobalSchema);
Iterator el = gi.elements.keySet().iterator();
int l = 0;
while (el.hasNext())
{
String v1 = (String) el.next().toString();
String s1 = (String) gi.elements.get(v1);
GlobalElement_lbl = getlbl_element(l, v1 + s1);
this.add(GlobalElement_lbl,
GlobalElement_lbl.getName());
l = l + 1;
SchemaElements.add(v1);
}
Globalpaths = (TreeMap) gi.paths;
Globalelements = (TreeMap) gi.elements;
}
catch (Throwable Exc)
{
handleException(Exc);
}
mycaller.menuItemlocal.setEnabled(true);
}
private JPanel getJFrameContentPane()
{
```

```
try
{
JFrameContentPane = new JPanel();
JFrameContentPane.setLayout(null);
JTextField textField = null;
JLabel Element_lbl = null;
gi = new GenerateIndex();
GlobalSchema = "schema_view1.xml";
gi.GenerateIndex(GlobalSchema);
Iterator e1 = gi.elements.keySet().iterator();
int l = 0;
while (e1.hasNext())
{
String v1 = (String) e1.next().toString();
String s1 = (String) gi.elements.get(v1);
GlobalElement_lbl = getlbl_element(l, v1 + s1);
JFrameContentPane.add(GlobalElement_lbl, GlobalElement_lbl.getName());
l = l + 1;
SchemaElements.add(v1);
}
Globalpaths = (TreeMap) gi.paths;
Globalelements = (TreeMap) gi.elements;
}
catch (Throwable Exc)
{
handleException(Exc);
}
return JFrameContentPane;
}
private JLabel getlbl_element(int i, String name)
{
try
{
lbl_element = new JLabel();
lbl_element.setName(name);
lbl_element.setText(name);
lbl_element.setBounds(50, 10 + (i * 5) * height, 150, 20);
}
catch (Throwable Exc)
{
handleException(Exc);
}
return lbl_element;
}
private void handleException(Throwable exception)
{
System.out.println("Could not initialize the frame. Error: " +
exception);
}
}

-----------------------------------------------------------------

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import java.util.*;
import java.text.*;
import java.io.*;
import java.lang.*;
import java.net.*;
import javax.swing.filechooser.*;

public class SourceSchemaPanel extends JPanel implements ActionListener
{
BorderLayout borderLayout1 = new BorderLayout();
private JPanel pnl_txt = null;
JTextField txt_Filed = new JTextField();
JTextField txt_fun = new JTextField();
JComboBox txt_Schm = new JComboBox();
JComboBox txt_Func = new JComboBox();
private Vector Functions = new Vector();
static TreeMap LocalFunctions = new TreeMap();
private JLabel lbl_head = null;
private JButton btn_ok = null;
private JButton btn_cancel = null;
private JButton btn_clear = null;
private JButton btn_add = null;
private JLabel lbl_show = null;
private JLabel lbl_element = null;
private JLabel GlobalElement_lbl = null;
private JPanel JFrameContentPane = null;
public Vector IndexVector = new Vector();
private Vector textFieldsVector = new Vector();
private Vector FuncFieldsVector = new Vector();
private Vector textFuncVector = new Vector();
private Vector AddFieldsVector = new Vector();
private Vector labelsVector = new Vector();
public static String GlobalSchema = "";
public static String SourceSchema = "";
public static GenerateIndex gi;
public static GenerateIndex si;
public Vector SchemaElements = new Vector();
JComboBox CB = new JComboBox();
static TreeMap Globalpaths = new TreeMap();
static TreeMap Globalelements = new TreeMap();
private Vector mappingPaths = new Vector();
private Vector sourceMetadata = new Vector();
private Mapping map;
public Frame fram;
private int y2 = 152, y3 = 12;
private int height = 5;
MainInterFaceF mycaller = null;
static TreeMap IntegratedPaths = new TreeMap();
public SourceSchemaPanel(String filename, MainInterFaceF caller)
{
Globalpaths = new TreeMap();
Globalelements = new TreeMap();
IntegratedPaths = new TreeMap();
SourceSchema = filename;
mycaller = caller;
mappingPaths= new Vector();
```

```
GlobalSchema=mycaller.integratedfile;
try
{
jbInit();
}
catch (Exception ex)
{
ex.printStackTrace();
}
}
void jbInit() throws Exception
{
SchemaElements.add("");
UIManager.put("Label.font", new Font("SansSerif", Font.BOLD, 12));
UIManager.put("Button.font", new Font("SansSerif", Font.BOLD, 12));
UIManager.put("TextField.font", new Font("SansSerif", Font.BOLD, 12));
UIManager.put("ComboBox.font", new Font("SansSerif", Font.PLAIN, 10));
UIManager.put("TextArea.font", new Font("SansSerif", Font.BOLD, 12));
try
{
this.setLayout(null);
JTextField textField = null;
JTextField textFunc = null;
JLabel Element_lbl = null;
JButton btn_add1 = null;
gi = new GenerateIndex();
gi.GenerateIndex(GlobalSchema);
Iterator e1 = gi.elements.keySet().iterator();
int l = 0;
while (e1.hasNext())
{
String v1 = (String) e1.next().toString();
String s1 = (String) gi.elements.get(v1);
GlobalElement_lbl = getlbl_element(l, v1 + s1);
l = l + 1;
SchemaElements.add(v1);
}
Globalpaths = (TreeMap) gi.paths;
Globalelements = (TreeMap) gi.elements;
si = new GenerateIndex();
sourceMetadata = new Vector();
si.GenerateIndex(SourceSchema);
sourceMetadata = (Vector) si.SourceMetadata.clone();
mycaller.SourceMetadata = (Vector) sourceMetadata;
Iterator e2 = si.elements.keySet().iterator();
Element_lbl = getlbl_heading1(0, "Data Source Name : ");
this.add(Element_lbl, Element_lbl.getName());
Element_lbl = getlbl_heading2(0, sourceMetadata.get(0).toString());
this.add(Element_lbl, Element_lbl.getName());
Element_lbl = getlbl_heading1(1, "Data Source Location : ");
this.add(Element_lbl, Element_lbl.getName());
Element_lbl = getlbl_heading2(1, sourceMetadata.get(1).toString());
this.add(Element_lbl, Element_lbl.getName());
Element_lbl = getlbl_heading1(2, "Data Source Type : ");
this.add(Element_lbl, Element_lbl.getName());
Element_lbl = getlbl_heading2(2, sourceMetadata.get(2).toString());
this.add(Element_lbl, Element_lbl.getName());
int n = 4;
while (e2.hasNext())
{
String v2 = (String) e2.next().toString();
String s2 = (String) si.elements.get(v2);
IndexVector.add(v2);
CB = gettxt_function(n);
textField = gettxt_Field(n);
Element_lbl = getlbl_show(n, s2);
this.add(Element_lbl, Element_lbl.getName());
this.add(textField, textField.getName());
this.add(CB, CB.getName());
textFunc = gettxt_multipleFunction(n);
this.add(textFunc, textField.getName());
btn_add1 = getbtn_add(n);
this.add(btn_add1, btn_add1.getName());
n = n + 1;
}
this.add(getbtn_cancel(n), getbtn_cancel(n).getName());
this.add(getbtn_ok(n), getbtn_ok(n).getName());
this.add(getbtn_clear(n), getbtn_clear(n).getName());
}
catch (Throwable Exc)
{
handleException(Exc);
}
}
private JPanel getJFrameContentPane()
{
try
{
JFrameContentPane = new JPanel();
JFrameContentPane.setLayout(null);
JTextField textField = null;
JLabel Element_lbl = null;
gi = new GenerateIndex();
GlobalSchema = "schema_view1.xml";
gi.GenerateIndex(GlobalSchema);
Iterator e1 = gi.elements.keySet().iterator();
int l = 0;
while (e1.hasNext())
{
String v1 = (String) e1.next().toString();
String s1 = (String) gi.elements.get(v1);
GlobalElement_lbl = getlbl_element(l, v1 + s1);
JFrameContentPane.add(GlobalElement_lbl,
GlobalElement_lbl.getName());
l = l + 1;
SchemaElements.add(v1);
}
Globalpaths = (TreeMap) gi.paths;
Globalelements = (TreeMap) gi.elements;
}
catch (Throwable Exc)
{
handleException(Exc);
}
```

