MEDIATION ON XQUERY VIEWS

Xiaobo Peng

Dissertation Prepared for the Degree of

DOCTOR OF PHILOSOPHY

UNIVERSITY OF NORTH TEXAS

December 2006

APPROVED:

Robert Brazile, Major Professor
Kathleen M. Swigger, Committee Member
Yan Huang, Committee Member
Armin Mikler, Program Coordinator
Krishna M. Kavi, Chair of Department of
Computer Science and Engineering
Oscar Garcia, Dean of the College of
Engineering
Sandra L. Terrell, Dean of the Robert B.
Toulouse School of Graduate Studies

Peng, Xiaobo, Mediation on XQuery Views. Doctor of Philosophy (Computer Science), December 2006, 141 pp., 3 tables, 15 illustrations, bibliography, 90 titles.

The major goal of information integration is to provide efficient and easy-to-use access to multiple heterogeneous data sources with a single query. At the same time, one of the current trends is to use standard technologies for implementing solutions to complex software problems. In this dissertation, I used XML and XQuery as the standard technologies and have developed an extended projection algorithm to provide a solution to the information integration problem.

In order to demonstrate my solution, I implemented a prototype mediation system called Omphalos based on XML related technologies. The dissertation describes the architecture of the system, its metadata, and the process it uses to answer queries. The system uses XQuery expressions (termed metaqueries) to capture complex mappings between global schemas and data source schemas. The system then applies these metaqueries in order to rewrite a user query on a virtual global database (representing the integrated view of the heterogeneous data sources) to a query (termed an outsourced query) on the real data sources. An extended XML document projection algorithm was developed to increase the efficiency of selecting the relevant subset of data from an individual data source to answer the user query. The system applies the projection algorithm to decompose an outsourced query into atomic queries which are each executed on a single data source. I also developed an algorithm to generate integrating queries, which the system uses to compose the answers from the atomic queries into a single answer to the original user query. I present a proof of both the extended XML document projection algorithm and the query integration algorithm. An

analysis of the efficiency of the new extended algorithm is also presented. Finally I describe a collaborative schema-matching tool that was implemented to facilitate maintaining metadata.

ACKNOWLEDGEMENTS

I would like to thank my defense committee members: Dr. Robert Brazile, Dr. Kathleen M. Swigger and Dr. Yan Huang for their guidance and encouragement in my pursuit of Ph. D. study. I would never have accomplished it without their help.

My deepest gratitude goes to my major professor Dr. Robert Brazile for his excellent guidance and many fruitful discussions on my research subject. He is the most important mentor in my academic development and taught me how to do research. I am indebted to Dr. Kathleen M. Swigger and Dr. Robert Brazile for their many insights on conducting research and generous financial support. They have provided an excellent work environment for me to work on my research. Their kind encouragement and patience enabled me to overcome many difficult times. I thank Dr. Robert Brazile, Dr. Kathleen M. Swigger and Dr. Yan Huang for much good advice on how to write research papers and my dissertation. Without their critical comments and patiently correcting my writing, this dissertation could never reach the current quality. I highly appreciate the enormous time they spent for me.

I would like to thank my wife Zhiqi Chen not only for her long term spirit support, but also the creation of most test data and some diagrams.

Finally, I would like to thank my fellow student Brian Harrington for many discussions on the research subject and programming issues.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1. Introduction

The information explosion that began in the last century has made information integration one of the key technologies for enterprises in this decade. In an information integration scenario, there are multiple data sources that contain information that is of interest to many users. Unfortunately, these data sources are often autonomous and heterogeneous and must be integrated into a single view. One way to integrate autonomous data sources is to use a mediator, which is a software component that provides a logical database system on top of existing data sources. The mediator does not store the data. Rather it requests the relevant data from its member sources and uses that data to formulate a response to a user's query. Therefore, one of the major challenges for mediator systems is capturing the mappings between the mediated schema and the local schemas identified with the heterogeneous data sources. Another major challenge is creating ways to translate user queries into atomic queries.

To respond to these challenges, this research sought to provide efficient and easy-to-use access to multiple heterogeneous data sources with a single query. This research achieves this goal by 1) representing the mappings between the global schema and the local schemas as XQuery expressions and 2) using a decomposition algorithm based on extended XML document projection. Rather than using a programming language or fabricated logic, this research uses XQuery to represent the mappings between data sources because of its power in processing and modeling the data. XQuery alone, without the help from other programming languages, can capture the complex data extraction, transformation, and restructuring that are required for expressing mappings. An extension of the Marian and Simeon algorithm was then implemented as a method for rewriting user queries into atomic queries. The resulting system was then evaluated and the outcome of this evaluation is presented. The resulting system gives users a powerful information integration system that allows flexible partial integration, scales well, and is easy to administer.

Thus the most important contributions of this work are 1) using XML technologies to describe mappings between a global schema and local data sources, and 2) extending the Marian and Simeon projection algorithm for use with information integration. An important part of the metadata of my solution to the information integration problem is the set of XQuery expressions that capture mappings between mediated schemas and local schemas of heterogeneous data sources. These XQuery expressions define mediated data as views on local source data. XQuery is a declarative and functional language. Thus the mediated views in the final system are easy to understand and maintain. XQuery is also powerful enough to express complicated data extraction, transformation, and restructuring during information integration. Moreover, the XQuery expressions represented in the metadata make it easy to rewrite a user query on the mediated schema into a query (which is termed outsourced query) on local schemas. An algorithm can simply substitute mapping expressions from the metadata into the user query on the global virtual data base, thus creating a query on local (real) data sources.

Once the metadata is represented, the next major challenge is translating outsourced user queries into atomic queries and integrating those queries. This problem was solved by extending the Marian and Simeon projection algorithm. In their previous work, Marian and Siméon developed a projection algorithm for XML documents to enable memory-based XQuery engines to process significantly larger documents [52]. However, their algorithm does not analyze navigation on constructed elements, which play a key role in my information integration solution. This research extends projection analysis onto constructed elements with literal or computed names and applies it to information integration. The extended projection algorithm enables the mediation system to pinpoint the relevant data needed to answer a user query from local data sources significantly more accurately than the Marian and Siméon algorithm. Thus my algorithm significantly reduces the amount of data transmitted over a network. The projection paths produced by this extended projection algorithm can be used directly as atomic queries of the mediation system or they can be converted to XQuery atomic queries. The Marian and Simeon algorithm was designed to reduce the amount of memory required by XQuery engines. This research extended that algorithm so that it could be used by an information integration system, in order to reduce the amount of data transmitted across a network.

The following sections contain an extensive survey of the published literature. The review begins with an overview of information integration, approaches that are used in information integration, and a description of mediator systems. The review then continues with an overview of XML technologies. Finally, the literature that relates to this specific research is presented.

1.2. Information Integration

In an information integration scenario, there are multiple data sources that contain information that is of interest to users. These data sources are usually autonomous and heterogeneous. The data sources can be databases, e.g. legacy databases, relational databases, XML databases, etc.; They can be web pages or XML files that are available through the Internet; They can be file repositories, e.g. text files, Microsoft®[1] Word files, PDF files, postscript files, etc. Because users do not have the time or desire to learn how to search every different data source, they need an information integration system that can provide an integrated view of all the information available in all the local data sources. Moreover, information integration systems have the potential to answer user queries that may be unsolvable by any single data source. Information integration systems have the ability to access relevant data scattered

[1]Microsoft Corporation, www.microsoft.com

across different data sources, process them to reconcile any discrepancy among them, and restructure them or derive new data.

The various applications of information integration occur in any area that needs to collect and process a significant amount of data in different formats and/or places, such as businesses, government agencies, researchers in physics, chemistry, bioinformatics, geoinformatics, etc. The acronym *EII* is a popular buzzword for enterprise information integration, which provides uniform data services by mediating the heterogeneous data sources within an enterprise. These types of systems are increasingly impacting our everyday lives, as we use them in Internet applications such as shopping carts, travel aggregators, etc.

1.3. Information Integration Approaches

There are essentially three approaches that are used to provide integrated services among multiple autonomous and heterogeneous data sources: federation, warehousing, and mediation [27, 30]. The federated approach integrates data by defining mappings between all pairs of schemas of the member databases [30]. If a user query cannot be answered by a member database, that database will, in turn, query other member databases to compute the answer. Although the federated systems are relatively easy to implement, they do not scale well. For example, if you have n member databases, you will need $n(n-1)$ mappings. If you add one new member, you will need to add $2n$ new mappings.

In the second approach, data from the local data sources are imported into a single database called a data warehouse [9]. With this method, the underlying data sources are still operational since the data is replicated for the warehouse. Data warehouses are usually not updated immediately after a local data source has changed. Generally, they are either totally reconstructed, which is very time consuming, or incrementally updated. Thus, data warehouses do not contain the most recent data. Data warehouses are more effective as long-term repositories for critical data or as storehouses for historical data that can be processed and mined [27]. Decision-making support is one of the important application areas of data warehouses.

The mediator approach has emerged as a popular solution to the problem of integration of heterogeneous databases. A mediator is a software component that provides a logical database system on top of existing data sources. The mediator does not store the data. Rather it requests the relevant data from its member sources and uses that data to formulate a response to a user's query. A considerable number of mediated query systems have been proposed including TSIMMIS [15, 28], DIOM [50], Sybil [42], and HERMES [78]. There are also many mediation systems discussed in [34, 38, 21]. The major advantage of this type of approach is that it provides the user with the most current information and it scales well. Mediation systems are able to provide users with the most recent information because the system accesses the local data sources, not a cached copy. It scales well mainly for two reasons. First, it can be easily extended to include additional data sources, since one only needs to add one mapping for each new data source; namely the one between the mediator and the new data source. Second, mediators can be stacked into a hierarchy of mediators. This hierarchy structure allows partial integration of the repositories, which facilitates the incremental integration of a high number of data sources.

1.4. Mediation

In a mediation system, users post their queries on the virtual global data source, which is exported by the mediator. The mediator's schema is called the global schema or mediated schema. In order to answer user queries on a virtual global data source, the queries must be transformed on real local data sources according to some relationship or mapping specified between the global and local schemas. The major problem is capturing the complex schema mappings. The approaches that have been suggested for capturing the mappings are global-as-view (GAV) [48], local-as-view (LAV) [48], and global-and-local-as-view (GLAV) [26, 46], which combines GAV and LAV. In the GAV approach, the global schema is defined as views on local schemas. For example, the global relations are expressed by horn rules as views on the local relations [48]. The advantages of the GAV approach is that it is easier to rewrite user queries posted on the global schema to queries on the local data source because you just substitute the relations of the global schema in user queries with the corresponding view definitions. Its weakness is that it is more difficult to scale than the LAV approach because whenever a new data source is added, all the views must be re-examined and subsequently might need to be rewritten. In contrast to the GAV technique, the LAV approach describes the contents of local data sources as queries, i.e. views over the relations in the global schema. This makes the LAV approach easier to scale than GAV because adding a new data source is completely independent of other existing data sources. It is also easier to specify constraints on the contents of data sources. However, it is more difficult to answer user queries in the LAV approach because it requires the use of more complicated and time consuming algorithms such as the Bucket algorithm [49], the model elimination theorem prover [22], the Inverse-rules algorithm [70, 23], or the MiniCon algorithm [69]. The data sources can be considered as materialized answers to queries over a global schema. Thus, the problem of processing user queries is to answer user queries using only views. It is a problem that is studied in the area of query optimization, which is found in traditional database research.

1.5. XML Technologies

XML [84] is a very flexible format for data representation, data storage, and data exchange. The W3C XML Schema [85] supports rich data types and other constraints to restrict XML document content. XQuery [87, 88, 86] is a powerful language to search and transform XML data. XML was originally designed for narrative-centric applications. It is a simplified version of the very complicated Standardized General Markup Language (SGML). The Document Type Definition (DTD) has been used to define and validate the structures of XML documents.

XML has found widespread applications in a data-centric world and has become a de facto standard for data exchange. For example, Microsoft® BizTalk™ is a framework that enables business processes to communicate with each other using messages in XML [81]. Web services use XML based standards such as SOAP, WSDL and UDDI to define and provide services.

XML is also a superior long-term data storage language. In the past, many people stored data in non-standard or binary formats, many of which are no longer understandable. For example, much of the moon landing data cannot be read because no one remembers its format [36]. Java previously used binary format to serialize objects, but they now recommend using XML for long-term storage. Leading relational databases including Oracle®[2] 9i™, Microsoft®[3] SQL Server™ 2000, IBM®[4] DB2®, and MySQL®[5] all support XML. There are also native XML databases, e.g. eXist, Xindice and Tamino™. A good discussion of this problem can be found in [13].

In data-centric applications, however, richer data types and other constraints are needed to restrict the contents of documents. Since DTDs do not have these features, other schema definition languages are designed for this purpose. W3C XML Schema [85] is the one that is defined and recommended by W3C. It provides 44 built-in data types and allows new data types to be defined; It supports uniqueness and referential constraints; It also provides better occurrence constraints than DTD. Other significant XML Schema languages include Relax NG [59, 17] and Schematron [40, 39]. Relax NG was based on two other earlier XML Schema proposals: TREX (Tree Regular Expressions for XML) [18] and RELAX (REgular LAnguage description for XML) [58]. RELAX NG is relatively simpler than W3C XML Schema. Rick Jelliffe, the author of Schematron, described Schematron as "a feather duster for the furthest

[2]ORACLE CORPORATION, www.oracle.com

[3]Microsoft Corporation, www.microsoft.com

[4]IBM, www.ibm.com

[5]MySQL AB, www.mysql.com

corners of a room where the vacuum cleaner (DTD) cannot reach." Both Relax NG and Schematron later became parts of ISO/IEC 19757 - DSDL Document Schema Definition Languages [37].

1.6. Related Work

Researchers have developed a variety of languages to describe mappings and to write queries. For example, Levy discussed Logic-Based Techniques In Data Integration [48, 47]. Restricted forms of first-order formulas are used to describe mappings. Queries are expressed by Horn rules. For systems that use the GAV approach, user queries can be reformulated by unfolding, using mapping rules. For systems that use the LAV approach, the bucket algorithm or the inverse-rules algorithm which are much more complicated are used for query reformulation.

Some examples of information integration systems that use the GAV approach are the TSIMMIS mediation system [29, 15], MIX [3], MOMIS [4], HERMES [78], Gestalt [71], Garlic [34], Amos II [72], COIN [33], Disco [80], IBIS [8], Observer, Squirrel[90], SilkRoute [25], XPERANTO [10], and YAT [16]. Systems that use the LAV approach are Agora [51], the Information Manifold [43] and Infomaster [22].

A variety of different languages are used to represent mappings and express queries. For example, the TSIMMIS mediation system, which uses the GAV approach, is based on an object model called the object exchange model (OEM). The query language for the TSIMMIS's mediator and wrapper is MSL, a logic language similar to datalog but with the power to express semi-structured data [29]. The MSL queries are first normalized and then reformulated into queries using matching and unification [62, 63]. The TSIMMIS system generates mediators and wrappers from their MSL specifications. The Agora data integration system, a LAV system, accepts user queries in XQuery [51]. However, its wrappers export all data sources as SQL data sources. It translates XQuery queries into SQL queries on the virtual generic schema and then rewrites the SQL queries on the generic schema into SQL queries on the real data sources using translation rules. The Information Manifold system uses a Descriptive Logic to specify global schemas. The e-XMLMedia integrates data sources with XML and XQuery [31, 32]; however, the mapping is not expressed in XQuery. The MIX mediator accepts queries in the XMAS query language, and its resolution module uses a mediator view definition in the XMAS language to rewrite a user query into a set of queries on the wrapper views [2].

There are a number of systems that use the object-oriented model. For example, Garlic [34] is based on the ODMG object-oriented data model, and it uses dynamic programming for query optimization. Disco [80] is built around the ODMG object-oriented data model, and its query language is OQL. Disco also describes its schemas using the ODL language. MOMIS [4] is built around ODMG CORBA technologies. MOMIS uses ODM_{I^3} common data model and the OQL_{I^3} query language. It supports semi-automatic generation of a global schema called the Global Virtual View (GVV) from the ODL_{I^3} descriptions that it receives from its wrappers. ODM_{I^3}, ODL_{I^3} and OQL_{I^3} are all *subsets of corresponding ones in ODMG*. The Query Manager for the MOMIS mediator is responsible for processing the queries. It accepts user queries on the global schema and produces atomic queries in OQL_{I^3} for the wrappers. The mediators and wrappers of MOMIS cooperate with each other through CORBA. The data model for Amos II [72] is an object-oriented extension of the functional data model DAPLEX. Amos II uses an object-oriented query language called AMOSQL, which provides the select-from-where clause. Amos II also supports distributed query compilation among mediators and translators. The communication among Amos II modules is through TCP/IP sockets.