```
return JFrameContentPane;
}
public void actionPerformed(ActionEvent e)
{
JButton add2 = null;
for (int i = 0; i < AddFieldsVector.size(); i++)
{
add2 = (JButton) AddFieldsVector.get(i);
if (e.getSource() == add2)
{
CB = (JComboBox) FuncFieldsVector.get(i);
txt_Filed = (JTextField) textFuncVector.get(i);
String last = txt_Filed.getText().toString();
String Sp = ",";
if (last.length() == 0)
{
Sp = "";
}
txt_Filed.setText(last + Sp + CB.getSelectedItem().toString());
}
}
if (e.getSource() == btn_cancel)
{
this.removeAll();
this.repaint();
return;
}
else if (e.getSource() == btn_ok)
{
mycaller.menuItemMappings.setEnabled(true);
btn_ok.setEnabled(false);
btn_cancel.setEnabled(false);
btn_clear.setEnabled(false);
generatePathMapping5();
JOptionPane.showMessageDialog(this,"Indexes Numbers Assigned
Successfully");
JTextField textField = null;
}
else if (e.getSource() == btn_clear)
{
for (int i = 0; i < textFieldsVector.size(); i++)
{
txt_Filed = (JTextField) textFieldsVector.get(i);
txt_Filed.setText("");
}
for (int i = 0; i < FuncFieldsVector.size(); i++)
{
CB = (JComboBox) FuncFieldsVector.get(i);
CB.setSelectedIndex(0);
}
}
}
private JLabel getlbl_element(int i, String name)
{
try
{
lbl_element = new JLabel();
lbl_element.setName(name);
lbl_element.setText(name);
lbl_element.setBounds(50, 10 + (i * 5) * height, 150, 20);
}
catch (Throwable Exc)
{
handleException(Exc);
}
return lbl_element;
}
private JButton getbtn_ok(int i)
{
if (btn_ok == null)
{
try
{
btn_ok = new JButton();
btn_ok.setName("btn_ok");
btn_ok.setText("Submit");
btn_ok.setBounds(400, 60 + (i * 5) * height, 85, 25);
btn_ok.addActionListener(this);
}
catch (Throwable Exc)
{
handleException(Exc);
}
}
return btn_ok;
}
private JTextField gettxt_Field(int i)
{
try
{
txt_Filed = new JTextField();
txt_Filed.setName("Field" + i);
txt_Filed.setEditable(true);
txt_Filed.setText("");
txt_Filed.setBounds(230, 10 + (i * 5) * height, 100, 20);
textFieldsVector.add(txt_Filed);
}
catch (Throwable Exc)
{
handleException(Exc);
}
return txt_Filed;
}
private JTextField gettxt_multipleFunction(int i)
{
try
{
txt_fun = new JTextField();
txt_fun.setName("FunField" + i);
txt_fun.setEditable(true);
txt_fun.setText("");
txt_fun.setBounds(560, 10 + (i * 5) * height, 200, 20);
textFuncVector.add(txt_fun);
}
```

```
catch (Throwable Exc)
{
handleException(Exc);
}
return txt_fun;
}
private JComboBox gettxt_SchemaElement(int i)
{
try
{
txt_Schm = new JComboBox(SchemaElements);
txt_Schm.setName("Schema_element" + i);
txt_Schm.setBounds(210, 10 + (i * 5) * height, 150, 15);
textFieldsVector.add(txt_Schm);
}
catch (Throwable Exc)
{
handleException(Exc);
}
return txt_Schm;
}
private JComboBox gettxt_function(int i)
{
String[] functions = {" ", "firstName", "lastName", "RateExchange", "is-
Part_of", "contains", "Merge"};
try
{
txt_Func = new JComboBox(functions);
txt_Func.setBackground(Color.white);
txt_Func.setName("function" + i);
txt_Func.setBounds(350, 10 + (i * 5) * height, 100, 20);
FuncFieldsVector.add(txt_Func);
}
catch (Throwable Exc)
{
handleException(Exc);
}
return txt_Func;
}
private JButton getbtn_cancel(int i)
{
if (btn_cancel == null)
{
try
{
btn_cancel = new JButton();
btn_cancel.setName("btn_cancel");
btn_cancel.setText("Cancel");
btn_cancel.setBounds(600, 60 + (i * 5) * height, 85, 25);
btn_cancel.addActionListener(this);
}
catch (Throwable Exc)
{
handleException(Exc);
}
}
return btn_cancel;
```

```
}
private JButton getbtn_clear(int i)
{
if (btn_clear == null)
{
try
{
btn_clear = new JButton();
btn_clear.setName("btn_clear");
btn_clear.setText("Clear");
btn_clear.setBounds(500, 60 + (i * 5) * height, 85, 25);
btn_clear.addActionListener(this);
}
catch (Throwable Exc)
{
handleException(Exc);
}
}
return btn_clear;
}
private JButton getbtn_add(int i)
{
try
{
btn_add = new JButton();
btn_add.setName("btn_add");
btn_add.setText("Add");
btn_add.setBounds(470, 10 + (i * 5) * height, 70, 20);
btn_add.addActionListener(this);
AddFieldsVector.add(btn_add);
}
catch (Throwable Exc)
{
handleException(Exc);
}
return btn_add;
}
private JLabel getlbl_show(int i, String name)
{
try
{
lbl_show = new JLabel();
lbl_show.setName(name);
lbl_show.setText(name);
lbl_show.setBounds(110, 10 + (i * 5) * height, 100, 20);
}
catch (Throwable Exc)
{
handleException(Exc);
}
return lbl_show;
}
private JLabel getlbl_heading2(int i, String name)
{
try
{
lbl_head = new JLabel();
```

```
lbl_head.setName(name);
lbl_head.setText(name);
lbl_head.setBounds(250, 10 + (i * 5) * height, 300, 20);
}
catch (Throwable Exc)
{
handleException(Exc);
}
return lbl_head;
}
private JLabel getlbl_heading1(int i, String name)
{
try
{
lbl_head = new JLabel();
lbl_head.setName(name);
lbl_head.setText(name);
lbl_head.setBounds(110, 10 + (i * 5) * height, 200, 20);
}
catch (Throwable Exc)
{
handleException(Exc);
}
return lbl_head;
}
private void handleException(Throwable exception)
{
System.out.println("Could not initialize the frame. Error:"+ exception);
}
private void generatePathMapping()
{
String indexKey = "";
String GlobalPath = "";
String SourcePath = "";
JTextField textField = null;
for (int i = 0; i < textFieldsVector.size(); i++)
{
textField = (JTextField) textFieldsVector.get(i);
map = new Mapping();
if (textField.getText().length() != 0)
{
CB = (JComboBox) FuncFieldsVector.get(i);
indexKey = (String) IndexVector.get(i);
String delimiters = ",";
String str;
str = textField.getText().toString();
StringTokenizer st = new StringTokenizer(str, delimiters);
SourcePath = (String) si.paths.get(indexKey);
map.SourcePath = SourcePath;
Vector pathsVector = null;
pathsVector = new Vector();
while (st.hasMoreTokens())
{
String index = st.nextToken();
indexKey = (String) IndexVector.get(i);
SourcePath = (String) si.paths.get(indexKey);
GlobalPath = (String) Globalpaths.get(index);
pathsVector.add(GlobalPath);
}
map.GlobalPaths = (Vector) pathsVector;
if ( (CB.getSelectedItem().toString().trim()).length() > 0)
{
map.FunctionName = CB.getSelectedItem().toString();
}
}
else
{
indexKey = (String) IndexVector.get(i);
SourcePath = (String) si.paths.get(indexKey);
map.SourcePath = SourcePath;
}
mappingPaths.add(map);
}
mycaller.MappingPaths = (Vector) mappingPaths.clone();
}
private void generatePathMapping2()
{
String indexKey = "";
String GlobalPath = "";
String SourcePath = "";
LocalFunction lf = new LocalFunction();
JTextField textField, txt_function = null;
Hashtable IntegratedPath = new Hashtable();
Vector localPaths = null;
Iterator el = Globalpaths.keySet().iterator();
while (el.hasNext())
{
String v1 = (String) el.next().toString();
String s1 = (String) Globalpaths.get(v1);
IntegratedPaths.put(v1, lf);
}
for (int i = 0; i < textFieldsVector.size(); i++)
{
textField = (JTextField) textFieldsVector.get(i);
map = new Mapping();
lf = new LocalFunction();
lf.LocalSourcePaths = null;
lf.FunctionName = null;
String myfunction = "";
if (textField.getText().length() != 0)
{
CB = (JComboBox) FuncFieldsVector.get(i);
indexKey = (String) IndexVector.get(i);
String delimiters = ",";
String str;
str = textField.getText().toString();
StringTokenizer st = new StringTokenizer(str, delimiters);
SourcePath = (String) si.paths.get(indexKey);
txt_function = (JTextField) textFuncVector.get(i);
String f = txt_function.getText().toString();
StringTokenizer stf = new StringTokenizer(f, delimiters);
Vector pathsVector = null;
Vector functionVector = new Vector();
pathsVector = new Vector();
```

```
localPaths = new Vector();
while (st.hasMoreTokens())
{
String index = st.nextToken();
if (stf.hasMoreElements())
{
myfunction = stf.nextToken();
}
indexKey = (String) IndexVector.get(i);
SourcePath = (String) si.paths.get(indexKey);
GlobalPath = (String) Globalpaths.get(index);
pathsVector.add(GlobalPath);
String Separator = ",";
localPaths.add(SourcePath);
txt_function = (JTextField) textFuncVector.get(i);
LocalFunctions.put(indexKey, myfunction);
functionVector.add(myfunction);
IntegratedPaths.put(index, lf);
map.SourcePath = index;
map.GlobalPaths = (Vector) localPaths;
map.FunctionName = "no function";
mappingPaths.add(map);
}
lf.LocalSourcePaths = (Vector) localPaths;
lf.FunctionName = (Vector) functionVector;
txt_function = (JTextField) textFuncVector.get(i);
Functions.add(txt_function.getText());
map.GlobalPaths = (Vector) pathsVector;
txt_function = (JTextField) textFuncVector.get(i);
Functions.add(txt_function.getText());
}
else
{
indexKey = (String) IndexVector.get(i);
SourcePath = (String) si.paths.get(indexKey);
}
}
mycaller.MappingPaths = (Vector) mappingPaths.clone();
Iterator e11 = IntegratedPaths.keySet().iterator();
while (e11.hasNext())
{
String v1 = (String) e11.next().toString();
String m1 = (String) Globalpaths.get(v1);
lf = (LocalFunction) IntegratedPaths.get(v1);
Vector j = new Vector();
Vector q = new Vector();
if (lf.LocalSourcePaths != null)
{
j = (Vector) lf.LocalSourcePaths;
q = (Vector) lf.FunctionName;
}
}
Iterator e111 = LocalFunctions.keySet().iterator();
while (e111.hasNext())
{
String v11 = (String) e111.next().toString();
String m11 = (String) si.paths.get(v11);
String s11 = (String) LocalFunctions.get(v11);
}
}
private void generatePathMapping5()
{
String indexKey = "";
String GlobalPath = "";
String SourcePath = "";
JTextField textField, txt_function = null;
Hashtable IntegratedPath = new Hashtable();
Vector localPaths = null;
Mapping mp=null;
Iterator e1 = Globalpaths.keySet().iterator();
while (e1.hasNext())
{
mp=new Mapping();
String v1 = (String) e1.next().toString();
String s1 = (String) Globalpaths.get(v1);
mp.SourcePath =s1;
mp.FunctionName="";
mp.GlobalPaths=null;
IntegratedPaths.put(v1, mp);
}
for (int i = 0; i < textFieldsVector.size(); i++)
{
mp=new Mapping();
textField = (JTextField) textFieldsVector.get(i);
String myfunction = "";
if (textField.getText().length() != 0)
{
indexKey = (String) IndexVector.get(i);
String delimiters = ",";
String str;
str = textField.getText().toString();
StringTokenizer st = new StringTokenizer(str, delimiters);
txt_function = (JTextField) textFuncVector.get(i);
String f = txt_function.getText().toString();
StringTokenizer stf = new StringTokenizer(f, delimiters);
Vector pathsVector = null;
Vector functionVector = new Vector();
pathsVector = new Vector();
localPaths = new Vector();
while (st.hasMoreTokens())
{
String index = st.nextToken();
if (stf.hasMoreElements())
{
myfunction = stf.nextToken();
}
indexKey = (String) IndexVector.get(i);
SourcePath = (String) si.paths.get(indexKey);
GlobalPath = (String) Globalpaths.get(index);
pathsVector.add(SourcePath);
String Separator = ",";
mp = (Mapping) IntegratedPaths.get(index);
Vector local = new Vector();
if (mp.GlobalPaths != null)
{
local = (Vector) mp.GlobalPaths;
local.add(SourcePath);
}
else
{
local.add(SourcePath);
}
mp.GlobalPaths=(Vector) local;
mp.FunctionName =myfunction ;
mp.SourcePath=GlobalPath;
IntegratedPaths.put(index, mp);
}
}
}
Iterator ell = IntegratedPaths.keySet().iterator();
mappingPaths = new Vector();
while (ell.hasNext())
{
String v1 = (String) ell.next().toString();
String m1 = (String) Globalpaths.get(v1);
mp = (Mapping) IntegratedPaths.get(v1);
mappingPaths.add(mp);
Vector j = new Vector();
if (mp.GlobalPaths != null)
{
j = (Vector) mp.GlobalPaths;
}
}
mycaller.MappingPaths = (Vector) mappingPaths.clone();
for (int i = 0; i < mappingPaths.size(); i++)
{
Vector sr= new Vector();
map = (Mapping) mappingPaths.get(i);
sr=(Vector) map.GlobalPaths;
}
}
private void generatePathMapping4()
{
String indexKey = "";
String GlobalPath = "";
String SourcePath = "";
LocalFunction lf = new LocalFunction();
JTextField textField, txt_function = null;
Hashtable IntegratedPath = new Hashtable();
Vector localPaths = null;
Iterator e1 = Globalpaths.keySet().iterator();
while (e1.hasNext())
{
String v1 = (String) e1.next().toString();
String s1 = (String) Globalpaths.get(v1);
IntegratedPaths.put(v1, lf);
}
for (int i = 0; i < textFieldsVector.size(); i++)
{
textField = (JTextField) textFieldsVector.get(i);
map = new Mapping();
```