The Infomaster [22] system represents both global and local schemas as relations, which are called interface and site relations respectively. It then introduces a virtual middle layer called base relations, which express the relationships between interface and site relations. Base relations are intended to be "the basic building blocks of the application domain at hand," so that it is easy to add new data sources and adapt to changes in the existing data sources. Both interface and site relations are views defined on base relations. User queries are posted on interface relations. User queries are unfolded by substituting interface relations with their view definitions. This step is called reduction. The next step is called abduction, which uses a standard model elimination theorem prover to translate queries on base relations to queries on site relations. The queries on site relations are what others refer to as atomic queries. These queries are executed after they have been optimized. The query language used in the Infomaster system is a first order logic language called knowledge interchange format (KIF), which has Lisp-like syntax. The relations are described in KIF also.

As previously stated, Marian and Siméon's work on document projection [52] is particularly relevant to the research described in this dissertation. The authors developed a special projection algorithm for XML documents that enable memory-based XQuery engines to process significantly larger documents [52]. This work has been cited by several other researchers. For example, XMChecker [1] is a system that applies a model checking technique to evaluate XPath queries. XStreamQuery [24] is an event-based XQuery interpreter based on SAX pipelines. FluX [45, 44] extends XQuery by making use of ordering constraints from DTDs, which requires less buffering. FluX reduces main memory consumption and facilitates event-based query processing on structured data streams. In another project, the XSQuirrel system [74] proposed a new language that facilitates queries to manipulate sub-documents, which are parts of original documents. XSQuirrel has the same syntax as XPath, although it is semantically different. A path in XPath selects the nodes at the end of the path and all its descendants, while a path in XSQuirrel is defined as the ancestors of the node at the end of a path, in addition to all those selected by XPath. A path in XSQuirrel actually represents a subtree beginning at the root of the document. Sub-documents are parts of documents selected by subtrees from the root to its leaves. The set of all the sub-documents of a original document is a subset of the set of all the projection documents of that original document. If a set of projection paths all end with #, then the union of those projection paths without # is a legal XSQuirrel query that returns, not surprisingly, the corresponding projection document. A small change to the loading algorithm of Marian and Siméon will be a implementation of the main part of XSQuirrel. The small change is to treat every XSQuirrel path as a projection path having # at the end.

1.7. Outline

The rest of this thesis is organized as follows. The next chapter introduces the basic system architecture and presents the different units that constitute the Omphalos mediation system. The metadata, wrappers and mediators are discussed. A simple information integration example scenario in Section 2.3 is used to explain the details of the Omphalos system. A collaborative schema matching tool is introduced in Section 2.4. Chapter 3 presents a description of the extended XML document projection algorithm, which is used as the query decomposition algorithm. The concepts and notations are introduced in the first few sections. Then the algorithm is presented as inference rules, following the syntax of an XQuery core subset. Finally, a proof of the soundness of the algorithm is presented. Chapter 4 is an evaluation of the algorithm. Both my algorithm and the original projection algorithm are applied on the evaluation queries that are typical to the Omphalos mediation system. Then the outcome of the comparison is presented. Chapter 5 concludes this thesis with a discussion of its contributions and future work. The appendices include some materials related to this thesis.

CHAPTER 2

THE OMPHALOS INFORMATION INTEGRATION SYSTEM

The two major contributions of this work are 1) its use of XQuery to represent mappings between the global schema and individual data sources, and 2) an extended XML document projection algorithm that identifies a subset of the source data needed to answer a user query on the global schema. By retrieving only a subset of the data, the query process is able to execute more efficiently than if all the data were retrieved. The next section provides a brief overview of the mediator system used in this dissertation. A more detailed description of the two contributions along with an example then follows.

2.1. Overview

For this research, a mediation approach was used to integrate data, because it can provide up to date information and scales well. Moreover, XML and Java related technologies were used to implement the mediation system. XML related technologies [84, 85, 87, 86] provide a platform independent data representation, a de facto data exchange standard, and a powerful query language (XQuery). Furthermore, by using XML technologies, this research is able to develop an extended XML document projection algorithm to reduce the amount of data retrieved from the real data sources to answer a user query.

The basic architecture of the system is presented in Figure 2.1 on page 13 and is similar to that developed in [38, 78]. A general mediation system consists of mediators, wrappers and metadata. A wrapper hides the heterogeneity of a data source and presents “a common information model” to a mediator [15]. A mediator accepts user queries and decomposes them into atomic and integrating queries [31]. An atomic query is one that can be answered by a single data source. The mediator then passes atomic queries to the corresponding

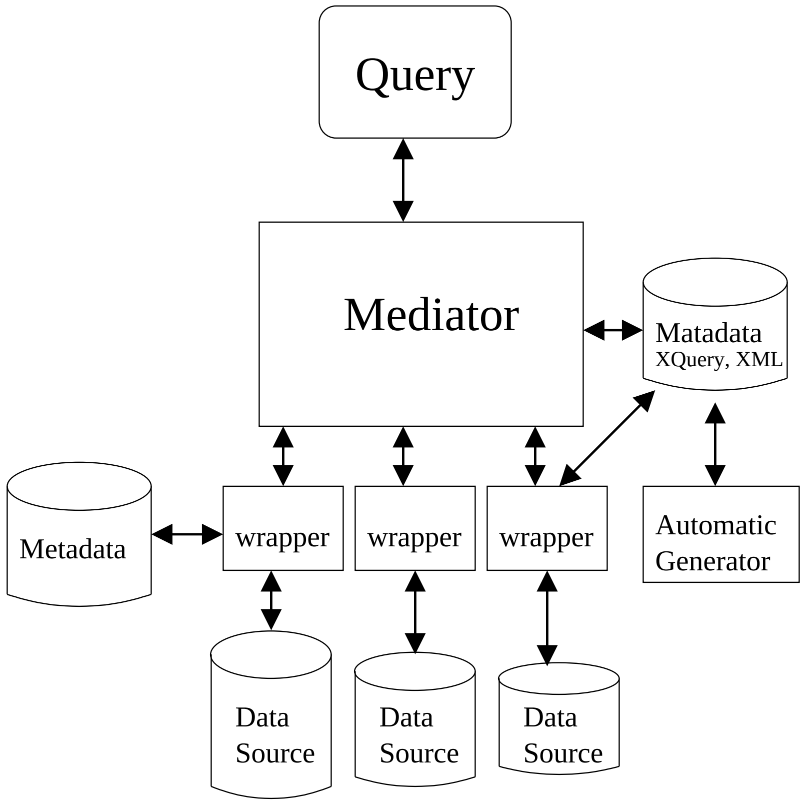


Figure 2.1. Mediation Architecture

wrappers for each of the relevant data sources. Each wrapper executes the atomic query against its associated data source and returns the results to the mediator. The mediator then runs the integrating query against the data collected from the wrappers and computes the final answer and passes that back to the user.

Query processing is represented in Figure 2.2 on page 14. A user query is parsed and normalized into a core XQuery query [88]. The core XQuery query is then decomposed into atomic and integrating queries [31]. Each relevant wrapper then runs its atomic query on its data and produces an XML stream. Finally, the mediator runs the integrating query on those XML streams and generates the answer.

The prototype system developed for this dissertation uses XML related technologies to wrap different data sources as well as represent the mappings among different data sources. XML [84] is a very flexible format for data representation, data storage, and data exchange.

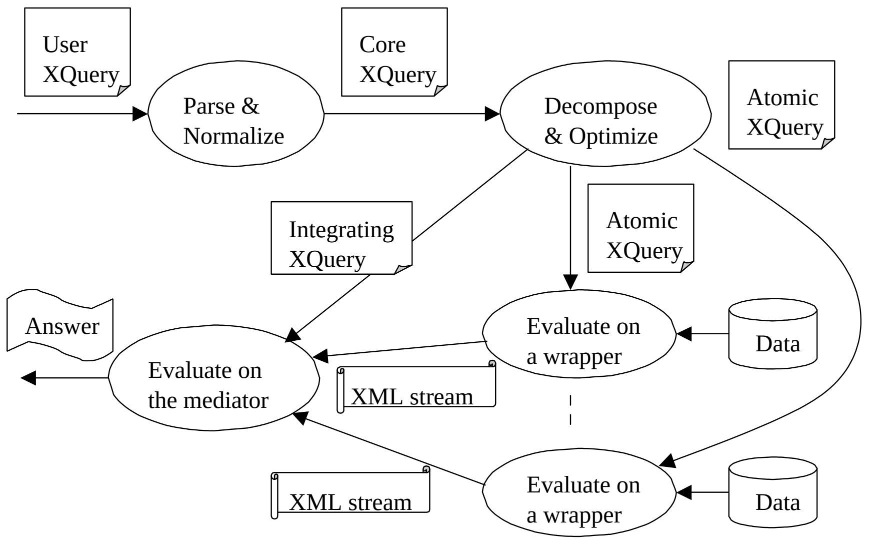


Figure 2.2. Query Processing

XML Schema [85] supports rich data types and other constraints in order to restrict XML document content. XQuery [87, 88, 86] is a powerful language to search and transform XML data. Thus, XQuery expressions are used to express the global data as views on local data. These expressions are able to capture very complex transformations that reconcile discrepancies between different schemas. The mediators and wrappers in the prototype system currently use Saxon-B XQuery engine developed by Michael Kay to execute atomic queries and integrating queries. They accept XQuery queries or sets of XPath paths and return responses in XML.

2.2. Creating an Information Integration System

This section describes the steps that are necessary to create an information integration system using the proposed approach. In order to create an information integration system, the data integrator must first identify the various data sources that are to be integrated as well as the schemas that describe these data sources. The next step is to define a global schema that represents the data items that can be queried. These data items may be defined directly or constructed from the real data sources. These two steps are then followed by a third step, which is to decide how the data items from the real data sources relate to the data items in the global schema. The results of these decisions are represented as mappings between the global schema and the data source schemas.

These initial steps are, for the most part, generic and not unique to any particular approach. However, this dissertation suggests that the mappings developed in the third step to be expressed as XQuery expressions. To facilitate this process, it was necessary to develop a graphical schema matching tool. This tool presents XML Schemas in a way that allows a user to indicate the mappings between the schemas by simply clicking on the selected entries. XQuery expressions can be entered by the user to represent the mappings. This implies that the global schema and the data source schemas are described using XML Schema. The data sources are wrapped and exported as XML documents, even though they may be in some other format. Expressing all components of the system as XML or XQuery allows the information integration system to use the extended XML document projection algorithm (described later in this dissertation).

After wrapping all data sources as XML sources, the next step is to combine the various XQuery mapping expressions into an XQuery query that can produce the global database from the individual data sources, i.e. define the global database as a view on the data sources. This query is referred to as a metaquery. Currently this metaquery must be coded manually. The information integrator must write the metaquery using the mapping expressions and the schemas. This metaquery is then placed into the metadata file. There should be a metaquery in the metadata file for each global XML document that the integrator wants to make available to users.

Whenever a user query is received by the system, it is combined with the metaquery to form what this research calls an *outsourced query.* This combination is necessary because the user query refers to the global database, which is a virtual database and does not exist physically. Therefore, the references to the global database are replaced by the metaqueries that can generate the global database from the real data sources. The creation of the outsourced query is automatic. The outsourced query could produce an answer to the original user query, but it would be very inefficient, because it would mean that the entire contents of data sources would have to be transmitted. For most queries, only a small subset of the data within data sources is needed to answer the query. The approach that was used in this dissertation was to use an extended XML document projection algorithm (describe below) to retrieve only the appropriate subset of data from the real data sources.

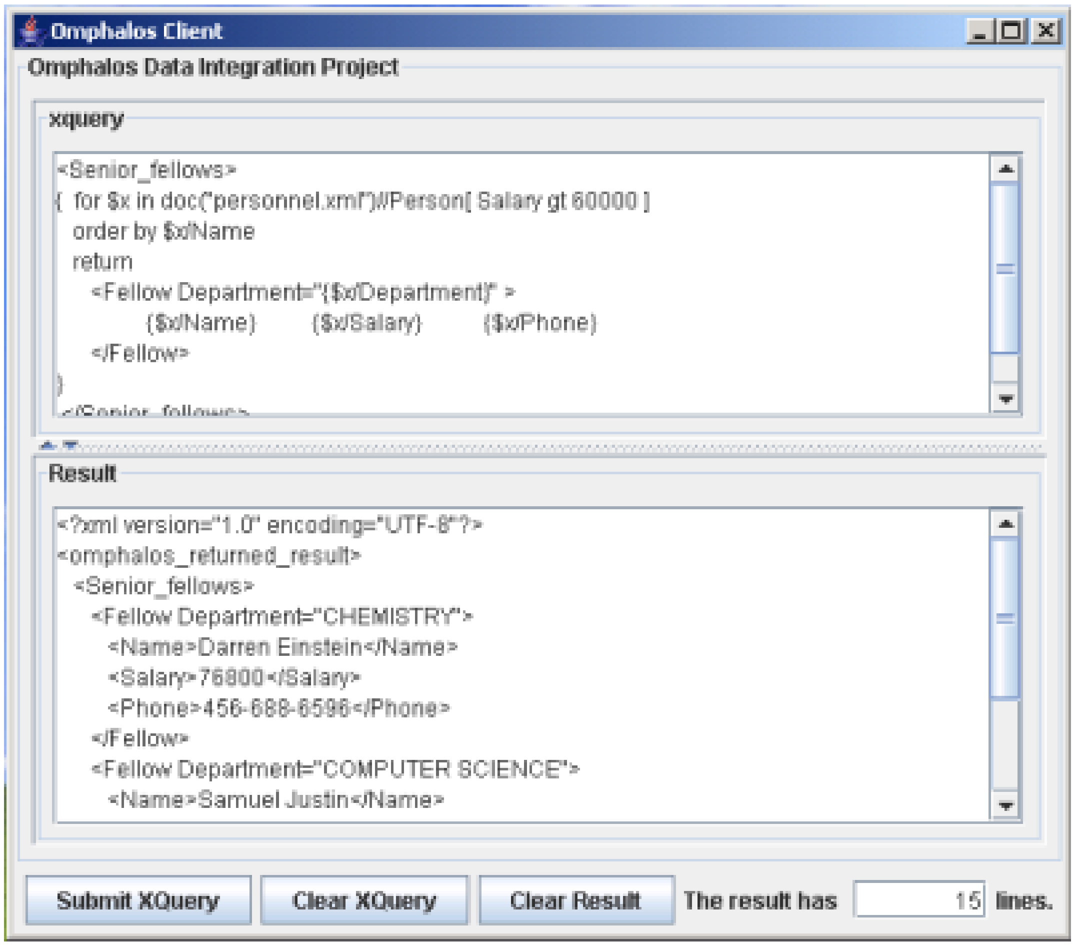
Therefore, the next step is to decompose the outsourced query into atomic queries, which may be answered by a single data source. The extended XML document projection algorithm reads the outsourced query and uses the inference rules from the algorithm to produce the atomic queries in such a way that they retrieve only the data that is required to answer the user query. Details of the algorithm and the proof of its correctness are given in the next chapter. Once the data is retrieved, the mediator then integrates the results from the atomic queries into an answer and presents that to the user.

The following sections describe a more detailed example of how the system uses XML technologies to integrate disparate data sources.

2.3. An Example Scenario

In order to demonstrate the effectiveness of using XQuery for the mappings and the correctness of the extended projection algorithm, an information integration system called Omphalos [64, 65] was created. Figure2.3 on page 17 shows a client accessing the Omphalos information integration system. The top window shows a query on the global database, and the bottom window shows the result of the query.

The Omphalos information integration system was populated with three data sources, which are used as examples throughout this dissertation. In this simplified version of an integration system, every data source has just one resource of interest. A real world information integration application usually has many more data sources and many more resources per



```
<Senior_fellows>
{ for $x in doc("personnel.xml")//Person[ Salary gt 60000 ]
  order by $x/Name
  return
    <Fellow Department="{$x/Department}" >
          {$x/Name}        {$x/Salary}        {$x/Phone}
    </Fellow>
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<omphalos_returned_result>
  <Senior_fellows>
    <Fellow Department="CHEMISTRY">
      <Name>Darren Einstein</Name>
      <Salary>76800</Salary>
      <Phone>456-688-6596</Phone>
    </Fellow>
    <Fellow Department="COMPUTER SCIENCE">
      <Name>Samuel Justin</Name>
```

Figure 2.3. Omphalos Information Integration System

data source; however, for the purpose of verifying the ideas presented in this dissertation, a simplified version serves equally well.

2.3.1. The Data Sources

The three data sources used by the system contain different forms of information items associated with a college and its chemistry department, and its computer science department. The college has a directory listing as an XML document named dir.xml (see Listing 2.1). The chemistry department has a relation called lecturers on an Oracle database server (see table 2.1). The computer science department has a relation called faculty on a MySQL database server (see table2.2). All three data sources are independently managed. For the convenience of the responsible personnel in the college, the Omphalos mediation system integrates these

Name	Rank	Start_year	Room	Pay_rate	Monthly_hours
Mark Russell	Assistant Professor	1995	CH235	38	120
Logan Nixon	Assistant Professor	1990	CH236	40	110
Darren Einstein	Full Professor	1976	CH238	80	80

Table 2.1. Oracle Database Table: lecturers

three data sources so that they can search for information of interest from all data sources. The system presents an integrated view of these three data sources to users as a virtual database that has a single virtual document named personnel.xml (see Listing 2.2).

Listing 2.1. Abridged dir.xml (see unabridged version in Appendix A)

```
<?xml version="1.0" encoding="UTF-8" ?>
<Directory>
   <Person Name="Frank Kerry">
      <Phone>456-677-8007</Phone>
      <Email>frank_kerry@unt.edu</Email>
      <Address>
         <Street>560 Bryan St</Street>
         <City>Denton</City>
         <State>Texas</State>
      </Address>
   </Person>
   <Person Name="Daisy Henry">
      <Phone>456-666-6591</Phone>
      <Email>daisy_henry@unt.edu</Email>
      <Address>
         <Street>1400 Caroll Blvd</Street>
         <City>Denton</City>
         <State>Texas</State>
      </Address>
   </Person>
</Directory>
```

FName	LName	Office	Title	Salary	Web_site
Daisy	Henry	CS123	Assistant Professor	56000	http://www.csci.unt.edu/~henry
Samuel	Justin	CS122	Full Professor	70000	http://www.csci.unt.edu/~justin
Frank	Kerry	CS111	Assistant Professor	50000	http://www.csci.unt.edu/~kerry

Table 2.2. MySQL Database Table: faculty

2.3.2. The Global Database Schema

The document personnel.xml (see Listing 2.2) does not actually exist. What really exists is its W3C XML Schema instance personnel.xsd (see Appendix B), which is included in the global database schema.

The global schema has items that are transferred directly from the data sources, such as the “Name” field which maps to the “Name” field in the lecturers table. It also has items that are generated from two or more items in the data sources, such as the “Name” field in the global database which also maps to the concatenation of the “FName” and the “LName” fields in the faculty table. A data item must be generated whenever that data item is not found in the original data sources. Generated data items occur frequently in real-world database applications. An example of generated data items is the department name, where “Chemistry” is created as the department name for people in the lecturers table and “Computer Science” is created as the department name for people in the faculty table. The objects in the global database are composed of data that are combined from the directory and the lecturers table, or from the directory and the faculty table using the name as a match key. Now that the global database schema and the data sources are identified, the mappings between them must be discovered and defined.

2.4. Collaborative Schema Matching Tool

One of the major tasks of data integration and many other database applications is to identify the matching elements among schemas. Data integrators generally rely on assistance

Listing 2.2. personnel.xml Presented by the Mediator

```
<Personnel xmlns="..."  xmlns:xsi="..."
                        xsi:schemaLocation="..." >
    <Person>
         <Name>Samuel Justin</Name>
         <Department>COMPUTER SCIENCE</Department>
         <Salary>70000</Salary>
         <Rank>Full Professor</Rank>
         <Office>CS122</Office>
         <Contacts>
            <Phone>456-666-6592</Phone>
            <Email>samuel_justin@unt.edu</Email>
         </Contacts>
         <Address>
            <Street>360 La vista St</Street>
            <City>Dallas</City>
            <State>Texas</State>
         </Address>
     </Person>
     ...
</Personnel>
```

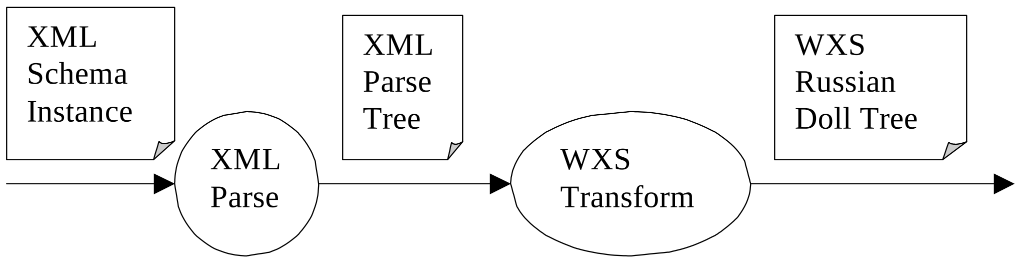
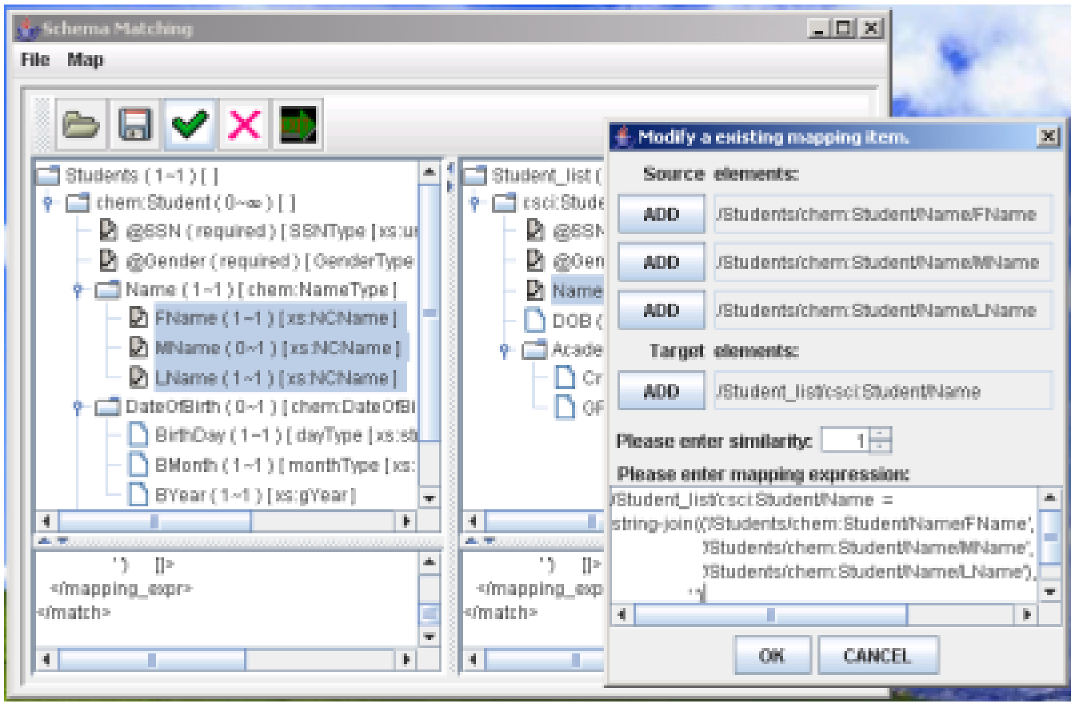


Figure 2.4. W3C XML Schema Transformation

from domain experts to accomplish schema-matching tasks [19]. This is also true in Omphalos, but it also offers a special tool to support the schema matching process. The tool is able to show schemas in graphic form to facilitate the manual matching process.

The Figure 2.4 shows the process by which a W3C XML Schema instance is transformed into a tree representation that can be more easily visualized. This tree is called a Russian Doll tree since all the references have been replaced with nested structures. The Figure 2.5 shows the GUI interface of the Omphalos collaborative schema matching tool. The right



```
Students ( 1~1 ) [ ]
  chem:Student ( 0~∞ ) [ ]
    @SSN ( required ) [ SSNType [ xs:u
    @Gender ( required ) [ GenderType
    Name ( 1~1 ) [ chem:NameType ]
      FName ( 1~1 ) [ xs:NCName ]
      MName ( 0~1 ) [ xs:NCName ]
      LName ( 1~1 ) [ xs:NCName ]
    DateOfBirth ( 0~1 ) [ chem:DateOfBi
      BirthDay ( 1~1 ) [ dayType [ xs:st
      BMonth ( 1~1 ) [ monthType [ xs:
      BYear ( 1~1 ) [ xs:gYear ]
```

```
Source elements:
/Students/chem:Student/Name/FName
/Students/chem:Student/Name/MName
/Students/chem:Student/Name/LName
Target elements:
/Student_list/csci:Student/Name
Please enter similarity: 1
Please enter mapping expression:
/Student_list/csci:Student/Name =
string-join(('/Students/chem:Student/Name/FName',
             '/Students/chem:Student/Name/MName',
             '/Students/chem:Student/Name/LName'),
          ' ')
```

Figure 2.5. Collaborative Schema Matching

panel shows a view of the global schema as target schema, and the left panel shows a view of a local schema as source schema. A user indicates the mappings between schema elements by simply clicking on the matched items. The mappings can be in the form of many to many relationship and can be associated with mapping expressions. The tool will produce schema matching files encoded in XML (see Listing 2.3 for an example), which are then used by the information integration implementer to write the appropriate metaquery.

Listing 2.4 is the Russian Doll tree for the W3C XML Schema instance personnel.xsd. It is obviously much easier for the user to understand this representation as opposed to the original schema, since it shows the nested structures directly while hiding any details.

What makes the Omphalos schema matching tool special is that it is a collaborative tool. It can run as a plug-in of ICE [6, 79], which is a platform independent collaborative environment developed by our lab. Users can run the tool as plug-in in the ICE on different machines that are connected by networks. The tool will communicate schema-matching information between the machines. When a user creates a match at one machine, other

Listing 2.3. Schema Matching Output File

```
<?xml version="1.0" encoding="UTF-8"?>
<schema_match>
  <source name="faculty.xsd" ID="1" />
  <target name="personnel.xsd" ID="1" />
  <match ID="1" similarity="1.0" >
     <source_element sourceID="1" xpath="/Faculty/Row/FName" />
     <source_element sourceID="1" xpath="/Faculty/Row/LName" />
     <target_element targetID="1" xpath="/Personnel/Person/Name" />
     <mapping_expr><![CDATA[
/Personnel/Person/Name =
string-join((/Faculty/Row/FName,
                    /Faculty/Row/LName), " ")  ]]>
     </mapping_expr>
  </match>
  <match ...>
  ...
  </match>
  ...
</schema_match>
```

Listing 2.4. The Russian Doll Tree for personnel.xsd

```
Personnel(1~1)
  Person (0~∞)
    Name(1~1)[xs:token]
    Department(1~1)[DepartmentType[xs:token]]
    Salary(1~1)[NonNegativeDecimal[xs:decimal]]
    Rank(1~1)[RankType[xs:token]]
    Office(1~1)[xs:token]
    Contacts(1~1)[ContactsType]
      Phone(1~1)[PhoneNumberType[xs:string]]
      Email(1~1)[EmailType[xs:token]]
    Address(1~1)[AddressType]
      Street(1~1)[xs:token]
      City(1~1)[xs:token]
      State(1~1)[xs:NMTOKEN]
```

users will see the match instantly at their machines. This way, the tool helps experts who are separated by distance.

To provide a uniform interface between the mediators and the data sources, every non-XML data source is wrapped as an XML data source. Wrappers use the W3C XML Schema to export XML data from data sources to the mediator. That is, the wrappers hide the different formats so that they can present uniform data to the mediator.

Listing 2.5. lecturers.xml Presented by the Chemistry Department Wrapper

```
<?xml version="1.0" encoding="UTF-8" ?>
<Lecturers>
  <Row>
    <Name>Mark Russell</Name>
    <Rank>Assistant Professor</Rank>
    <Room>CH235</Room>
    <Pay_rate>38</Pay_rate>
    <Monthly_hours>120</Monthly_hours>
  </Row>
  <Row>
    <Name>Logan Nixon</Name>
    <Rank>Assistant Professor</Rank>
    <Room>CH236</Room>
    <Pay_rate>40</Pay_rate>
    <Monthly_hours>110</Monthly_hours>
  </Row>
  <Row>
    <Name>Darren Einstein</Name>
    <Rank>Full Professor</Rank>
    <Room>CH238</Room>
    <Pay_rate>80</Pay_rate>
    <Monthly_hours>80</Monthly_hours>
  </Row>
</Lecturers>
```

For the example scenario, the tables in the databases are trivially wrapped. The chemistry department wrapper presents its data as lecturers.xml (see Listing 2.5); The computer science department wrapper presents its data as faculty.xml (see Listing 2.6); The college wrapper simply presents its data as is (see Listing 2.1 or Appendix A). Note that wrappers do not export any local data that is not of interest to the user. Those local data have no correspondences

Listing 2.6. faculty.xml Presented by the Computer Science Department Wrapper

```
<?xml version="1.0" encoding="UTF-8" ?>
<Faculty>
  <Row>
    <FName>Daisy</FName>
    <LName>Henry</LName>
    <Office>CS123</Office>
    <Title>Assistant Professor</Title>
    <Salary>56000</Salary>
  </Row>
  <Row>
    <FName>Samuel</FName>
    <LName>Justin</LName>
    <Office>CS122</Office>
    <Title>Full Professor</Title>
    <Salary>70000</Salary>
  </Row>
  <Row>
    <FName>Frank</FName>
    <LName>Kerry</LName>
    <Office>CS111</Office>
    <Title>Assistant Professor</Title>
    <Salary>50000</Salary>
  </Row>
</Faculty>
```

in the global documents. For example, the computer science department wrapper does not export its faculty websites.

Therefore, the mappings between the global schema and the schemas of the data sources are expressed as XQuery expressions, even though the data sources themselves are not necessarily XML files.

2.6. Metaquery

Mediators present integrated data that users can query. The integrated data include all data in the data sources that is of interest to the user. The example system has one mediator. It presents the data in Listing 2.2 to users as personnel.xml. This document is never actually generated. What really exists is the W3C XML Schema personnel.xsd (see Appendix B), which is stored in the metadata. The mediator acts as if it had an XML document called personnel.xml, which combines data from all three data sources. In order to access data from the data sources, the mediator must create the converted query (see Listing 2.8). This query is called a metaquery and is written by the information integrator using the XQuery expressions that represent the mappings between the global and local schemas. This metaquery is stored in the metadata file described in the next section.

2.7. Metadata

The integrated view presented by Omphalos' mediator is perceived by users as an XML database that consists of XML documents. These XML documents are virtual, however, since they are not actually real.

To answer user queries, the mediator needs to determine how to extract relevant data from local data sources and process them, i.e. decompose user queries into atomic and integrating queries. The mediator accomplishes the task by consulting the metadata.

Omphalos' metadata includes the following four parts:

(1) The mediated schema, i.e. the global schema (personnel.xsd)

(2) The schemas of local data sources (lecturers.xsd, faculty.xsd, and directory.xsd)

(3) The mappings between the global schema and the local schemas (the metaqueries)

(4) The locations of the wrappers (the host names and the TCP port numbers)

The mediated schema consists of a W3C XML Schema instance for each virtual XML document. The schemas of data sources include all W3C XML Schema instances exported by the wrappers. A local data source is not necessarily an XML database or repository because its wrapper hides the source's representation by wrapping it as an XML source.