```
lf = new LocalFunction();
lf.LocalSourcePaths = null;
lf.FunctionName = null;
String myfunction = "";
if (textField.getText().length() != 0)
{
indexKey = (String) IndexVector.get(i);
String delimiters = ",";
String str;
str = textField.getText().toString();
StringTokenizer st = new StringTokenizer(str, delimiters);
SourcePath = (String) si.paths.get(indexKey);
txt_function = (JTextField) textFuncVector.get(i);
String f = txt_function.getText().toString();
StringTokenizer stf = new StringTokenizer(f, delimiters);
Vector pathsVector = null;
Vector functionVector = new Vector();
pathsVector = new Vector();
localPaths = new Vector();
while (st.hasMoreTokens())
{
String index = st.nextToken();
if (stf.hasMoreElements())
{
myfunction = stf.nextToken();
}
indexKey = (String) IndexVector.get(i);
SourcePath = (String) si.paths.get(indexKey);
GlobalPath = (String) Globalpaths.get(index);
pathsVector.add(GlobalPath);
String Separator = ",";
localPaths.add(SourcePath);
txt_function = (JTextField) textFuncVector.get(i);
LocalFunctions.put(indexKey, myfunction);
functionVector.add(myfunction);
lf = (LocalFunction) IntegratedPaths.get(index);
Vector j = new Vector();
Vector q = new Vector();
if (lf.LocalSourcePaths != null)
{
j = (Vector) lf.LocalSourcePaths;
q = (Vector) lf.FunctionName;
}
j.add(SourcePath);
q.add(myfunction);
lf.LocalSourcePaths=(Vector) j;
lf.FunctionName = (Vector) q;
IntegratedPaths.put(index, lf);
map.SourcePath = index;
map.GlobalPaths = (Vector) localPaths;
map.FunctionName = "no function";
mappingPaths.add(map);
}
lf.LocalSourcePaths = (Vector) localPaths;
lf.FunctionName = (Vector) functionVector;
txt_function = (JTextField) textFuncVector.get(i);
Functions.add(txt_function.getText());
map.GlobalPaths = (Vector) pathsVector;
txt_function = (JTextField) textFuncVector.get(i);
Functions.add(txt_function.getText());
}
else
{
indexKey = (String) IndexVector.get(i);
SourcePath = (String) si.paths.get(indexKey);
}
}
mycaller.MappingPaths = (Vector) mappingPaths.clone();
Iterator el1 = IntegratedPaths.keySet().iterator();
while (el1.hasNext())
{
String v1 = (String) el1.next().toString();
String m1 = (String) Globalpaths.get(v1);
lf = (LocalFunction) IntegratedPaths.get(v1);
Vector j = new Vector();
Vector q = new Vector();
if (lf.LocalSourcePaths != null)
{
j = (Vector) lf.LocalSourcePaths;
q = (Vector) lf.FunctionName;
}
}
Iterator el11 = LocalFunctions.keySet().iterator();
while (el11.hasNext())
{
String v11 = (String) el11.next().toString();
String m11 = (String) si.paths.get(v11);
String s11 = (String) LocalFunctions.get(v11);
}
}
private void generatePathMapping1()
{
String indexKey = "";
String GlobalPath = "";
String SourcePath = "";
JTextField textField = null;
for (int i = 0; i < textFieldsVector.size(); i++)
{
CB = (JComboBox) textFieldsVector.get(i);
if (CB.getSelectedItem().toString() != "")
{
indexKey = (String) IndexVector.get(i);
SourcePath = (String) si.paths.get(indexKey);
GlobalPath = (String) Globalpaths.get(CB.getSelectedItem().toString());
}
}
}
}
```

APPENDIX F

Java code for paths mapping generation

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;
import java.util.*;
import java.text.*;
import java.io.*;
import java.lang.*;
import java.net.*;
import javax.swing.filechooser.*;

public class MappingPanel extends JPanel
{
BorderLayout borderLayout1 = new BorderLayout();
private JPanel pnl_txt = null;
JTextField txt_Filed = new JTextField();
JComboBox txt_Schm = new JComboBox();
JComboBox txt_Func = new JComboBox();
private JButton btn_ok = null;
private JButton btn_cancel = null;
private JLabel lbl_show = null;
private JLabel lbl_line = null;
private JLabel lbl_element = null;
private JLabel lbl_Gpath = null;
private JLabel lbl_function = null;
private JLabel line_lbl = null;
private JLabel GlobalElement_lbl = null;
private JPanel JFrameContentPane = null;
public Vector IndexVector = new Vector();
private Vector textFieldsVector = new Vector();
private Vector FuncFieldsVector = new Vector();
private Vector labelsVector = new Vector();
public static String GlobalSchema = "";
public static String SourceSchema = "";
public static GenerateIndex gi;
public static GenerateIndex si;
public Vector SchemaElements = new Vector();
JComboBox CB = new JComboBox();
static TreeMap Globalpaths = new TreeMap();
static TreeMap Globalelements = new TreeMap();
private Vector mappingPaths = new Vector();
private Mapping map;
public Frame fram;
private int y2 = 152, y3 = 12;
private int height = 5;
MainInterFaceF mycaller=null;
private String sourcePath;
private Vector globalPaths;
private String functionName;
public MappingPanel(MainInterFaceF caller)
{
mappingPaths = new Vector();
mycaller=caller;
mappingPaths=(Vector) mycaller.MappingPaths;
mycaller.menuItemKB.setEnabled(true);
try
{ jbInit(); }
catch (Exception ex)
{
ex.printStackTrace();
}}
void jbInit() throws Exception
{
SchemaElements.add("");
UIManager.put("Label.font", new Font("SansSerif", Font.BOLD, 12));
UIManager.put("Button.font", new Font("SansSerif", Font.BOLD, 12));
UIManager.put("TextField.font", new Font("SansSerif", Font.BOLD, 12));
UIManager.put("ComboBox.font", new Font("SansSerif", Font.PLAIN, 10));
UIManager.put("TextArea.font", new Font("SansSerif", Font.BOLD, 12));
try
{
this.setLayout(null);
JTextField textField = null;
JLabel Element_lbl = null;
Element_lbl = getlbl_Gpath(0,  "Data Source Element path");
this.add(Element_lbl, Element_lbl.getName());
Element_lbl = getlbl_element(0,"Master View Element path");
this.add(Element_lbl, Element_lbl.getName());
Element_lbl = getlbl_function(0, "Mapping Function");
this.add(Element_lbl, Element_lbl.getName());
int n = 2;
for (int i = 0; i < mappingPaths.size(); i++)
{
n = n + 1;
Mapping map = new Mapping();
map=(Mapping) mappingPaths.get(i);
sourcePath=(String) map.SourcePath;
Element_lbl = getlbl_element(n, sourcePath);
this.add(Element_lbl, Element_lbl.getName());
globalPaths=(Vector) map.GlobalPaths;
if (globalPaths != null)
{
globalPaths=(Vector) map.GlobalPaths;
functionName=(String) map.FunctionName;
int m=0;
for (int j = 0; j < globalPaths.size(); j++)
{
String gp=new String();
gp=(String) globalPaths.get(j);
if ((globalPaths.size()>1) && (j==0))
Element_lbl = getlbl_Gpath(n, gp+",");
else
Element_lbl = getlbl_Gpath(n, gp);
this.add(Element_lbl, Element_lbl.getName());
n = n + 1;
m=j+1;
}
Element_lbl = getlbl_function(n-m, functionName);
this.add(Element_lbl, Element_lbl.getName());
n=n-1;
}
else
{
Element_lbl = getlbl_Gpath(n, "Null");
this.add(Element_lbl, Element_lbl.getName());
}}}
catch (Throwable Exc)
{
handleException(Exc);
}}
private JLabel getlbl_element(int i, String name)
{
try
{
lbl_element = new JLabel();
lbl_element.setName(name);
lbl_element.setText(name);
lbl_element.setBounds(20, 10 + (i * 5) * height, 250, 20);
}
catch (Throwable Exc)
{ handleException(Exc); }
return lbl_element;
}
private JLabel getlbl_Gpath(int i, String name)
{
try
{
lbl_Gpath = new JLabel();
lbl_Gpath.setName(name);
lbl_Gpath.setText(name);
lbl_Gpath.setBounds(350, 10 + (i * 5) * height, 250, 20);
}
catch (Throwable Exc)
{ handleException(Exc); }
return lbl_Gpath;
}
private JLabel getlbl_function(int i, String name)
{
try
{
```

```
lbl_function = new JLabel();
lbl_function.setName(name);
lbl_function.setText(name);
lbl_function.setBounds(650, 10 + (i * 5) * height, 200, 20);
}
catch (Throwable Exc)
{ handleException(Exc); }
return lbl_function;
}
private JTextField gettxt_Field(int i)
{
try
{
txt_Filed = new JTextField();
txt_Filed.setName("Field" + i);
txt_Filed.setEditable(true);
txt_Filed.setText("");
txt_Filed.setBounds(450, 10 + (i * 5) * height, 100, 15);
textFieldsVector.add(txt_Filed);
}
catch (Throwable Exc)
{ handleException(Exc); }
return txt_Filed;
}
private JLabel getlbl_show(int i, String name)
{
try
{
lbl_show = new JLabel();
lbl_show.setName(name);
lbl_show.setText(name);
lbl_show.setBounds(20, 10 + (i * 5) * height, 100, 20);          }
catch (Throwable Exc)
{
handleException(Exc);
}
return lbl_show;
}
private JLabel getlbl_line(int i, String name)
{
try
{
lbl_line = new JLabel();
lbl_line.setName(name);
lbl_line.setText(name);
lbl_line.setBounds(10, 10 + (i * 5) * height, 300, 20);
}
catch (Throwable Exc)
{ handleException(Exc); }
return lbl_line;
}
private void handleException(Throwable exception)
{
System.out.println("Could not initialize the frame. Error: " +
exception);
}}
```