Omphalos uses the Global As View (GAV) approach [48]. The XQuery metaqueries in the metadata express ways to generate virtual global XML documents from local data sources. The metaqueries capture all the complex data extractions, transformations, and restructuring that need to take place. There is one XQuery metaquery for each virtual global

XML document. These XQuery metaqueries collectively define the virtual global database as a view on local data sources. The mediator uses these XQuery metaqueries to convert user queries over the global database to queries over the data sources.

2.7.1. DTD for the Metadata File

The Omphalos metadata is represented by XML documents. The global and local schemas consist of W3C XML Schema instances and thus are already XML documents. Listing 2.7 presents the DTD for the metadata file, which constitutes the third and fourth parts of the metadata. An instance of the metadata file is shown in Listing 2.8.

Listing 2.7. DTD for the Metadata File config.dtd

```
<?xml version="1.0" encoding="UTF-8" ?>

<!ELEMENT om:mediator ( om:data-sources, om:global-as-views ) >
<!ATTLIST om:mediator xmlns:om CDATA #FIXED
   "http://mars.csci.unt.edu/dbgroup/omphalos/mediator/config/" >

<!ELEMENT om:data-sources ( om:source+ )>

<!ELEMENT om:source ( om:resource+ )>
<!ATTLIST om:source host CDATA #REQUIRED
                    port CDATA #REQUIRED >

<!ELEMENT om:resource EMPTY>
<!ATTLIST om:resource name CDATA #REQUIRED >

<!ELEMENT om:global-as-views ( om:gav+ ) >

<!ELEMENT om:gav ( #PCDATA )>
<!ATTLIST om:gav name CDATA #REQUIRED >
```

The root element is *om:mediator*. It includes one om:data-source element and one om:global-as-view element. An *om:data-sources* element includes one or more om:source elements. Every *om:source* element has information about one wrapped data source: what *om:resource* (i.e. local XML documents) the wrapper has exported and how to communicate with this wrapper. A *om:global-as-view* element includes one or more *om:gav* elements. Every *om:gav* element specifies how to generate one global XML document from relevant local XML documents.

2.7.2. An Instance of the Metadata File

Listing 2.8 shows the metadata file of the Omphalos mediator that integrates the three data sources in the example system (see Section 2.3). Each data source is wrapped as having just one resource, i.e. a local XML document. The first data source resides on the host mars.csci.unt.edu and is wrapped as having dir.xml. The second resides on last.csci.unt.edu and is wrapped as having lecturers.xml. The third is located on poseidon.csci.unt.edu and is wrapped as having faculty.xml. The actual characteristics of the data are not obvious by looking at this metadata file. For example, faculty.xml is actually generated by the wrapper from a table in a MySQL database, while lecturers.xml is generated from a table in an Oracle database. All three wrappers provide services to the mediator through TCP port 8889.

This mediator has just one global document personnel.xml, which can be queried by users. This document is virtual. Only its W3C XML Schema instance personnel.xsd really exists, as previously discussed in Subsection 2.3.2. An XQuery expression (metaquery) is included in the metadata to express the mapping between the global schema and the three local schemas exported by the wrappers.

Listing 2.8. An Instance of the Metadata File config.xml

```
<?xml version='1.0' encoding="UTF-8" ?>
<!DOCTYPE om:mediator SYSTEM "config.dtd">
<om:mediator xmlns:om=
    "http://mars.csci.unt.edu/dbgroup/omphalos/mediator/config/">
  <om:data-sources>
    <om:source host="mars.csci.unt.edu" port="8889" >
      <om:resource name="dir.xml" />
    </om:source>
    <om:source host="last.csci.unt.edu" port="8889" >
      <om:resource name="lecturers.xml" />
    </om:source>
    <om:source host="poseidon.csci.unt.edu" port="8889" >
      <om:resource name="faculty.xml" />
    </om:source>
  </om:data-sources>
  <om:global-as-views>
    <om:gav name="personnel.xml" >
      <![CDATA[
document {
<Personnel>
{ (
for $x in doc("lecturers.xml")/Lecturers/Row,
    $y in doc("dir.xml")/Directory/Person
where $x/Name = $y/@Name
return
  <Person>
    {$x/Name}
    <Department>CHEMISTRY</Department>
    <Salary> {($x/Pay_rate) * ($x/Monthly_hours) * 12}
    </Salary>
    {$x/Rank}
    <Office>{$x/Room/text()}</Office>
    <Contacts>
      {$y/Phone}
      {$y/Email}
    </Contacts>
    {$y/Address}
  </Person> )
union (
for $x in doc("faculty.xml")/Faculty/Row,
    $y in doc("dir.xml")/Directory/Person
let $xname := string-join(($x/FName, $x/LName), " ")
where $xname = $y/@Name
return
  <Person>
    <Name>{$xname}</Name>
    <Department>COMPUTER SCIENCE</Department>
    {$x/Salary}
    <Rank>{$x/Title/text()}</Rank>
    {$x/Office}
    <Contacts>
      {$y/Phone}
      {$y/Email}
    </Contacts>
    {$y/Address}
  </Person> )
}
</Personnel>
}
      ]]>
    </om:gav>
  </om:global-as-views>
</om:mediator>
```

2.8. Omphalos's Solution to the Example User Query

2.8.1. The Example User Query over the Global Document

For this example, Omphalos has just one global document called personnel.xml. The user queries the following:

for each person in the college with an income over $60000, list his or her full name, salary and phone number inside a Fellow element. Enclose all the fellow elements inside a High_income element.

The corresponding XQuery query is shown in Listing 2.9.

Listing 2.9. The User XQuery Query over Global Virtual Database

```
<High_income>
{
  for $x in doc("personnel.xml")/Personnel/Person[Salary > 60000]
  return
    <Fellow Department="{$x/Department}" >
      {$x/Name}
      {$x/Salary}
      {$x/Contacts/Phone}
    </Fellow>
}
</High_income>
```

2.8.2. The Example Outsourced Query over the Local Documents

Using the metadata, the mediator transforms the user query over the virtual global database into a query over local data sources (*outsourced query*) as shown in Listing 2.10. Basically, the transformation unfolds the global document references in the query using their corresponding metaqueries from the metadata. In other words, the transformation replaces the global document references with their corresponding mappings.

Listing 2.10. The Outsourced Query

```
<High_income>
{
let $z := document {
<Personnel>
{ (
for $x in doc("lecturers.xml")/Lecturers/Row,
    $y in doc("dir.xml")/Directory/Person
where $x/Name = $y/@Name
return
  <Person>
    {$x/Name}
    <Department>CHEMISTRY</Department>
    <Salary> {($x/Pay_rate) * ($x/Monthly_hours) * 12}
    </Salary>
    {$x/Rank}
    <Office>{$x/Room/text()}</Office>
    <Contacts>
      {$y/Phone}
      {$y/Email}
    </Contacts>
    {$y/Address}
  </Person> )
union (
for $x in doc("faculty.xml")/Faculty/Row,
    $y in doc("dir.xml")/Directory/Person
let $xname := string-join(($x/FName, $x/LName), "␣")
where $xname = $y/@Name
return
  <Person>
    <Name>{$xname}</Name>
    <Department>COMPUTER SCIENCE</Department>
    {$x/Salary}
    <Rank>{$x/Title/text()}</Rank>
    {$x/Office}
    <Contacts>
      {$y/Phone}
      {$y/Email}
    </Contacts>
    {$y/Address}
  </Person> )
}
</Personnel>
}
for $x in $z/Personnel/Person[ Salary gt 60000 ]
return
  <Fellow Department="{$x/Department}" >
    {$x/Name}
    {$x/Salary}
    {$x/Contacts/Phone}
  </Fellow>
}
</High_income>
```

Figure 2.6 on page 32 illustrates the query decomposition process in Omphalos. A user XQuery query is unfolded into an outsourced XQuery query, which is normalized into an XQuery core query. Then the outsourced XQuery core query goes through the extended projection analysis which generates projection paths. From the outsourced XQuery core query and its projection paths, Omphalos produces the integrating query (IQ). The projection paths are converted to or used directly as atomic queries (AQ's). Generally, there can be multiple decompositions for any given outsource query. Each decomposition has its advantages and disadvantages. Omphalos's decomposition places a low requirement on the capabilities of the wrappers when projection paths are used directly as atomic queries. The wrappers do not need to be full-fledged XQuery engines, but only need to be able to process XPath paths. Another advantage of using projection paths directly is that they can be stream processed. If necessary, a set of projection paths can be converted to a single XQuery query using the algorithm presented by Arnaud Sahuguet and Bogdan Alexe [74].

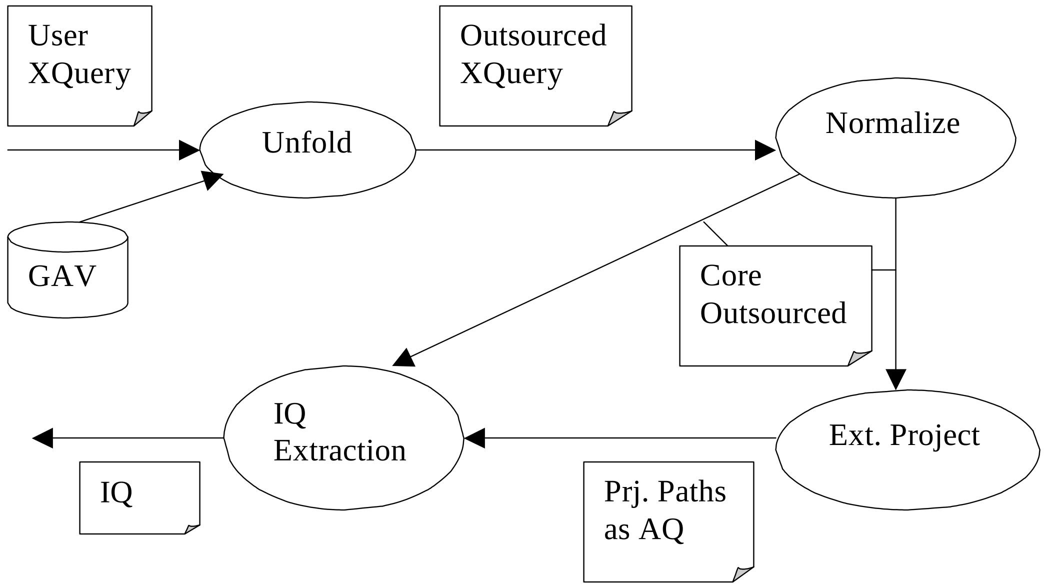


Figure 2.6. Query Decomposition

The next four subsections show the atomic queries and the integrating query produced by Omphalos from the decomposition of the example outsourced query. There are three atomic queries since the outsourced user query needs data from all the three wrappers.

2.8.3. The Atomic Query to the Chemistry Department Wrapper

Given the example query, the relevant data in the chemistry data source are names, pay rates and monthly hours of lecturers. So the atomic query should retrieve only those data. Since the wrapper can accept sets of projection paths, the Omphalos mediator sends the following set of projection paths as the atomic query:

$$\{doc(\text{"lecturers.xml"})/Lecturers/Row\ ,$$

$$doc(\text{"lecturers.xml"})/Lecturers/Row/Name,$$

$$doc(\text{"lecturers.xml"})/Lecturers/Row/Pay\_rate,$$

$$doc(\text{"lecturers.xml"})/Lecturers/Row/Monthly\_hours\}$$

The set of projection paths can be converted to a single XQuery query in Listing 2.11 using an algorithm presented in the paper [74] if the wrapper only accepts an XQuery query as input.

Listing 2.11. Atomic Query Sent to the Chemistry Department Wrapper

```
<Lecturers>
{
  for $x in doc("lecturers.xml")/Lecturers/Row
  return
    <Row>
      {$x/Name}
      {$x/Pay_rate}
      {$x/Monthly_hours}
    </Row>
}
</Lecturers>
```

2.8.4. The Atomic Query to the Computer Science Department Wrapper

Again referring to the example query, the relevant data in the computer Science data source are first names, last names and salaries of faculty. So the atomic query should only retrieve those data. Therefore, the Omphalos mediator just sends the following set of projection paths as the atomic query:

$$\{doc(\text{"faculty.xml"})/Faculty/Row\,,$$
$$doc(\text{"faculty.xml"})/Faculty/Row/FName,$$
$$doc(\text{"faculty.xml"})/Faculty/Row/LName,$$
$$doc(\text{"faculty.xml"})/Faculty/Row/Salary\}$$

The Omphalos mediator can send the XQuery query in Listing 2.12 to the wrapper if it only accepts an XQuery query as input.

Listing 2.12. Atomic Query Sent to the Computer Department Wrapper

```
<Faculty>
{
  for $x in doc("faculty.xml")/Faculty/Row[Salary gt 60000]
    return
      <Row>
        {$x/$FName}
        {$x/$LName}
        {$x/Salary}
      </Row>
}
</Faculty>
```

2.8.5. The Atomic Query to the College Wrapper

The relevant data for the example query in the college data source are names and phone numbers. The Omphalos mediator sends the following set of projection paths as atomic query:

$$\begin{aligned}\{&doc(\text{"dir.xml"})/Directory/Person\,,\\ &doc(\text{"dir.xml"})/Directory/Person/Name\,,\\ &doc(\text{"dir.xml"})/Directory/Person/Phone\}\end{aligned}$$

The Omphalos mediator can send the XQuery query in Listing 2.13 to the wrapper if it only accepts an XQuery query as input.

Listing 2.13. Atomic Query Sent to the College Wrapper

```
<Directory>
{
  for $x in doc("dir.xml")/Directory/Person
  return
    <Person>
      {$x/@Name}
      {$x/Phone}
    </Person>
}
```

```
</Directory>
```

2.8.6. The Integrating Query

The Listing 2.14 shows the integrating query that Omphalos generates. The external variables $chemStream, $csciStream and $collStream in Listing 2.14 represent the three XML streams returned by the three wrappers. Note that the Omphalos decomposition eliminates the unnecessary construction of the contents of the following elements: Rank, Office, Email and Address.

Listing 2.14. The Integrating Query

```
declare variable $chemStream external;
declare variable $csciStream external;
declare variable $collStream external;
<High_income>
{
let $z := document {
<Personnel>
{ (
for $x in doc("lecturers.xml")/Lecturers/Row,
    $y in doc("dir.xml")/Directory/Person
where $x/Name = $y/@Name
return
  <Person>
    {$x/Name}
    <Department>CHEMISTRY</Department>
    <Salary> {($x/Pay_rate) * ($x/Monthly_hours) * 12}
    </Salary>
    <Office />
    <Contacts>
      {$y/Phone}
    </Contacts>
  </Person> )
union (
for $x in doc("faculty.xml")/Faculty/Row,
    $y in doc("dir.xml")/Directory/Person
let $xname := string-join(($x/FName, $x/LName), "␣")
where $xname = $y/@Name
return
```

```
  <Person>
    <Name>{$xname}</Name>
    <Department>COMPUTER SCIENCE</Department>
    {$x/Salary}
    <Rank />
    <Contacts>
      {$y/Phone}
    </Contacts>
  </Person> )
}
</Personnel>
}
for $x in $z/Personnel/Person[ Salary gt 60000 ]
return
  <Fellow Department="{$x/Department}" >
    {$x/Name}
    {$x/Salary}
    {$x/Contacts/Phone}
  </Fellow>
}
</High_income>
```

2.8.7. Executing the Atomic Queries

The atomic queries are then sent to their corresponding wrappers, and the resulted data streams are returned to the mediator.

The chemistry department wrapper returns the XML stream shown in Listing 2.15 to the mediator.

Listing 2.15. The XML Stream Returned by the Chemistry Department Wrapper

```
<?xml version="1.0" encoding="UTF-8" ?>
<Lecturers>
  <Row>
    <Name>Mark Russell</Name>
    <Pay_rate>38</Pay_rate>
    <Monthly_hours>120</Monthly_hours>
  </Row>
  <Row>
    <Name>Logan Nixon</Name>
    <Pay_rate>40</Pay_rate>
    <Monthly_hours>110</Monthly_hours>
  </Row>
  <Row>
    <Name>Darren Einstein</Name>
    <Pay_rate>80</Pay_rate>
    <Monthly_hours>80</Monthly_hours>
  </Row>
</Lecturers>
```

The computer department wrapper returns the XML stream shown in Listing 2.16 to the mediator.

Listing 2.16. The XML Stream Returned by the Computer Department Wrapper

```
<?xml version="1.0" encoding="UTF-8" ?>
<Faculty>
  <Row>
    <FName>Daisy</FName>
    <LName>Henry</LName>
    <Salary>56000</Salary>
  </Row>
  <Row>
    <FName>Samuel</FName>
    <LName>Justin</LName>
    <Salary>70000</Salary>
  </Row>
  <Row>
    <FName>Frank</FName>
    <LName>Kerry</LName>
    <Salary>50000</Salary>
  </Row>
</Faculty>
```

The college wrapper returns the XML stream shown in Listing 2.17 to the mediator.

Listing 2.17. The XML Stream Returned by the College Wrapper

```
<?xml version="1.0" encoding="UTF-8" ?>
<Directory>
  <Person Name="Frank␣Kerry">
    <Phone>456-677-8007</Phone>
  </Person>
  <Person Name="Mark␣Russell">
```

```
    <Phone>456-677-8006</Phone>
  </Person>
  <Person Name="Logan␣Nixon">
    <Phone>456-666-6595</Phone>
  </Person>
  <Person Name="Darren␣Einstein">
    <Phone>456-688-6596</Phone>
  </Person>
  <Person Name="Samuel␣Justin">
    <Phone>456-666-6592</Phone>
  </Person>
  <Person Name="Daisy␣Henry">
    <Phone>456-666-6591</Phone>
  </Person>
</Directory>
```

2.8.8. Integrated Results

The mediator executes the integrating query after it receives the XML streams returned by the individual wrappers. It returns the XML stream shown in Listing 2.18 as the answer to the original user query.

Listing 2.18. The XML Stream Returned to the User as the Answer

```
<?xml version="1.0" encoding="UTF-8"?>
<High_income>
  <Fellow Department="CHEMISTRY">
    <Name>Darren Einstein</Name>
    <Salary>76800</Salary>
    <Phone>456-688-6596</Phone>
  </Fellow>
  <Fellow Department="COMPUTER␣SCIENCE">
    <Name>Samuel Justin</Name>
    <Salary>70000</Salary>
    <Phone>456-666-6592</Phone>
  </Fellow>
</High_income>
```

CHAPTER 3

QUERY DECOMPOSITION ALGORITHM

As discussed in Chapter 2, it takes several steps to process user queries on the virtual global data sources and create the atomic queries that operate on individual data sources. First, a user XQuery query is combined with a metaquery from the metadata file to form an outsourced query, as discussed in the previous chapter. XQuery query decomposition then translates the outsourced query into either atomic or integrating queries. The atomic queries are executed immediately by the wrappers, and the results are sent back to the mediators. Integrating queries are executed by mediators that compute the final response based on the results of atomic queries. The query decomposition algorithm that is used in this research is called Omphalos and is based on an extension to the XML document projection algorithm. One of the challenges of any decomposition algorithm is to retrieve the minimum data from the data sources that are required to answer user queries. The algorithm used in this research attempts to meet this challenge. A more detailed description of the extended XML document projection algorithm is presented below.

3.1. Projection Paths

Omphalos uses the definition of *projection paths* that was first developed by Marian and Siméon [52]. Intuitively, the projection paths of an XQuery query are those XPath paths that select only the nodes from the input documents that are relevant to a particular query. The selected portion of the input document is called its *projection document*. To keep things simple, Marian and Siméon restrict the form of projection paths to only forward navigation without predicates. So the syntax for a projection path is a subset of XPath. Note, however, XPath paths with reverse axes can be transformed to reverse-axis-free paths [61]. Thus, this restriction does not appear to result in the loss of the algorithm's generality in any significant way.

The projection path grammar used in this thesis is presented in Listing 3.1. This grammar was adapted from the one described by Marian and Siméon [52]. The ''()|?* characters are meta symbols.

Listing 3.1. The Grammar of Projection Paths

```
ProjectionPath ::= doc‘(’ URL ‘)’ #?
           | doc‘(’ URL ‘)’ / RelativePath #?
           | ε

RelativePath ::= ForwardStep
           | RelativePath / ForwardStep

ForwardStep ::= Axis NodeTest

Axis ::= child::
           | self::
           | descendant::
           | descendant-or-self::
           | attribute::

NodeTest ::= ((NCName | *):)?(NCName | *)
           | node‘(’‘)’
           | text‘(’‘)’
```

A projection path normally starts with an input document and includes a series of forward steps. If the path ends with a "#," it selects the nodes in the subtree that are rooted at its end. If it does not end with a "#," the path does not select that subtree. The special case, "ε" is defined as an empty projection path, which is used in the inference rules of conditional expressions (refer to subsection 3.5.10 on page 70) and `for`-expressions (refer to subsection 3.5.7 on page 60).

To distinguish between the nodes identified with input documents from those that were newly constructed by XQuery queries, the former are called *source nodes,* because input documents are data sources, and the latter are called *constructed nodes* for obvious reasons.

3.2. Environment and Store

3.2.1. Returned Paths, Used Paths and Constructed Nodes

As mentioned previously, the projection paths of an XQuery query select the source nodes that are relevant to a specific query and are of two types. The first type, called *returned projection paths* or simply *returned paths,* selects source nodes that appear directly in the result of a query. The other, called *used projection paths* or simply *used paths,* selects source nodes that contribute indirectly to the result of a query.

For example, consider the XQuery expression in Listing 3.2 that lists the names of all full professors.

Listing 3.2. List the Names of All Full Professors

```
1 let $x := doc("lecturers.xml")/Lecturers/Row
2 return
3       if $x/Rank = "Full␣Professor"
4       then
5            $x/Name
6       else ( )
```

The source nodes selected by the projection path $x/Name on line 5 , i.e.,

doc("lecturers.xml")/Lecturers/Row/Name

appear in the final result, while the source nodes selected by the projection path $x/Rank on line 3 , i.e.,

doc("lecturers.xml")/Lecturers/Row/Rank

affect the result but do not appear directly in the result. So the projection path

doc("lecturers.xml")/Lecturers/Row/Name

is one of the *returned paths,* and the projection path

doc("lecturers.xml")/Lecturers/Row/Rank

is one of the *used paths* for the expression. These two projection paths are said to be related as a *path pair.* In a path pair, it is said that the *returned path* uses the *used path.* A *returned* *path* may use a set of *used paths,* and itself may function as a *used path* in a higher level expression. However, a *used path* may never function as a *returned path*. For example the projection path

doc("lecturers.xml")/Lecturers/Row/Rank

is a *returned path* for the expression $x/Rank on line 3, if the expression is analyzed as a standalone.

The expression on line 1

doc("lecturers.xml")/Lecturers/Row

is a subexpression of the expression in Listing 3.2. Its value is bound to the variable *x* during query evaluation. The returned path of this subexpression is

doc("lecturers.xml")/Lecturers/Row

and this returned path is said to be bound to the variable name *x* during the query's projection analysis.

Listing 3.3 shows an XQuery expression that answers the following question

How much does Dr. Mark Russell earn yearly?

Listing 3.3. Dr. Mark Russell's Salary

```
let $x := doc("lecturers.xml")/Lecturers/Row
return
   if $x/Name="Mark␣Russell"
   then
        ($x/Pay_rate)*($x/Monthly_hours)*12
   else ()
```

If there is no such person, nothing will result. Otherwise it prints out the yearly income. The following two projection paths

doc("lecturers.xml")/Lecturers/Row/Pay_rate

doc("lecturers.xml")/Lecturers/Row/Monthly_hours look like the *returned paths* for the whole expression, but they are actually *used paths*. The source nodes that were selected by them affect the result, but they do not appear in the result. You cannot find any occurrences of those source nodes in the result, since the multiplication operator returns a number and not a node.

Next, consider an XQuery expression with element constructors. The expression in Listing 3.4 lists all people living in Denton.

Listing 3.4. List People Living in Denton

```
1 for $y in doc("dir.xml")/Directory/Person
2 return
3       if ($y/Adddress/City = "Denton")
4       then
5            element Dentonese {
6                   element FullName {$y/@Name},
7                   $y/Phone
8            }
9       else ()
```

The returned path of the subexpression “$y/@Name” on line 6 is

doc("dir.xml")/Directory/Person/@Name

The subexpression on line 6

element FullName {$y/@Name}

constructs a new node with the name $FullName$. The content of this new node is

doc("dir.xml")/Directory/Person/@Name

Constructed nodes are identified by their identifiers. Let *fN* be the new node identifier assigned to this new node. The higher level subexpression on lines 5 to 7

element Dentonese

{ element FullName {$y/@Name}, $y/Phone } constructs a new node with the name *Dentonese*. Let *dN* be the new node identifier assigned to this new node. This node has as its children: a constructed node fN and some source nodes selected from the input document by the path

doc("dir.xml")/Directory/Person/Phone

It is obvious that the projection path

doc("dir.xml")/Directory/Person/Address/City

controls whether or not a Dentonese element will actually be created. The projection path

doc("dir.xml")/Directory/Person

controls how many Dentonese elements will be created. These two paths are called the *control paths* of the constructed node *dN* in this dissertation.

3.2.2. Environment

The techniques used to describe the algorithm used in this research are derived from structural operational semantics (*SOS*), which was originally developed by Gordon Plotkin in 1981 in his notes on A Structural Approach to Operational Semantics [67]. They are also discussed in many books [76, 56, 66, 83, 11], particularly in [11].

An environment called *Env* is used to record the mappings between variable names and their properties. Different properties are kept in their corresponding components in the environment.

The following are the environment components used by the inference rules in the research system:

- *VarRtnPathSet* records bindings between variable names and their returned paths.
- *VarConNodeSet* records bindings between variable names and their constructed nodes.

- *VarValue* records bindings between variable names and their assigned values. This is only used for determining whether or not the result of an expression will be an empty sequence when some of its variables are bound to empty sequences.

The notations used in this dissertation are similar to those used in [11] and in the XQuery 1.0 and XPath 2.0 Formal Semantics specification that appears on the W3C website [88]. Suppose *Comp* is a component of an environment *Env*. This fact is denoted as *Env.Comp*. The notation

$$Env.Comp\,(Var_1 \Rightarrow Val_1, \cdots, Var_n \Rightarrow Val_n)$$

denotes an environment component that maps a variable name Var_k to a value Val_k for $k = 1, \cdots, n$.

The notation $dom\,(Env.Comp)$ denotes the domain of *Env.Comp*, i.e., the set of variable names mapped in *Env.Comp*. In the above formula, the following assignment is made:

$$dom\,(Env.Comp) \quad = \quad \{Var_1, \cdots, Var_n\}$$

The notation $Env.Comp\,(Var)$ denotes the value mapped to the variable name *Var* in the environment component *Env.Comp*. If the value *Val* is its value, the equation is written

$$Env.Comp\,(Var) = Val$$

The notation $Env + Comp\,(Var \Rightarrow Val)$ denotes an environment that is the same as *Env* , except that its *Env.Comp* component maps the variable name *Var* to the value *Val*. When the formula is long, for convenience, the new state of the environment is given a new name, e.g. Env_2. This equation is written as:

$$Env_2 = Env + Comp\,(Var \Rightarrow Val)$$

Note that the extra *Env*. prefix is omitted from the notation *Env.Comp*, because the *Env* in the first term on the right can indicate which data structure is being changed (i.e., an *environment* or a *store* - see the next section).

The symbol $\emptyset$ is used to denote the empty set.

For example, the environment includes the following at one point of the projection analysis of the expression in Listing 3.2:

$$\mathit{Env.VarRtnPathSet}(x) = \{doc(\text{"lecturers.xml"})/Lecturers/Row\}$$

$$\mathit{Env.VarConNodeSet}(x) = \emptyset$$

Note if the variable x were bound to the result of the whole expression in Listing 3.4, then

$$\mathit{Env.VarRtnPathSet}(x) = \emptyset$$

$$\mathit{Env.VarConNodeSet}(x) = \{dN\}$$

3.2.3. Store

A store (i.e., *Store*) is used to record mappings between node identifiers and their different properties.

The store used by the inference rules in this dissertation has the following components:

- *NodeName* records the node name of a node, which is a mapping from a node identifier to a name.
- *NodeKind* records the kind of the node identified by a node identifier.
- *Parent* records the mapping from a node identifier to the node identifier of that node's parent. If the parent is unknown, the special value *nil* is used.
- *ConstructedChildren* records the mapping from a node identifier to the node identifiers of that node's children, which are also constructed nodes.
- *SourcePathPairSet* If a constructed node has some source nodes as its children, this component records the mapping from the node identifier to those children. Every child is represented as a path pair: a returned and a used projection path. If a returned path uses many used paths, the returned path is associated with the set of those used paths.

- *CtrlPathSet* records the mapping from the node identifier to the projection paths that control how the node is constructed. These paths control the occurrences of constructed nodes.

The notations for using the different store components are similar to those defined for environment components.

The following are in the store at the end of the projection analysis of the expression in Listing 3.4:

$$\begin{aligned}
&\mathit{Store.NodeName}(\mathit{fN}) = \text{"FullName"}\\
&\mathit{Store.NodeKind}(\mathit{fN}) = \text{"element"}\\
&\mathit{Store.Parent}(\mathit{fN}) = \mathit{dN}\\
&\mathit{Store.ConstructedChildren}(\mathit{fN}) = \emptyset\\
&\mathit{Store.SourcePathPairSet}(\mathit{fN}) =\\
&\qquad\qquad\{\langle doc(\text{"dir.xml"})/Directory/Person/@Name, \emptyset\rangle\}\\
&\mathit{Store.CtrlPathSet}(\mathit{fN}) = \emptyset
\end{aligned}$$

$$\begin{aligned}
&\mathit{Store.NodeName}(\mathit{dN}) = \text{"Dentonese"}\\
&\mathit{Store.NodeKind}(\mathit{dN}) = \text{"element"}\\
&\mathit{Store.Parent}(\mathit{dN}) = \mathit{nil}\\
&\mathit{Store.ConstructedChildren}(\mathit{dN}) = \{\mathit{fN}\}\\
&\mathit{Store.SourcePathPairSet}(\mathit{dN}) =\\
&\qquad\qquad\{\langle doc(\text{"dir.xml"})/Directory/Person/Phone, \emptyset\rangle\}\\
&\mathit{Store.CtrlPathSet}(\mathit{dN}) =\\
&\qquad\qquad\{doc(\text{"dir.xml"})/Directory/Person/Adddress/City\ ,\\
&\qquad\qquad doc(\text{"dir.xml"})/Directory/Person\}
\end{aligned}$$

3.3. Projection Judgment

Calligraphic letters are used to represent sets, and normal upper case letters are used to represent elements of sets.

The projection judgment is defined as:

$$Store_0, Env \vdash Expr \Rightarrow \langle \mathcal{P}, \mathcal{N} \rangle \, , Store_1$$

This is interpreted as, "given an old store $Store_0$ and environment Env, the projection analysis of $Expr$ yields an ordered pair of sets $\langle \mathcal{P}, \mathcal{N} \rangle$ with new store $Store_1$." The set $\mathcal{P}$ is a set of ordered pairs of projection paths, and the set $\mathcal{N}$ is a set of node identifiers of constructed nodes. $Store_1$ keeps all the nodes that are constructed along with their properties during the projection analysis. Before the projection analysis of any whole query, the initial environment and store are both represented as $\emptyset$. Judgments are inferred by applying the rules described in section 3.5.

The set of ordered path pairs $\mathcal{P}$ is in the form:

$$\{\langle R_i, U_i \rangle \mid i = 1, \ldots, m\} \text{ or } \{\langle R_i, \mathcal{U}_i \rangle \mid i = 1, \ldots, n\}$$

The latter form is a shorthand that merges path pairs with the same first elements; Consequently the second elements become sets. In the former form $\{\langle R_i, U_i \rangle \mid i = 1, \ldots, m\}$, the first path element R_i is a *returned path.* The source nodes in an input document selected by the $R_i$path appear directly as a result of evaluating $Expr$. The second path element U_i is a *used path*. The source nodes selected by the path U_i contribute indirectly to the result of the evaluation of $Expr$. The ordered path pair$\langle R_i, U_i \rangle$ denotes that the path U_i is a *used path* of the *returned path* R_i.