APPENDIX G

Java code for merging mapping information with XMKB

```
import org.jdom.*;
import org.jdom.input.SAXBuilder;
import java.io.IOException;
import java.util.*;
import java.io.FileOutputStream;
import org.jdom.output.XMLOutputter;
import java.io.File;
public class KBmerge
{
public static MainInterFaceF mycaller;
static TreeMap paths = new TreeMap();
static TreeMap elements = new TreeMap();
static Vector SourceMetadata = new Vector();
public KBmerge()
{
String lastSign = "";
paths = new TreeMap();
elements = new TreeMap();
}
public static void mergeMapping(MainInterFaceF caller, String filename)
{
File f = new File(filename);
if (!f.exists())
{
buildKB(caller, filename);
}
else
{
cumKB(caller, filename);
}}
private static void writeToFile(String fname, Document doc)
{
try
{
```

```
FileOutputStream out = new FileOutputStream(fname);
XMLOutputter op = new XMLOutputter();
op.output(doc, out);
out.flush();
out.close();
}
catch (IOException e)
{
System.err.println(e);
}}
public static void cumKB(MainInterFaceF caller, String filename)
{
SAXBuilder builder = new SAXBuilder();
mycaller = caller;
Vector sourceMetadata = new Vector();
try
{
String att = "";
Element integ, child;
sourceMetadata = (Vector) mycaller.SourceMetadata.clone();
Vector mappingPaths = (Vector) mycaller.MappingPaths;
String sourcePath = "";
String functionName = "";
Vector globalPaths = null;
Document doc = builder.build(filename);
Element root = doc.getRootElement();
Element information = root.getChild("DS_information");
int number = information.getAttribute("number").getIntValue();
number = number + 1;
String num = "" + number;
information.removeAttribute("number");
information.setAttribute("number", num.trim());
List children = information.getChildren();
ListIterator iterator = children.listIterator();
child = (Element) iterator.next();
att = child.getAttributeValue("name");
Element newSource = new Element("DS_Location");
newSource.setText(sourceMetadata.get(1).toString());
newSource.setAttribute("name", sourceMetadata.get(0).toString());
newSource.setAttribute("type", sourceMetadata.get(2).toString());
information.addContent(newSource);
Element structure = root.getChild("Med_component");
children = structure.getChildren();
iterator = children.listIterator();
int count = 0;
while (iterator.hasNext())
{
child = (Element) iterator.next();
att = child.getAttributeValue("path");
String paths = " ";
Mapping map = new Mapping();
map = (Mapping) mappingPaths.get(count);
sourcePath = (String) map.SourcePath;
functionName = (String) map.FunctionName;
if (functionName.trim().length() == 0)
{
functionName = "Null";
}
Element local = new Element("target");
globalPaths = (Vector) map.GlobalPaths;
if (globalPaths != null)
{ globalPaths = (Vector) map.GlobalPaths;
for (int j = 0; j < globalPaths.size(); j++)
{
String gp = new String();
gp = (String) globalPaths.get(j);
if ( (globalPaths.size() > 1))
{
if (j==0)
paths=gp;
else
paths = paths + "," + gp;
}
else
{
paths = gp;
}}}
else
{ paths = "Null";
functionName = "Null";
}
local.setText(paths);
local.setAttribute("name", sourceMetadata.get(0).toString());
local.setAttribute("fun", functionName);
child.addContent(local);
count++;
}
writeToFile(filename, doc);
}
catch (JDOMException e)
{
System.out.println(" is not well-formed.");
System.out.println(e.getMessage());
}
catch (IOException e)
{
System.out.println(e);
}}
public static void buildKB(MainInterFaceF caller, String xmlfile)
{
mycaller = caller;
Element concept;
Element dbase;
Element relations;
Vector sourceMetadata = new Vector();
sourceMetadata = (Vector) mycaller.SourceMetadata.clone();
Vector mappingPaths = (Vector) mycaller.MappingPaths;
String sourcePath = "";
String functionName = "";
Vector globalPaths = null;
Element root = new Element("XMKB");
Document doc = new Document(root);
Element DS_info = new Element("DS_information");
DS_info.setAttribute("number", "1");
```

```
Element DS_Loc = new Element("DS_Location");
DS_Loc.setText(sourceMetadata.get(1).toString());
DS_Loc.setAttribute("name", sourceMetadata.get(0).toString());
DS_Loc.setAttribute("type", sourceMetadata.get(2).toString());
DS_info.addContent(DS_Loc);
root.addContent(DS_info);
Element Med_comp = new Element("Med_component");
for (int i = 0; i < mappingPaths.size(); i++)
{
String paths = " ";
Mapping map = new Mapping();
map = (Mapping) mappingPaths.get(i);
sourcePath = (String) map.SourcePath;
functionName = (String) map.FunctionName;
if (functionName.trim().length() == 0)
{ functionName = "Null"; }
Element integrated = new Element("source");
integrated.setAttribute("path", sourcePath);
Element local = new Element("target");
globalPaths = (Vector) map.GlobalPaths;
if (globalPaths != null)
{
globalPaths = (Vector) map.GlobalPaths;
for (int j = 0; j < globalPaths.size(); j++)
{
String gp = new String();
gp = (String) globalPaths.get(j);
if ( (globalPaths.size() > 1))
{
if (j==0)
paths = gp;
else
paths = paths + "," + gp;
}
else
{
paths = gp;
}}}
else
{ paths = "Null";
functionName = "Null";
}
local.setText(paths);
local.setAttribute("name", sourceMetadata.get(0).toString());
local.setAttribute("fun", functionName);
integrated.addContent(local);
Med_comp.addContent(integrated);
}
root.addContent(Med_comp);
writeToFile(xmlfile, doc);
}
public static void main(String[] args)
{
KBmerge kb = new KBmerge();
}}
```

APPENDIX H

Sample of XMKB document

```
<?xml version="1.0" encoding="UTF-8" ?>
<DS_information number="4">
 <DS_Location name ="books.xml" type="XML
                        document" >http://www.w3schools.com/xquery</DS_Location >
 <DS_Location name=" bib.xml" type=" XML document">C :\prototype\doc </DS_Location>
 <DS_Location name=" SCMFMA" type="Relational Database">
                                  jdbc:oracle:thin: @helot:1521:oracle9 </DS_Location>
 <DS_Location name=" bookdata.xml" type="XML document">C:\prototype\doc</DS_Location>
</DS_information>
<Med_component>
<source path="/book" >
  <target name="books.xml" fun="Null">/bookstore/book</target>
  <target name="bib.xml" fun="Null" >/bib/book</target>
  <target name="SCMFMA" fun="Null"> /scmfma/book</target>
  <target name="bookdata.xml" fun=" Null">/bookdata/book</target>
  </source>
<source path="/book/price">
  <target name="books.xml" fun="RateExchange">/bookstore/book/price</target>
  <target name="bib.xml" fun="RateExchange">/bib/book/price</target>
  <target name="SCMFMA" fun="Null">Null</target>
  <target name="bookdata.xml" fun=" Null">/bookdata/book/price</target>
  </source>
<source path="/book/author">
  <target name="books.xml" fun="Null">Null</target >
  <target name="bib.xml" fun="Null" >/bib/book/author</target>
  <target name="SCMFMA" fun="Null">Null</target>
  <target name="bookdata.xml" fun=" Null">/bookdata/book/author</target>
  </source>
<source path="/book/author/full_name">
  <target name="books.xml" fun="Null">Null</target >
  <target name="bib.xml" fun="Null" >Null</target>
  <target name="SCMFMA" fun="Null">Null</target>
  <target name="bookdata.xml" fun=" Null">Null</target>
  </source>
<source path="/book/author/full_name/first_name">
  <target name="books.xml" fun="firstName">/bookstore/book/author </target>
  <target name="bib.xml" fun="Null" >/bib/book/author/first</target>
  <target name="SCMFMA" fun="firstName">/scmfma/book/author</target>
  <target name="bookdata.xml" fun=" firstName">/bookdata/book/author/name</target >
  </source>
<source path="/book/author/full_name/last_name">
  <target name="books.xml" fun="LastName">/bookstore/book/author< /target>
  <target name="bib.xml" fun="Null" >/bib/book/author/last</target >
  <target name="SCMFMA" fun="LastName">/scmfma/book/author</target>
  <target name="bookdata.xml" fun=" LastName">/bookdata/book/author/name</target>
  </source>
<source path="/book/title">
```

```
  <target name="books.xml" fun="Null">/bookstore/book/title</target>
  <target name="bib.xml" fun="Null" >/bib/book/title</target>
  <target name="SCMFMA" fun="Null"> /scmfma/book/title</target>
  <target name="bookdata.xml" fun=" Null">/bookdata/book/title</target>
  </source>
<source path="/book/year">
  <target name="books.xml" fun="Null">/bookstore/book/year</target>
  <target name="bib.xml" fun="Null" >Null</target>
  <target name="SCMFMA" fun="Null"> /scmfma/book/year</target>
  <target name="bookdata.xml" fun=" Null">Null</target>
  </source>
<source path="/book/publisher">
  <target name="books.xml" fun="Null">Null</target>
  <target name="bib.xml" fun="Null" >/bib/book/publisher</target>
  <target name="SCMFMA" fun="Null"> /scmfma/book/publisher</target>
  <target name="bookdata.xml" fun=" Null">Null</target>
  </source>
<source path="/book/editor">
  <target name="books.xml" fun="Null">Null</target>
  <target name="bib.xml" fun="Null" >/bib/book/editor</target>
  <target name="SCMFMA" fun="Null">Null</target>
  <target name="bookdata.xml" fun=" Null">Null</target>
  </source>
<source path="/book/editor/affiliation">
  <target name="books.xml" fun="Null">Null</target>
  <target name="bib.xml" fun="Null" >/bib/book/editor/affiliation</target>
  <target name="SCMFMA" fun="Null">Null</target>
  <target name="bookdata.xml" fun=" Null">Null</target>
  </source>
<source path="/book/editor/full_name">
  <target name="books.xml" fun="Null">Null</target>
  <target name="bib.xml" fun="Merge">/bib/book/editor/last,/bib/book/editor/first</target>
  <target name="SCMFMA" fun="Null">Null</target>
  <target name="bookdata.xml" fun=" Null">Null</target>
  </source>
  </Med_component>
  </XMKB>
```

APPENDIX I

Java code for removing data source from XMKB

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import org.jdom.*;
import org.jdom.input.SAXBuilder;
import java.io.IOException;
import java.util.*;
import java.io.FileOutputStream;
import org.jdom.output.XMLOutputter;
class RemoveSources extends JDialog implements ActionListener
{
private JPanel pnl_txt = null;
JComboBox txt_Func = new JComboBox();
private JLabel lbl_show = null;
private JButton btn_ok = null;
private JButton btn_cancel = null;
private Vector sources = new Vector();
private JPanel JFrameContentPane = null;
public String kbase = "";
JComboBox CB = new JComboBox();
public Frame fram;
private int y2 = 152, y3 = 12;
private int height = 5;
public RemoveSources(String kb)
{
kbase = kb;
sources = (Vector) getSources(kbase);
UIManager.put("Label.font", new Font("SansSerif", Font.BOLD, 12));
UIManager.put("Button.font", new Font("SansSerif", Font.BOLD, 12));
UIManager.put("TextField.font", new Font("SansSerif", Font.BOLD, 12));
UIManager.put("ComboBox.font", new Font("SansSerif", Font.PLAIN, 10));
UIManager.put("TextArea.font", new Font("SansSerif", Font.BOLD, 12));
setContentPane(getJFrameContentPane());
this.setSize(500, 200);
setTitle("Remove Data Source");
}
public void actionPerformed(ActionEvent e)
```

```
{
if (e.getSource() == btn_cancel)
{
dispose();
return;
}
else if (e.getSource() == btn_ok)
{
String selectedSource = txt_Func.getSelectedItem().toString();
removeSource(kbase, selectedSource);
JOptionPane.showMessageDialog(this,selectedSource +" has been removed
successfully");
dispose();
}}
private JPanel getJFrameContentPane()
{
if (JFrameContentPane == null)
{
try
{
JFrameContentPane = new JPanel();
JFrameContentPane.setName("JFrameContentPane");
JFrameContentPane.setLayout(null);
getJFrameContentPane().add(getlbl_show(0, "Remove Data Source"),
getlbl_show(0, "Source").getName());
getJFrameContentPane().add(gettxt_sources(0),
gettxt_sources(0).getName());
getJFrameContentPane().add(getbtn_cancel(5),
getbtn_cancel(5).getName());
getJFrameContentPane().add(getbtn_ok(5), getbtn_ok(5).getName());
}
catch (Throwable Exc)
{
handleException(Exc);
}}
return JFrameContentPane;
}
private JButton getbtn_ok(int i)
{
if (btn_ok == null)
{
try
{
btn_ok = new JButton();
btn_ok.setName("btn_ok");
btn_ok.setText("Remove");
btn_ok.setBounds(250, 100 + i * height, 85, 25);
btn_ok.addActionListener(this);
}
catch (Throwable Exc)
{
handleException(Exc);
}}
return btn_ok;
}
private JComboBox gettxt_sources(int i)
{
```

```
try
{
txt_Func = new JComboBox(sources);
txt_Func.setBackground(Color.white);
txt_Func.setName("Sources");
txt_Func.setBounds(200, 20 + (i * 5) * height, 200, 25);
}
catch (Throwable Exc)
{ handleException(Exc); }
return txt_Func;
}
private JButton getbtn_cancel(int i)
{
if (btn_cancel == null)
{
try
{
btn_cancel = new JButton();
btn_cancel.setName("btn_cancel");
btn_cancel.setText("Cancel");
btn_cancel.setBounds(350, 100 + i * height, 85, 25);
btn_cancel.addActionListener(this);
}
catch (Throwable Exc)
{
handleException(Exc);
}}
return btn_cancel;
}
private JLabel getlbl_show(int i, String name)
{
try
{
lbl_show = new JLabel();
lbl_show.setName(name);
lbl_show.setText(name);
lbl_show.setBounds(50, 20 + (i * 5) * height, 200, 25);
}
catch (Throwable Exc)
{ handleException(Exc); }
return lbl_show;
}
private void handleException(Throwable exception)
{
System.out.println("Could not initialize the frame. Error:"+ exception);
}
private static void writeToFile(String fname, Document doc)
{
try
{
FileOutputStream out = new FileOutputStream(fname);
XMLOutputter op = new XMLOutputter();
op.output(doc, out);
out.flush();
out.close();
}
catch (IOException e)
```