For example, the projection analysis of the following subexpression in Listing 3.2

```
if $x/Rank = "Full Professor"
then $x/Name else ( )
```

will yield the path pair

$\langle doc(\text{"lecturers.xml"})/Lecturers/Row/Name$,

$$doc(\text{"lecturers.xml"})/Lecturers/Row/Rank\rangle$$

For convenience and clarity, if two path pairs $\langle R_1, U_1\rangle$ and $\langle R_2, U_2\rangle$ have the same first element , i.e., $R_1 = R_2$, the shorthand notation $\langle R_1, \{U_1, U_2\}\rangle$is used. In fact, the notation $\{\langle R_i, \mathcal{U}_i\rangle \mid i = 1, \dots, n\}$ is used to denote the set $\mathcal{P}$ throughout the explanation of the inference rules, which are presented in section 3.5, where $\mathcal{U}_i$ is the set of all paths used by the returned path R_i.

As a special case, the returned path R_i can be ε, i.e., an empty path. The ordered path pair $\langle \varepsilon, \mathcal{U}_i\rangle$ deals with cases where the expression *Expr* returns no nodes from input documents, e.g. literals. For example, consider the following question:

Is Dr. Mark Russell a full professor?

The XQuery expression in the Listing 3.5 returns a “yes” or “no” to this question. If there is no such person, then nothing will be returned from the query.

Listing 3.5. Is Dr. Mark Russell a Full Professor?

```
1 let $x := doc("lecturers.xml")/Lecturers/Row
2 return
3      if $x/Name="Mark Russell"
4      then
5         if $x/Rank = "Full Professor"
6         then "Yes"
7         else  "No"
8      else ()
```

The projection analysis of the subexpression on line 5 to 7 of Listing 3.5

if \$x/Rank = "Full Professor" then "Yes" else "No"

will yield the path pair

$$\langle \varepsilon,\ doc(\text{"lecturers.xml"})/Lecturers/Row/Rank\rangle$$

No nodes from the input document appear in the final result.

The set of constructed nodes $\mathcal{N}$ is in the form

$$\{N_j \mid j = 1, \ldots, l\}$$

Each N_j is a node identifier of a top level node constructed by the expression *Expr*. Lower level nodes constructed by *Expr* , and nodes constructed by other expressions, are all memorized in the "Store."

As previously mentioned, the properties of constructed nodes are recorded in the components of a data structure called *store*. The forms of most components of the store are simple. However, *SourcePathPairSet* component records the path pairs of returned and used paths that contribute to the immediate content of a constructed node. The elements of the set called$Store.SourcePathPairSet(N_j)$ are in the form:

$$\langle S_j, H_i\rangle$$

This means that the source nodes selected by the paths S_j and $\mathcal{H}_j$ affect the immediate content of the newly constructed node directly and indirectly. The relationship between the S_j and the H_i paths of a path pair $\langle S_j, H_i\rangle$ are the same as the one between the path R_i and the U_i paths of a path pair $\langle R_i, U_i\rangle$. In fact, every path pair $\langle S_j, H_i\rangle$ is originally a path pair $\langle R_i, U_i\rangle$ of some subexpression. When that subexpression becomes part of the content of a XQuery constructor expression, its path pair $\langle R_i, U_i\rangle$ becomes an element of the $Store.SourcePathPairSet(N_j)$ set, if the newly constructed node has the $N_j$node identifier

The following shorthand form that merges path pairs with the same first elements is also used:

$$\langle S_j\ , \mathcal{H}_j\rangle$$

The set of paths $\mathcal{H}_j$ includes all the paths used by the path S_j.

Consider a slightly more complex example in the Listing 3.6

The result of this expression includes not only the source nodes of the input document selected by $x/Name, i.e., Listing 3.6. List Salaries of Professors

```
let $x := doc("lecturers.xml")/Lecturers/Row
return
   if $x/Rank = "Full␣Professor"
   then  (
        $x/Name,
        <Salary>
           {($x/Pay_rate)*($x/Monthly_hours)*12}
        </Salary> )
   else ( )
```

doc("lecturers.xml")/Lecturers/Row/Name

but also the newly created nodes of Salary, denoted by sN. The contents of the new nodes of Salary are related to the nodes of the input document selected by $x/Pay_rate, i.e.,

doc("lecturers.xml")/Lecturers/Row/Pay_rate,

and $x/Monthly_hours, i.e.,

doc("lecturers.xml")/Lecturers/Row/Monthly_hours

The projection analysis of this expression yields an ordered pair of sets $\langle \mathcal{P}, \mathcal{N} \rangle$ as

$$\langle \{ \langle doc(\text{"lecturers.xml"})/Lecturers/Row/Name,$$
$$\{ doc(\text{"lecturers.xml"})/Lecturers/Row\ ,$$
$$doc(\text{"lecturers.xml"})/Lecturers/Row/Rank \}\ ,$$
$$\rangle \}\ , \{ sN \} \rangle$$

with a store including the following content:

$Store.NodeName(sN) = \text{"Salary"}$

$Store.NodeKind(sN) = \text{"element"}$

$Store.Parent(sN) = nil$

$Store.ConstructedChildren(sN) = \emptyset$

$Store.SourcePathPairSet(sN) =$

$$\{\langle \varepsilon, \{doc(\text{"lecturers.xml"})/Lecturers/Row/Pay\_rate\#,$$
$$doc(\text{"lecturers.xml"})/Lecturers/Row/Monthly\_hours\#\}\rangle\}$$

$Store.CtrlPathSet(sN) =$

$$\{doc(\text{"lecturers.xml"})/Lecturers/Row\ ,$$
$$doc(\text{"lecturers.xml"})/Lecturers/Row/Rank\}$$

After processing with the extended XML document projection algorithm, the final projection path set of the example that is presented in Listing 3.6 is

$$\{doc(\text{"lecturers.xml"})/Lecturers/Row/Name\#,$$
$$doc(\text{"lecturers.xml"})/Lecturers/Row,$$
$$doc(\text{"lecturers.xml"})/Lecturers/Row/Rank,$$
$$doc(\text{"lecturers.xml"})/Lecturers/Row/Pay\_rate\#,$$
$$doc(\text{"lecturers.xml"})/Lecturers/Row/Monthly\_hours\#\ \}$$

In order to make the inference rules easier to understand, the following shorthands or operations are defined: Let $\mathcal{P} = \{\langle R_i, \mathcal{U}_i\rangle \mid i = 1, \ldots, n\}$ and $\mathcal{N} = \{N_j \mid j = 1, \ldots, l\}$. The set $\mathcal{N}$ is implicitly a set of nodes *under a store*. The store provides the parent and children information about a node. For simplicity, the store is omitted in the following notations.

- $RtnPathSet(\mathcal{P}) = RtnPathSet(\{\langle R_i, \mathcal{U}_i\rangle \mid i = 1, \ldots, n\}) = \{R_i \mid i = 1, \ldots, n\}$
- $UsedPathSet(\mathcal{P}) = UsedPathSet(\{\langle R_i, \mathcal{U}_i\rangle \mid i = 1, \ldots, n\}) = \cup_{i=1}^{n}\mathcal{U}_i$

- $\mathit{RtnUsedPathSet}(\mathcal{P}) = \mathit{RtnPathSet}(\mathcal{P}) \cup \mathit{UsedPathSet}(\mathcal{P})$
- $\mathit{Ancestors}(\mathcal{N}) = \bigcup_{N \in \mathcal{N}} (\{\mathit{Parent}(N)\} \cup \mathit{Ancestors}\,(\mathit{Parent}(N)))$
- $\mathit{Descendants}(\mathcal{N}) = \bigcup_{N \in \mathcal{N}} (\mathit{ConstructedChildren}(N) \cup$
  $\mathit{Descendants}\,(\mathit{ConstructedChildren}(N))$
- $\mathit{UpCtrlPathSet}(\mathcal{N}) = \cup_{N \in \mathcal{N} \cup \mathit{Ancestors}(\mathcal{N})} \mathit{CtrlPathSet}(N)$
- $\mathit{AllCtrlPathSet}(\mathcal{N}) = \cup_{N \in \mathcal{N} \cup \mathit{Ancestors}(\mathcal{N}) \cup \mathit{Descendants}(\mathcal{N})} \mathit{CtrlPathSet}(N)$
- $\mathit{TreePathPairSet}(\mathcal{N}) = \cup_{N \in \mathcal{N} \cup \mathit{Descendants}(\mathcal{N})} \mathit{SourcePathPairSet}(N)$
- $\mathit{BasePathSet}(\mathcal{N}) = \mathit{AllCtrlPathSet}(\mathcal{N}) \cup$
  $\mathit{RtnPathSet}(\mathit{TreePathPairSet}(\mathcal{N})) \cup$
  $\mathit{UsedPathSet}(\mathit{TreePathPairSet}(\mathcal{N}))$
- $\mathit{FinalizedRtnPathSet}(\mathcal{P}) = \cup_{R \in \mathit{RtnPathSet}(\mathcal{P})} \{R\#\}$
- $\mathit{FinalizedPathSet}(\mathcal{P}, \mathcal{N}) = \cup_{R \in \mathit{RtnPathSet}(\mathcal{P})} \{R\#\} \cup$
  $\cup_{U \in \mathit{UsedPathSet}(\mathcal{P})} \{U\} \cup$
  $\cup_{S \in \mathit{RtnPathSet}(\mathit{TreePathPairSet}(\mathcal{N}))} \{S\#\} \cup$
  $\cup_{H \in \mathit{UsedPathSet}(\mathit{TreePathPairSet}(\mathcal{N}))} \{H\} \cup$
  $\mathit{AllCtrlPathSet}(\mathcal{N})$

3.4. XQuery Core Subset

An XQuery core subset is used to demonstrate the techniques described in this dissertation. It is similar to the one used by Marian and Siméon [52]. This grammar can provide full support for XPath 1.0 and all important XQuery features, including FLWR expressions that could pose significant challenges to the application of the techniques implemented in this research. The symbols ''()|?* are meta symbols. Symbols beginning with upper case letter are grammatical symbols, while symbols in lower cases are keywords.

The Listing 3.7 shows the grammar used in this system.

Listing 3.7. The Grammar of the Subset of XQuery Core

```
VarName ::= QName
Expr ::= Literal
        | ‘(’ ‘)’
        | /
        | Expr, Expr
        | QName ‘(’ (Expr (, Expr) * ) ? ‘)’
        | doc‘(’ URL ‘)’
        | Expr (= | >) Expr
        | Expr (+ | *) Expr
        | Expr union Expr
        |$VarName
        | for $VarName in Expr return Expr
        | let $VarName := Expr return Expr
        | Axis NodeTest
        |  if ‘(’ Expr ‘)’ then Expr else Expr
        | (element | attribute) QName {Expr}
        | (element | attribute) {Expr} {Expr}
        | document {Expr}
        | text {String}
```

3.5. Inference Rules

The following is a description of the inference rules that are used in my extended projection algorithm. The notations are similar to those used in [11, 88]. Each rule has zero or more judgements above the line, called hypotheses or premises, and one judgement below the line, called the conclusion [76, 56, 66, 83, 11]. The conclusion holds if all the hypotheses hold.

The rationale behind every rule is that the projection paths need to be able to retrieve all the necessary data from the source documents that are required to compute an XQuery expression. The different types of rules available in the system are presented below.

3.5.1. Literal Values

Literal values do not reference any data in input documents or create any new nodes. Thus, they do not contribute any projection paths nor constructed nodes. No changes are made to the store either.

$$\frac{}{Store_0, Env \vdash Literal \Rightarrow \langle \emptyset, \emptyset \rangle,\ Store_0}$$

3.5.2. Empty Sequence

Similar to Literals, an empty sequence does not refer to any data in source documents or create any new nodes, and thus they contribute zero projection paths and zero constructed nodes. No changes are made to the store either.

$$\frac{}{Store_0, Env \vdash (\ ) \Rightarrow \langle \emptyset, \emptyset \rangle,\ Store_0}$$

3.5.3. Root Path

A root path expression requires that the root path be kept as a projection path. It does not construct any new elements, and it does not change the store.

$$\frac{}{Store_0, Env \vdash / \Rightarrow \langle \{ \langle /, \emptyset \rangle \}, \emptyset \rangle,\ Store_0}$$

3.5.4. Sequence Expressions

The source data that is needed to compute a sequence expression is the union of the source data that is required to compute its subexpressions. Thus, the sets of projection path pairs and the sets of constructed nodes are propagated from the subexpressions to the higher level expression. In addition, subexpressions sometimes change the store.

$$\frac{\begin{array}{l} \mathit{Store}_0, \mathit{Env} \vdash \mathit{Expr}_1 \Rightarrow \langle \mathcal{P}_1, \mathcal{N}_1 \rangle, \mathit{Store}_1 \\ \mathit{Store}_1, \mathit{Env} \vdash \mathit{Expr}_2 \Rightarrow \langle \mathcal{P}_2, \mathcal{N}_2 \rangle, \mathit{Store}_2 \end{array}}{\begin{array}{l} \mathit{Store}_0, \mathit{Env} \vdash \mathit{Expr}_1, \mathit{Expr}_2 \\ \qquad \Rightarrow \langle \mathcal{P}_1 \cup \mathcal{P}_2, \mathcal{N}_1 \cup \mathcal{N}_2 \rangle, \mathit{Store}_2 \end{array}}$$

3.5.5. Function Calls

There are inference rules that call user defined functions, as well as inference rules that call important or typical built-in functions. The inference rules that call user defined functions are described first. Generally, built-in functions must be analyzed on a case by case basis[52].

3.5.5.1. Calls to User Defined Functions

The system currently only analyzes non-recursive functions. First, the function declaration is retrieved. The environment Env_0 defined by the module of the function declaration is also retrieved. Next, the actual argument expressions are analyzed under the current environment Env in which the function is called. The results are bound to the formal parameter variables and added to Env_0 to form a new environment Env_0'. Finally, the function body is analyzed under the environment Env_0'. The store as well as the function body might have been changed by the actual argument expressions.

The projection analysis of a call to a user defined function with no parameter QName() is equivalent to the projection analysis of the corresponding function body. Thus, the system has the following rule.

$$\frac{\begin{array}{c}\text{declare function QName}(\cdots)\ \{\mathit{Expr}_0\}\\ \mathit{Store}_0, \mathit{Env}_0 \vdash \mathit{Expr}_0 \Rightarrow \langle \mathcal{P}_0, \mathcal{N}_0 \rangle,\ \mathit{Store}_1\end{array}}{\mathit{Store}_0, \mathit{Env} \vdash \text{QName}() \Rightarrow \langle \mathcal{P}_0, \mathcal{N}_0 \rangle,\ \mathit{Store}_1}$$

The system also accepts user defined functions with parameters. Suppose the function declaration of a user defined function that requires some parameters is

$$\text{declare function QName}(\$\mathit{VarName}_1, \cdots, \$\mathit{VarName}_n)\{\mathit{Expr}_0\}$$

The projection analysis of a call to this function QName($\mathit{Expr}_1, \cdots, \mathit{Expr}_n$) is equivalent to the projection analysis of the expression in Listing 3.8.