```
{
System.err.println(e);
}}
public static Vector getSources(String filename)
{
SAXBuilder builder = new SAXBuilder();
Vector sources = new Vector();
try
{
String att = "";
Element child;
Document doc = builder.build(filename);
Element root = doc.getRootElement();
Element information = root.getChild("DS_information");
java.util.List children = information.getChildren();
ListIterator iterator = children.listIterator();
while (iterator.hasNext())
{
Element source = (Element) iterator.next();
att = source.getAttributeValue("name");
sources.add(att);
}}
catch (JDOMException e)
{
System.out.println(" is not well-formed.");
System.out.println(e.getMessage());
}
catch (IOException e)
{
System.out.println(e);
}
return sources;
}
public static void removeSource(String filename, String Source)
{
SAXBuilder builder = new SAXBuilder();
Vector sourceMetadata = new Vector();
try
{
String att = "";
Element integ, child;
Document doc = builder.build(filename);
Element root = doc.getRootElement();
Element information = root.getChild("DS_information");
int number = information.getAttribute("number").getIntValue();
number = number - 1;
String num = "" + number;
information.removeAttribute("number");
information.setAttribute("number", num.trim());
java.util.List children = information.getChildren();
ListIterator iterator = children.listIterator();
int index = -1;
int count = 0;
while (iterator.hasNext())
{
Element source = (Element) iterator.next();
att = source.getAttributeValue("name");
if (att.toString().equalsIgnoreCase(Source))
{ index = count; }
count++;
}
if (index > -1)
{ children.remove(index); }
Element structure = root.getChild("Med_component");
children = structure.getChildren();
iterator = children.listIterator();
count = 0;
while (iterator.hasNext())
{
child = (Element) iterator.next();
java.util.List locals = child.getChildren();
ListIterator iterator1 = locals.listIterator();
index = -1;
count = 0;
while (iterator1.hasNext())
{
Element local = (Element) iterator1.next();
att = local.getAttributeValue("name");
if (att.toString().equalsIgnoreCase(Source))
{ index = count; }
count++;
}
if (index > -1)
{
locals.remove(index);
}}
writeToFile(filename, doc);
}
catch (JDOMException e)
{
System.out.println(" is not well-formed.");
System.out.println(e.getMessage());
}
catch (IOException e)
{
System.out.println(e);
}}
public static void main(String args[])
{
JDialog a = new RemoveSources(null);
a.addWindowListener(new WindowAdapter()
{
public void windowClosing(WindowEvent e)
{
System.exit(0);
}});
a.show();
}}
```

APPENDIX J

Query Processor and XFEP code for parsing XQuery FLWR Expression query

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

public class QueryProcessor extends JFrame implements ActionListener
{
private JTextArea textArea1,textArea2,textArea3;
public JButton run,reset,exit,getschema;
private JLabel label1,label2,label3,label4,label5,label6;
private JPanel buttonPanel;
String integratedfile;
public QueryProcessor()
{
super("QUERY PROCESSOR");
Box box =Box.createVerticalBox();
label2 = new JLabel ("MASTER VIEW");
box.add(label2);
textArea1 = new JTextArea(17,30);
textArea1.setEditable(false);
box.add(new JScrollPane(textArea1));
label4 = new JLabel ("ENTER YOUR XQUERY : ");
box.add(label4);
textArea2= new JTextArea(3,30);
box.add(new JScrollPane(textArea2));
label6 = new JLabel ("THE RESULT :  ");
box.add(label6);
textArea3= new JTextArea(15,30);
textArea3.setEditable(false);
box.add(new JScrollPane(textArea3));
Container container = getContentPane();
container.add(box);
```

```
getschema = new JButton("Get Master View");
run = new JButton  ("Generate Local Sub-Query");
reset = new JButton("  Reset   ");
exit = new JButton ("    Exit     ");
run.setEnabled(false);
getschema.addActionListener(this);
run.addActionListener(this);
reset.addActionListener(this);
exit.addActionListener(this);
buttonPanel= new JPanel();
buttonPanel.setLayout(new GridLayout(1,3));
buttonPanel.add (getschema);
buttonPanel.add (run);
buttonPanel.add (reset);
buttonPanel.add (exit);
container.add(buttonPanel,BorderLayout.SOUTH);
label5 = new JLabel (" ");
box.add(label5);
setSize(1020,740);
setVisible (true);
}
public void actionPerformed (ActionEvent event)
{
if (event.getSource() == getschema)
{
final JFileChooser vc = new JFileChooser();
int returnVal = vc.showOpenDialog(this);
if (returnVal == JFileChooser.APPROVE_OPTION)
{
File file1 = vc.getSelectedFile();
integratedfile = file1.getAbsolutePath();
textArea1.setText("");
try
{
FileInputStream fstream = new FileInputStream(integratedfile);
DataInputStream in = new DataInputStream(fstream);
String output="";
while (in.available() !=0)
{
output += (in.readLine())+"\n";
}
in.close();
textArea1.append(output);
run.setEnabled(true);
textArea2.setText("");
textArea3.setText("");
}
catch (Exception e)
{
System.err.println("File input error");
}
}
}
else if (event.getSource() == run)
{
QueryParser application = new QueryParser();
String query = textArea2.getText();
try
{
int ln=integratedfile.length();
String kbName1=integratedfile.substring(0,ln-4)+"_kb.xml";
Vector q=(Vector) application.GetQueries(kbName1,query);
for (int i=0;i< q.size();i++)
{
textArea3.append((q.get(i)).toString());
textArea3.append("---------------------------------------------\n");
}
}
catch(Exception excp)
{
if ( query .equals (""))
JOptionPane.showMessageDialog(null,"Please enter your XQuery
query","Error Message",JOptionPane.ERROR_MESSAGE);
else
JOptionPane.showMessageDialog(null,"Please check your XQuery
query","Error Message", JOptionPane.ERROR_MESSAGE);
}
}
else if (event.getSource() == reset)
{
textArea2.setText("");
textArea3.setText("");
}
else if (event.getSource() == exit)
{
dispose();
}
}
public static void main (String args[])
{
QueryProcessor application = new QueryProcessor();
application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}

-------------------------------------------------------------------
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;
import org.jdom.*;
import org.jdom.input.SAXBuilder;
import java.io.IOException;
import java.util.*;
import java.io.FileOutputStream;
import org.jdom.output.XMLOutputter;
import java.io.File;

public class QueryParser
{
int slash,wh,in,variable,strat,end,cond,stag,etag,t1,le;
StringTokenizer st1,st2;
public QueryParser()
{
}
public  Vector GetQueries (String file,String query1)
{
Vector queries = new Vector();
String VarRetrived[]=new String[100];
String correspondRet[]=new String[100];
String RelVar[]=new String[100];
String funRet[]=new String[100];
String tablepath[]=new String[100];
String myQuery=null;
String query = query1;
String condition="",mainPath="",mainvar="";
String retrivedVar="",output="",operator="";
String condVar="",condVal="",condition1="";
String stagName="",etagName="",path="";
String sourceName="",sourceType="",sourceLoca="";
String correspondPath="",fun="",condVarReal="";
String correspondcon="",funcon="",seprator="";
String subquery="",relquery="",Relcon="";
String table ="";
String filename=file;
variable = query.indexOf("$");
if (query.toUpperCase().indexOf("IN") > -1)
in = query.toUpperCase().indexOf("IN");
mainvar = query.substring(variable,in);
slash= query.indexOf("/");
if (query.toUpperCase().indexOf("WHERE") > -1)
{
wh = query.toUpperCase().indexOf("WHERE");
condition = query.substring(wh+5,query.toUpperCase().indexOf("RETURN"));
}
else wh = query.toUpperCase().indexOf("RETURN");
mainPath = query.substring(slash,wh);
strat = query.indexOf("{");
end = query.indexOf("}");
retrivedVar = query.substring(strat+1,end);
condition1 = condition.trim();
if (condition1.indexOf("=")> -1)
{
operator = "=";
condVar = condition1.substring(0,condition1.indexOf("="));
condVal = condition1.substring(condition1.indexOf("=")+1);
}
else if (condition1.indexOf(">")> -1)
{
operator = ">";
condVar = condition1.substring(0,condition1.indexOf(">"));
condVal = condition1.substring(condition1.indexOf(">")+1);
}
else if (condition1.indexOf("<")> -1)
{
operator = "<";
condVar = condition1.substring(0,condition1.indexOf("<"));
condVal = condition1.substring(condition1.indexOf("<")+1);
}
if (condition1.indexOf(">=")> -1)
{
```

```
operator = ">=";
condVar = condition1.substring(0,condition1.indexOf(">="));
condVal = condition1.substring(condition1.indexOf(">=")+2);
}
if (condition1.indexOf("<=")> -1)
{
operator = "<=";
condVar = condition1.substring(0,condition1.indexOf("<="));
condVal = condition1.substring(condition1.indexOf("<=")+2);
}
if (condition1.indexOf("!=")> -1)
{
operator = "!=";
condVar = condition1.substring(0,condition1.indexOf("!="));
condVal = condition1.substring(condition1.indexOf("!=")+2);
}
mainvar = removeSpaces(mainvar.trim());
mainPath = removeSpaces(mainPath.trim());
retrivedVar = removeSpaces(retrivedVar.trim());
condition = condition.trim();
condVar = condVar.trim();
condVal = condVal.trim();
mainPath = "/" + mainPath.substring(1).trim();
if (condVar != "")
{
t1 = condVar.indexOf("/");
condVar = mainPath +"/"+ condVar.substring(t1+1).trim();
}
st1= new StringTokenizer(retrivedVar,",");
int i=0;
while (st1.hasMoreTokens())
{
VarRetrived[i] = st1.nextToken().trim();
i++;
}
for (int e = 0 ; e < i ; e++)
{
t1 = VarRetrived[e].indexOf("/");
VarRetrived[e] = mainPath +"/"+ VarRetrived[e].substring(t1+1).trim();
}
stag = query.toUpperCase().indexOf("RETURN");
stagName = query.substring(stag+6,query.toUpperCase().indexOf("{"));
etag = query.indexOf("}");
etagName = query.substring(etag+1);
stagName=stagName.trim();
etagName=etagName.trim();
SAXBuilder builder = new SAXBuilder();
try
{
Document doc = builder.build(filename);
Element root = doc.getRootElement();
Element information = root.getChild("DS_information");
int number = information.getAttribute("number").getIntValue();
java.util.List children = information.getChildren();
ListIterator iterator = children.listIterator();
while (iterator.hasNext())
{
```

```
int test=0,test1=0;
Element source = (Element) iterator.next();
sourceName = source.getAttributeValue("name");
sourceType = source.getAttributeValue("type");
sourceLoca = source.getText();
if ( sourceLoca.indexOf("/") > -1)
seprator="/";
else
seprator="\\";
Element Med_com = root.getChild("Med_component");
java.util.List children1 = Med_com.getChildren();
ListIterator iterator1 = children1.listIterator();
while (iterator1.hasNext())
{
Element integrated = (Element) iterator1.next();
path = integrated.getAttributeValue("path");
if (path .equals (mainPath.trim()))
{
java.util.List children2 = integrated.getChildren();
ListIterator iterator2 = children2.listIterator();
while (iterator2.hasNext())
{
Element target = (Element) iterator2.next();
if (sourceName .equals (target.getAttributeValue("name")))
{
correspondPath = target.getText();
fun = target.getAttributeValue("fun");
}
}
}
if ( condVar != "" && path .equals (condVar))
{
java.util.List children3 = integrated.getChildren();
ListIterator iterator3 = children3.listIterator();
while (iterator3.hasNext())
{
Element target = (Element) iterator3.next();
if (sourceName .equals (target.getAttributeValue("name")))
{
correspondcon = target.getText();
Relcon = target.getText();
if ( (correspondcon.compareTo("Null")!= 0) &&
( correspondPath.compareTo("Null")!= 0))
{
le = correspondPath.length();
correspondcon = mainvar+correspondcon.substring(le);
}
funcon = target.getAttributeValue("fun");
}
}
}
for (int e = 0 ; e < i ; e++)
{
if ( path .equals (VarRetrived[e]))
{
java.util.List children4 = integrated.getChildren();
ListIterator iterator4 = children4.listIterator();
while (iterator4.hasNext())
{
Element target = (Element) iterator4.next();
if (sourceName .equals (target.getAttributeValue("name")))
{
correspondRet[e] = target.getText();
funRet[e] = target.getAttributeValue("fun");
RelVar [e] = target.getText();
if ((correspondRet[e].compareTo("Null")!= 0) &&
( correspondPath.compareTo("Null")!= 0))
{
le = correspondPath.length();
int t2 = correspondRet[e].indexOf(",");
if (t2  > -1)
correspondRet[e] = mainvar+correspondRet[e].substring(le,t2)+" , "
+mainvar+correspondRet[e].substring(t2+le+1);
else
correspondRet[e] = mainvar+correspondRet[e].substring(le);
}
}
}
}
}
}
if (sourceType .equals ("XML document"))
{
subquery = "FOR "+mainvar+" IN document(\""+sourceLoca + seprator
+sourceName+"\")"+correspondPath;
if (condVar != "")
subquery = subquery + " WHERE "+correspondcon+operator+condVal;
subquery = subquery +" RETURN "+ stagName+" { ";
for (int e = 0 ; e < i ; e++)
{
if (funRet[e] .equals ("Null"))
subquery = subquery + correspondRet[e];
else
subquery = subquery + funRet[e]+"("+correspondRet[e]+")";
if (e != i-1)
subquery = subquery +" , ";
}
subquery = subquery +" } "+etagName;
}
else
{
correspondPath = correspondPath.substring(1).replace('/','.');
st2= new StringTokenizer(correspondPath,".");
int j=0;
while (st2.hasMoreTokens())
{
tablepath[j] = st2.nextToken().trim();
j++;
}
table = tablepath[0] +"."+ tablepath[1];
subquery = "Select  ";
for (int e = 0 ; e < i ; e++)
{
if (funRet[e] .equals ("Null"))
```

```
subquery = subquery + RelVar[e].substring(1).replace('/','.');
else
subquery=subquery+funRet[e]+
"("+RelVar[e].substring(1).replace('/','.')+")";
if (e != i-1)
subquery = subquery +" , ";
}
subquery = subquery +"  From  "+table;
if (condVar != "")
subquery = subquery + " WHERE "+Relcon.substring(1).replace('/','.')+
operator+condVal.replace('"','\'');
}
int b=1;
for (int e = 0 ; e < i ; e++)
{
if (correspondRet[e].compareTo("Null")== 0) b=-1;
}
myQuery = "Sub-Query Generate For "+sourceType+" "+sourceLoca + seprator
+sourceName +" is :\n";
if ((correspondPath .equals ("Null")) || (correspondcon .equals
("Null")) || (b==-1))
myQuery =myQuery+ "No matched Query Generated For This Dtad
Source"+"\n\n";
else
myQuery=myQuery+subquery+"\n\n";
queries.add(myQuery);
}
}
catch (JDOMException e)
{
System.out.println(e.getMessage());
}
catch (IOException e)
{
System.out.println(e);
}
return queries;
}
public static void main (String args[])
{
QueryParser application = new QueryParser();
}
public static String removeSpaces(String s)
{
StringTokenizer st = new StringTokenizer(s," ",false);
String t="";
while (st.hasMoreElements()) t += st.nextElement();
return t;
}
}
```