Listing 3.8. An Equivalent Expression to a Function Call for Projection Analysis

```
let $VarName_1 := Expr_1 return
...
let $VarName_n := Expr_n return
Expr_0
```

Thus, the system has the following rule. (Please refer to the analysis of `let` expressions in 3.5.8.)

declare function QName($\$VarName_1, \cdots, \$VarName_n$)$\{Expr_0\}$

$$\frac{\begin{array}{l} Store_0, Env \vdash Expr_1 \Rightarrow \langle \mathcal{P}_1, \mathcal{N}_1 \rangle, Store_1 \\ \quad\vdots \\ Store_{n-1}, Env \vdash Expr_n \Rightarrow \langle \mathcal{P}_n, \mathcal{N}_n \rangle, Store_n \\ Env'_0 = Env_0 + \sum_{k=1}^{n} VarRtnPathSet(VarName_k \Rightarrow RtnPathSet(\mathcal{P}_k)) \\ \qquad + \sum_{k=1}^{n} VarConNodeSet(VarName_k \Rightarrow \mathcal{N}_k) \\ Store_n, Env'_0 \vdash Expr_0 \Rightarrow \langle \{\langle R_{0,i}, \mathcal{U}_{0,i} \rangle \mid i = 1, \ldots, n_0\}, \{N_{0,j} \mid j = 1, \ldots, l_0\} \rangle, Store_{n+1} \\ Store_{n+2} = Store_{n+1} + \sum_{j=1}^{l_0} CtrlPathSet(N_{0,j} \Rightarrow CtrlPathSet(N_{0,j}) \cup \\ \qquad \cup_{k=1}^{n} (UsedPathSet(\mathcal{P}_k) \cup UpCtrlPathSet(\mathcal{N}_k)) \end{array}}{\begin{array}{l} Store_0, Env \vdash \mathrm{QName}(Expr_1, \cdots, Expr_n) \\ \quad \Rightarrow \langle \{\langle R_{0,i}, \mathcal{U}_{0,i} \cup \cup_{k=1}^{n} (UsedPathSet(\mathcal{P}_k) \cup UpCtrlPathSet(\mathcal{N}_k)) \rangle \mid i = 1, \ldots, n_0\}, \\ \quad \{N_{0,j} \mid j = 1, \ldots, l_0\} \rangle, Store_{n+2} \end{array}}$$

3.5.5.2. Input Functions

Similar to a root path expression, an input function expression $doc(URL)$ brings forward a whole source document, but it does not construct any new elements nor change the store.

$$\frac{}{Store_0, Env \vdash doc(URL) \Rightarrow \langle \{\langle doc(URL), \emptyset \rangle\}, \emptyset \rangle, Store_0}$$

3.5.5.3. Comparison Expressions

Comparison expressions are implemented by internal functions. Comparison expressions do not return any source nodes or construct any new nodes. However, they do use source nodes to accomplish the comparison. Their subexpressions might change the store. Marian and Siméon [52] simply add all returned paths without # from both subexpressions to the set of used paths. However, the system described in this dissertation appends # to the end of every returned path because the atomization of document nodes and element nodes might need their descendant text nodes [87].

$$\frac{\begin{array}{l} Store_0, Env \vdash Expr_1 \Rightarrow \langle \mathcal{P}_1,\ \mathcal{N}_1 \rangle,\ Store_1 \\ Store_1, Env \vdash Expr_2 \Rightarrow \langle \mathcal{P}_2,\ \mathcal{N}_2 \rangle,\ Store_2 \end{array}}{\begin{array}{l} Store_0, Env \vdash Expr_1\ (=\mid>)\ Expr_2 \Rightarrow \\ \qquad \langle \{ \langle \varepsilon,\ FinalizedPathSet(\mathcal{P}_1,\ \mathcal{N}_1) \cup FinalizedPathSet(\mathcal{P}_2,\ \mathcal{N}_2) \rangle \},\emptyset \rangle,\ Store_2 \end{array}}$$

3.5.5.4. Arithmetic Expressions

Arithmetic expressions are also implemented by internal functions. The analysis done on them is similar to the one done on comparison expressions. They do not return any source nodes or construct any new nodes. They only use source nodes to accomplish the computation. Their subexpressions might change the store.

$$\frac{\begin{array}{l} Store_0, Env \vdash Expr_1 \Rightarrow \langle \mathcal{P}_1,\ \mathcal{N}_1 \rangle,\ Store_1 \\ Store_1, Env \vdash Expr_2 \Rightarrow \langle \mathcal{P}_2,\ \mathcal{N}_2 \rangle,\ Store_2 \end{array}}{\begin{array}{l} Store_0, Env \vdash Expr_1\ (+|*)\ Expr_2 \Rightarrow \\ \qquad \langle \{ \langle \varepsilon,\ FinalizedPathSet(\mathcal{P}_1,\ \mathcal{N}_1) \cup FinalizedPathSet(\mathcal{P}_2,\ \mathcal{N}_2) \rangle \},\emptyset \rangle,\ Store_2 \end{array}}$$

3.5.5.5. Union Expressions

Union expressions are implemented by internal functions and are similar to sequence expressions. The sets of projection path pairs and the sets of constructed nodes are propagated from the subexpressions to the higher level expressions. In addition, the subexpressions might change the store.

$$\frac{\begin{array}{l} Store_0,\ Env \vdash Expr_1 \Rightarrow \langle \mathcal{P}_1,\ \mathcal{N}_1 \rangle,\ Store_1 \\ Store_1,\ Env \vdash Expr_2 \Rightarrow \langle \mathcal{P}_2,\ \mathcal{N}_2 \rangle,\ Store_2 \end{array}}{\begin{array}{l} Store_0,\ Env \vdash Expr_1 \text{ union } Expr_2 \\ \qquad \Rightarrow \langle \mathcal{P}_1 \cup \mathcal{P}_2,\ \mathcal{N}_1 \cup \mathcal{N}_2 \rangle,\ Store_2 \end{array}}$$

3.5.6. Variable References

Variable reference expressions are cited in the set of projection paths and the set of constructed nodes they are bound to.

$$\frac{\begin{array}{l} Env.varRtnPathSet(VarName) = \{R_i | i = 1, \ldots, n\} \\ Env.VarConNodeSet(VarName) = \mathcal{N} \end{array}}{Store_0, Env \vdash \$VarName \Rightarrow \langle \{\langle R_i, \emptyset \rangle \mid i = 1, \ldots, n\}, \mathcal{N} \rangle, Store_0}$$

3.5.7. `for` Expressions

The system uses two inference rules for `for` expressions. The first rule is an unoptimized rule, whereas the second rule is an optimized rule that may result in smaller projection documents. To use the optimized rule, you need to know whether the expression in the return clause of a `for` expression will be evaluated to an empty sequence when the binding variable is bound to an empty sequence. In order to do that, the system could have included rules that would test for every different case. This was not done for this system, however, because it would have resulted in more complicated rules. Thus, a new, high level judgement was introduced to avoid this problem. The new judgement is called the empty sequence judgement and is in the following form:

$$Env \vdash Expr \text{ is } ()$$

This judgement holds true when, given the *Env*environment, the evaluation of the expression *Expr* yields an empty sequence. Since this judgement has nothing to do with the store, the store is omitted.

The system also contains an unoptimized rule for `for` expressions. According to this rule, the `for` expression preserves all projection paths of its subexpressions. The set of projection paths need to be preserved from the input expression $Expr_1$ is $RtnUsedPathSet(\mathcal{P}_1) \cup UpCtrlPathSet(\mathcal{N}_1)$. This set of paths becomes part of the used paths of every returned path $R_{2,i}$, $i = 1, \ldots, n_2$, and part of the control paths of every returned top level constructed node $N_{2,j}$, $j = 1, \ldots, l_2$.

The $Env'' \vdash not(Expr_2 \text{ is } ())$ premise is what tells the system that the $Expr_2$ expression in the body of the `for` expression is not necessarily evaluated to an empty sequence under the

Env'' environment, which maps the binding *VarName* variable to an empty sequence. Note that the path pair $\langle \varepsilon,\ \mathit{RtnUsedPathSet}(\mathcal{P}_1) \cup \mathit{UpCtrlPathSet}(\mathcal{N}_1)\rangle$ accommodates special cases where the expression Expr_2 is a literal, such as a string or an integer, which projects to $\langle \emptyset, \emptyset \rangle$. Admittedly contrived, an example expression is

$$\texttt{for}\ \$x\ \texttt{in}\ doc(\text{"lecturers.xml"})/Lecturers/Row\ \texttt{return}\ \text{"Lecturer Found"}$$

This expression prints "Lecturer Found" every time that it is found, depending on the path

doc("lecturers.xml")/Lecturers/Row

If the path pair $\langle \varepsilon,\ \mathit{RtnUsedPathSet}(\mathcal{P}_1) \cup \mathit{UpCtrlPathSet}(\mathcal{N}_1)\rangle$ is omitted from the rule, the projection analysis of the above example expression will produce $\langle \emptyset, \emptyset \rangle$ instead of

$$\langle \{\langle \varepsilon,\ doc(\text{"lecturers.xml"})/Lecturers/Row\rangle\}\ , \emptyset\rangle$$

Consequently, the system will be unable to generate the required number of "Lecturers Found".

When the Expr_2 expression is an empty sequence, it also projects to $\langle \emptyset, \emptyset \rangle$. However, the system does not need to remember how many empty sequences are created, since the result is still a single empty sequence.

$$\begin{array}{l}
Store_0, Env \vdash Expr_1 \Rightarrow \langle \mathcal{P}_1, \mathcal{N}_1 \rangle, Store_1 \\
\quad Env' = Env + VarRtnPathSet(VarName \Rightarrow RtnPathSet(\mathcal{P}_1)) \\
\qquad\qquad + VarConNodeSet(VarName \Rightarrow \mathcal{N}_1) \\
Store_1, Env' \vdash Expr_2 \Rightarrow \langle \{\langle R_{2,i}, \mathcal{U}_{2,i} \rangle \mid i = 1, \ldots, n_2\}, \{N_{2,j} \mid j = 1, \ldots, l_2\} \rangle, Store_2 \\
\quad Store_3 = Store_2 + \sum_{j=1}^{l_2} CtrlPathSet(N_{2,j} \Rightarrow \\
\qquad\qquad CtrlPathSet(N_{2,j}) \cup RtnUsedPathSet(\mathcal{P}_1) \cup UpCtrlPathSet(\mathcal{N}_1)) \\
\quad Env'' = Env + VarValue(VarName \Rightarrow (\,)) \\
\quad Env'' \vdash not(Expr_2 \text{ is } (\,)) \\
\hline
Store_0, Env \vdash \text{for } \$VarName \text{ in } Expr_1 \text{ return } Expr_2 \Rightarrow \\
\qquad \langle \{\langle R_{2,i}, \mathcal{U}_{2,i} \cup RtnUsedPathSet(\mathcal{P}_1) \text{ Artichoke Leaf Extract} \\
\qquad\qquad \cup UpCtrlPathSet(\mathcal{N}_1) \rangle \mid i = 1, \ldots, n_2\} \\
\qquad \bigcup \{\langle \varepsilon, RtnUsedPathSet(\mathcal{P}_1) \cup UpCtrlPathSet(\mathcal{N}_1) \rangle\}, \\
\qquad \{N_{2,j} \mid j = 1, \ldots, l_2\} \rangle, Store_3
\end{array}$$

The unoptimized rule could have been broken into several rules. For example, if one adds a premise to tell the system whether the expression $Expr_2$ is a literal, the path pair

$$\langle \varepsilon, RtnUsedPathSet(\mathcal{P}_1) \cup UpCtrlPathSet(\mathcal{N}_1) \rangle$$

will need to appear in the rule only when the premise is true. If one adds a rule that has a premise that tells the system that the projection path $R_{2,i}$ is generated whenever the binding variable $VarName$ is bound to $RtnPathSet(\mathcal{P}_1)$, the system will not need to add the projection paths $UpCtrlPathSet(\mathcal{N}_1)$ as used paths of the path $R_{2,i}$. However, this seemed to make the system more complicated and was not added at this time.

The second `for`rule deals with cases in which optimization can be applied. The purpose of optimization is to avoid unnecessary intermediate projection paths. The techniques used in this research are similar to those presented in Marian and Siméon [52, 53].

The premise of optimization is:

$Env'' \vdash Expr_2$ is $(\,)$

This declares that the expression $Expr_2$ in the body of the `for` expression is always evaluated to an empty sequence under the environment Env'', which maps the binding variable $VarName$ to an empty sequence. Under this premise, the projection paths in the path set $RtnPathSet(\mathcal{P}_1)$ can be omitted, because the nodes selected are either kept by the projection paths in the $RtnUsedPathSet(\mathcal{P}_2)$ set or they produce no observable effects. Also, the path pair $\langle \varepsilon,\ RtnUsedPathSet(\mathcal{P}_1) \cup UpCtrlPathSet(\mathcal{N}_1)\rangle$are not included in this rule. This was required when the expression $Expr_2$ was a literal, but since $Env'' \vdash Expr_2$ is $(\,)$, we know that the the expression $Expr_2$ could never return just a literal. For example, the following expression

$$\texttt{for}\ \$x\ \texttt{in}\ doc(\text{"lecturers.xml"})/Lecturers/Row\ \texttt{return}\ \text{"Lecturer Found"}$$

violates the judgement $Env'' \vdash Expr_2$ is $(\,)$.

$$\frac{\begin{array}{l} Store_0, Env \vdash Expr_1 \Rightarrow \langle \mathcal{P}_1,\ \mathcal{N}_1\rangle,\ Store_1 \\ \quad Env' = Env + VarRtnPathSet(VarName \Rightarrow RtnPathSet(\mathcal{P}_1)) \\ \qquad\qquad + VarConNodeSet(VarName \Rightarrow \mathcal{N}_1) \\ Store_1, Env' \vdash Expr_2 \Rightarrow \langle \{\langle R_{2,i},\ \mathcal{U}_{2,i}\rangle \mid i = 1,\ldots,n_2\},\{N_{2,j} \mid j = 1,\ldots,l_2\}\rangle,\ Store_2 \\ \quad Store_3 = Store_2 + \sum_{j=1}^{l_2} CtrlPathSet(N_{2,j} \Rightarrow \\ \qquad\qquad CtrlPathSet(N_{2,j}) \cup UsedPathSet(\mathcal{P}_1) \cup UpCtrlPathSet(\mathcal{N}_1)) \\ \quad Env'' = Env + VarValue(VarName \Rightarrow (\,)) \\ \quad Env'' \vdash Expr_2 \text{ is } (\,) \end{array}}{\begin{array}{l} Store_0, Env \vdash \text{for } \$VarName \text{ in } Expr_1 \text{ return } Expr_2 \Rightarrow \\ \qquad \langle \{\langle R_{2,i},\ \mathcal{U}_{2,i} \cup UsedPathSet(\mathcal{P}_1) \cup UpCtrlPathSet(\mathcal{N}_1)\rangle \mid i = 1,\ldots,n_2\}, \\ \qquad \{N_{2,j} \mid j = 1,\ldots,l_2\}\rangle,\ Store_3 \end{array}}$$

This rule does not need to include the path set $UpCtrlPathSet(\mathcal{N}_1)$, because the necessary projection paths among them have already been bound to the sets $\mathcal{U}_{2,i}$ or

$\mathit{UpCtrlPathSet}(\{N_{2,j} \mid j = 1, \ldots, l_2\})$. Nevertheless, they are included n this rule because the algorithm's correctness is difficult to prove without them.

3.5.8. `let` Expressions

The `let` expression has a rule that is similar to the one used for `for` expressions. If the nodes selected by the projection paths in the $\mathit{RtnPathSet}(\mathcal{P}_1)$ set result in observable output that is not contained in one of the constructed nodes, then those nodes have already been assigned to the $\mathit{RtnUsedPathSet}(\{\langle R_{2,i}, \mathcal{U}_{2,i}\rangle \mid i = 1, \ldots, n_2\})$ set. The nodes selected by the projection paths in the $\mathit{RtnPathSet}(\mathcal{P}_1)$ set might also produce observable output as the content of some constructed nodes, so the projection path $\mathit{UsedPathSet}(\mathcal{P}_1)$ set must be added to $\mathit{CtrlPathSet}(N_{2,j})$.

The optimized rule for the`let` expression does not include the path set $\mathit{UpCtrlPathSet}(\mathcal{N}_1)$ because the necessary projection paths among them have already been bound to the sets $\mathcal{U}_{2,i}$ or $\mathit{UpCtrlPathSet}(\{N_{2,j} \mid j = 1, \ldots, l_2\})$. Nevertheless, these are included because the algorithm's correctness is difficult to prove without them.

The contributing projection paths in the projection path set $\mathit{UpCtrlPathSet}(\mathcal{N}_1)$ have already been inherited by $\mathcal{P}_2$ and $\mathcal{N}_2$.

$$\frac{\begin{array}{l} \mathit{Store}_0, \mathit{Env} \vdash \mathit{Expr}_1 \Rightarrow \langle \mathcal{P}_1, \mathcal{N}_1\rangle, \mathit{Store}_1 \\ \qquad \mathit{Env}' = \mathit{Env} + \mathit{VarRtnPathSet}(\mathit{VarName} \Rightarrow \mathit{RtnPathSet}(\mathcal{P}_1)) \\ \qquad\qquad + \mathit{VarConNodeSet}(\mathit{VarName} \Rightarrow \mathcal{N}_1) \\ \mathit{Store}_1, \mathit{Env}' \vdash \mathit{Expr}_2 \Rightarrow \langle \{\langle R_{2,i}, \mathcal{U}_{2,i}\rangle \mid i = 1, \ldots, n_2\}, \{N_{2,j} \mid j = 1, \ldots, l_2\}\rangle, \mathit{Store}_2 \\ \qquad \mathit{Store}_3 = \mathit{Store}_2 + \sum_{j=1}^{l_2} \mathit{CtrlPathSet}(N_{2,j} \Rightarrow \\ \qquad\qquad \mathit{CtrlPathSet}(N_{2,j}) \cup \mathit{UsedPathSet}(\mathcal{P}_1) \cup \mathit{UpCtrlPathSet}(\mathcal{N}_1)) \end{array}}{\begin{array}{l} \mathit{Store}_0, \mathit{Env} \vdash \texttt{let } \$\mathit{VarName} \texttt{ in } \mathit{Expr}_1 \texttt{ return } \mathit{Expr}_2 \Rightarrow \\ \qquad \langle \{\langle R_{2,i}, \mathcal{U}_{2,i} \cup \mathit{UsedPathSet}(\mathcal{P}_1) \cup \mathit{UpCtrlPathSet}(\mathcal{N}_1)\rangle \mid i = 1, \ldots, n_2\}, \\ \qquad \{N_{2,j} \mid j = 1, \ldots, l_2\}\rangle, \mathit{Store}_3 \end{array}}$$

3.5.9. XPath Steps

The algorithm is now ready to harvest the fruits of building a forest of constructed elements or documents in memory. First, it is necessary to introduce a new judgement called a step judgement:

$$\mathit{Store}, \mathit{Env} \vdash \mathcal{N}_1 / \mathit{Axis}\ \mathit{NodeTest} = \langle \mathcal{P}_2, \mathcal{N}_2 \rangle$$

This judgement states that, "given store *Store* and environment *Env*, the evaluation of step *Axis NodeTest* on the set of constructed nodes $\mathcal{N}_1$ yields $\langle \mathcal{P}_2, \mathcal{N}_2 \rangle$". Note this is not a projection judgement. To signify the difference, the symbol $=$ is used instead of $\Rightarrow$.

The step operations are defined to mimic the semantics of XPath steps.

Suppose $\mathcal{N}_1$ is

$$\{N_{1,j} \mid j = 1, \ldots, l_1\}$$

As an example, an explanation of the how to evaluate the operation when the axis is "child::" is provided. Note that only constructed documents and element nodes can have children. Thus, any other constructed nodes in the set $\mathcal{N}_1$ should be thrown away immediately. It is assumed that the set $\mathcal{N}_1$only includes constructed document nodes and element nodes in the discussion of this subsection.

(1) $\mathcal{N}_2 = \left\{ N \in \bigcup_{j=1}^{l_1} \mathit{ConstructedChildren}(N_{1,j}) \mid N \text{ satisfies } \mathit{NodeTest} \right\}$

The system must now decide how "N satisfies *NodeTest*". There are two cases to discuss.

(a) If NodeTest is NameTest, the system checks if NodeName(N) satisfies NameTest.

(i) The NameTest is the wildcard *.

Any element node will satisfy the NameTest.

(ii) The NameTest has the form NCName:*.

Any element node N will satisfy the NameTest if the namespace URI of its NodeName(N) is the same as the namespace URI corresponding to NCName.

(iii) The NameTest has the form *:NCName.

Any element node N will satisfy the NameTest if the local name of its NodeName(N) is the same as NCName.

(iv) The NameTest has the form NCName:NCName.

Any element node N will satisfy the NameTest if the expanded qualified name of the node is equal to the expanded qualified name specified by the NameTest, i.e. the namespace URI of the NodeName(N) is the same as the namespace URI corresponding to the first NCName and the local name of the NodeName(N) is the same as the second NCName.

(v) The NameTest has the form NCName

This subcase is the same as the above except that the namespace URI corresponding to the NCName is the default one or none.

Note that any element node N with the special node name **unknown** satisfies all the above NameTest variations. The special node called **unknown** is reserved for the nodes constructed by element or attribute constructors with name expressions (refer to subsections 3.5.12 and 3.5.14). Namespaces were not processed by the prototype system, because they do not increase the complexity the problem.

(b) If NodeTest is KindTest, the system checks if NodeKind(N) satisfies KindTest

(i) The KindTest is node()

Any node will satisfy this KindTest.

(ii) The KindTest is text()

Any node N will satisfy the KindTest if its NodeKind(N) is "text".

The grammar of the projection paths includes only the two kind tests above, although other types of tests are not difficult to add.

(2) To compute $\mathcal{P}_2$, consider each projection path pair

$$\langle S\ , \mathcal{H}\rangle \in \bigcup_{j=1}^{l_1} SourcePathPairSet(N_{1,j}) \text{ and } S \text{ is not } \varepsilon$$

There exist two cases that need to be discussed depending on whether the system is *able to know* that the projection path S satisfies *NodeTest*. The information that is available during the projection analysis is the XQuery query and possibly the schemas of the input documents).

(a) The system is *able to know* that the projection path S satisfies *NodeTest* or not.

This case can be divided into the following subcases. In each subcase, check the last step L of the projection path S to decide if it satisfies *NodeTest*.

(i) The NodeTest is the wildcard *.

The projection path S satisfies *NodeTest* if the step L selects element nodes.

The projection path S does not satisfy *NodeTest* if the step L selects document nodes or attribute nodes or text nodes.

(ii) The NodeTest has the form NCName:*.

The projection path S satisfies *NodeTest* if the step L selects element nodes whose namespace URI's are the same as the namespace URI corresponding to NCName.

The projection path S does not satisfy *NodeTest* if

(A) the step L selects element nodes whose namespace URI's are different from the namespace URI corresponding to NCName.

(B) the step L selects document nodes or attribute nodes or text nodes.

(iii) The NodeTest has the form *:NCName.

The projection path S satisfies *NodeTest* if the step L selects element nodes whose local names are the same as NCName.

The projection path S does not satisfy *NodeTest* if

(A) the step L selects element nodes whose whose local names are different from NCName.

(B) `let` expression the step L selects document nodes or attribute nodes or text nodes.

(iv) The NodeTest has the form NCName:NCName.

The projection path S satisfies *NodeTest* if the step L selects element nodes whose namespace URI's are the same as the namespace URI corresponding to the first NCName and whose local names are the same as the second NCName.

The projection path S does not satisfy *NodeTest* if

(A) the step L selects element nodes whose namespace URI's are different from the namespace URI corresponding to the first NCName or whose local names are different from the second NCName.

(B) the step L selects document nodes or attribute nodes or text nodes.

(v) The NodeTest has the form NCName

This subcase is the same as the above except that the namespace URI corresponding to the NCName is the default or nothing is used.

(vi) The NodeTest is node()

The projection path S always satisfies this *NodeTest*.

(vii) The NodeTest is text()

The projection path S satisfies this *NodeTest* if its last step L selects text nodes, that is the NodeTest of step L is text().

The projection path S does not satisfy *NodeTest* if the step L selects document nodes or attribute nodes or element nodes.

The system is unable to utilize schemas at this time. If it could, then it would be able to recognize when the projection path S satisfied *NodeTest* more often. However, the following actions are performed for each path pair $\langle S\ , \mathcal{H}\rangle$.

(i) If the system is *able to know* that the projection path S satisfies *NodeTest*, then it adds the projection path pair $\langle S\ , \mathcal{H} \cup \mathit{AllCtrlPathSet}(N_{1,j})\rangle$ to the path pair set $\mathcal{P}_2$. Denote

$$\begin{aligned} \mathcal{P}_{2,1} \quad = \quad & \{\langle S\ , \mathcal{H} \cup \mathit{AllCtrlPathSet}(N_{1,j})\rangle \mid \\ & \langle S\ , \mathcal{H}\rangle \in \cup_{j=1}^{l_1} \mathit{SourcePathPairSet}(N_{1,j}) \text{ and} \\ & S \text{ is not } \varepsilon \text{ and know } S \text{ satisfies } \mathit{NodeTest}\} \end{aligned}$$

(ii) If the system is *able to know* that the projection path S does *not* satisfy *NodeTest*, then the system will not add any projection path pair to the path pair set $\mathcal{P}_2$ for the path pair $\langle S\ , \mathcal{H}\rangle$.

(b) If the system does not *know* that the projection path S satisfies *NodeTest* or not.

This is the unoptimized and default case. The system concludes that the projection path S might or might not satisfy *NodeTest*. The system adds the projection path pair $\langle S/\mathit{Self} :: \mathit{NodeTest}\ , \mathcal{H} \cup \mathit{AllCtrlPathSet}(N_{1,j})\rangle$to the path pair set $\mathcal{P}_2$. Denote

$$\begin{aligned} \mathcal{P}_{2,2} \quad = \quad & \{\langle S/\mathit{Self} :: \mathit{NodeTest}\ , \mathcal{H} \cup \mathit{AllCtrlPathSet}(N_{1,j})\rangle \mid \\ & \langle S\ , \mathcal{H}\rangle \in \cup_{j=1}^{l_1} \mathit{SourcePathPairSet}(N_{1,j}) \text{ and} \end{aligned}$$

S is not ε and I do not know if S satisfies $\mathit{NodeTest}\}$

Finally, $\mathcal{P}_2 = \mathcal{P}_{2,1} + \mathcal{P}_{2,2}$

Other axes can be similarly defined. Note that XPath paths with reverse axes can be transformed into reverse-axis-free ones [60]. The following is the inference rule for XPath steps.

$$\frac{\begin{array}{l} \mathit{Env.VarRtnPathSet}(.) = \{R_1, \cdots, R_n\} \\ \mathit{Env.VarConNodeSet}(.) = \mathcal{N}_1 \\ \mathit{Store}, \mathit{Env} \vdash \mathcal{N}_1 / \mathit{Axis}\ \mathit{NodeTest} = \langle \mathcal{P}_2, \mathcal{N}_2 \rangle \end{array}}{\begin{array}{l} \mathit{Store}, \mathit{Env} \vdash \mathit{Axis}\ \mathit{NodeTest} \Rightarrow \\ \qquad \left\langle \cup_{i=1}^{n} \{\langle R_i / \mathit{Axis}\ \mathit{NodeTest}, \emptyset \rangle \mid R_i \text{ is not } \varepsilon\} \cup \mathcal{P}_2, \mathcal{N}_2 \right\rangle, \mathit{Store} \end{array}}$$

3.5.10. Conditional Expressions

The result of a conditional expression combines the results of the then-expression and the else-expression. The projection path pairs from the test expression Expr_0 are used to compute the result. The test expression Expr_0 might yield a non-empty set of constructed nodes $\mathcal{N}_0$. The system does not need to preserve the contents of the constructed nodes in the set $\mathcal{N}_0$. As long as they exist, the truth value is true, so the system only preserves the set of projection paths $\mathit{UpCtrlPathSet}(\mathcal{N}_0)$. Note that the path pair $\langle \varepsilon, \mathit{RtnUsedPathSet}(\mathcal{P}_0) \cup \mathit{UpCtrlPathSet}(\mathcal{N}_0) \rangle$ accommodates special cases when both the then- and the else-expression are either literals or empty sequences. Admittedly contrived, an example expression is

$$\mathit{if}\ (\$x/\mathit{Address}/\mathit{City} = \text{"Denton"})\ \texttt{then}\ \text{"living in Denton"}\ \texttt{else}\ \text{"not living in Denton"}$$

This expression prints "living in Denton" or "not living in Denton," depending on the path

doc("dir.xml")/Directory/Person/Address/City

Suppose the variable x is bound to doc("dir.xml")/Directory/Person. If the path pair $\langle \varepsilon, \mathit{RtnUsedPathSet}(\mathcal{P}_0) \cup \mathit{UpCtrlPathSet}(\mathcal{N}_0) \rangle$ is omitted from the rule, than the projection analysis of the above example expression will produce $\langle \emptyset, \emptyset \rangle$ instead of

$$\langle \{\langle \varepsilon, \mathit{doc}(\text{"dir.xml"})/\mathit{Directory}/\mathit{Person}/\mathit{Address}/\mathit{City} \rangle\}, \emptyset \rangle$$

Consequently, the system will always print "not living in Denton".

3.5.11. Element Constructors with QNames

In order to process element constructors with QNames, a new judgement called a copy judgement is introduced:

$$Store_1 \vdash \text{copy } \mathcal{N} \text{ to } \mathcal{N}', Store_2$$

Both $\mathcal{N}$ and $\mathcal{N}'$ denote sets of constructed nodes. This judgement does not use the environment *Env*, so it is omitted. The copy operation works as follows:

For each $N \in \mathcal{N}$,

(1) if N has no parent i.e. $Store_1.Parent(N) = nil$,

- (a) no need to make a copy.
- (b) record N as a member of $\mathcal{N}'$.

(2) if N has parent i.e. $Store_1.Parent(N) \neq nil$,

- (a) make a copy of the subtree rooted at N.
- (b) allocate fresh node identifiers to all the nodes in the copy.

  The new store $Store_2$ includes the old store $Store_1$ plus the newly copied subtree.
- (c) call the copy of the node N as N'. set

  $Store_2.Parent(N') = nil$.
- (d) In order to remember all the projection paths needed to construct N', set

$$\begin{aligned} Store_2.CtrlPathSet(N') \quad &= \quad Store_1.CtrlPathSet(N) + \\ &\qquad Store_1.UpCtrlPathSet(N) \end{aligned}$$

- (e) Record N' as a member of $\mathcal{N}'$.

The copy judgement is used in the inference rule of element constructors because the same previously constructed nodes could be used to construct different new nodes. For example, consider the XQuery expression in Listing 3.9.If the copy judgement is omitted, the con-

Listing 3.9. The Copy Judgement is Needed in the Inference Rule of Element Constructors

```
let $z := document {
    let $x := <H>foo bar</H>
   return
      <A>
         <B>
             let $y := doc("C.xml")/C
             return
                 <C> {$y/D, $x} </C>
         </B>
         <E>
             let $y := doc("F.xml")/F
             return
                 <F> {$y/G, $x} </F>
         </E>
      </A>
}
return $z/A/E/F
```

structed node H will have to remember two parents. Consequently, the system will be unable to determine which of the following two projection paths should be excluded from the final set of projection paths.

(1) doc("C.xml")/C/D

(2) doc("F.xml")/F/G

The following is the inference rule of element constructors with QNames.

$$\frac{\begin{array}{l} \mathit{Store}_0, \mathit{Env} \vdash \mathit{Expr} \Rightarrow \langle \mathcal{P}, \mathcal{N} \rangle, \mathit{Store}_1 \\ \quad \mathit{Store}_1 \vdash \text{copy } \mathcal{N} \text{ to } \{N_j \mid j = 1, \ldots, l\}, \mathit{Store}_2 \\ \qquad eN \notin \mathrm{dom}\,(\mathit{Store}_2) \text{ Create a new element node.} \\ \quad \mathit{Store}_3 = \mathit{Store}_2 + \mathit{NodeName}(eN \Rightarrow \mathit{QName}) \\ \qquad\qquad + \mathit{NodeKind}(eN \Rightarrow \text{"element"}) + \mathit{Parent}(eN \Rightarrow \mathit{nil}) \\ \qquad\qquad + \mathit{CtrlPathSet}(eN \Rightarrow \emptyset) \\ \qquad\qquad + \mathit{ConstructedChildren}\,(eN \Rightarrow \{N_{j,} \mid j = 1, \ldots, l\}) \\ \qquad\qquad + \mathit{SourcePathPairSet}\,(eN \Rightarrow \mathcal{P}) \\ \qquad\qquad + \sum_{j=1}^{l} \mathit{Parent}(N_j \Rightarrow eN) \end{array}}{\mathit{Store}_0, \mathit{Env} \vdash \text{element } \mathit{QName}\ \{\mathit{Expr}\} \Rightarrow \langle \emptyset, \{eN\} \rangle, \mathit{Store}_3}$$

3.5.12. Element Constructors with Name Expressions

Since the names are computed by a name expression, it is generally impossible to know the name of the element node that is being constructed without actually executing the corresponding XQuery query. The system assigns a special value **unknown** to the node name in this situation. No further navigation can be done on the projection paths of a name expression. The system needs to remember only the set of all projection paths that are needed to compute the name expression, which is $\mathit{FinalizedPathSet}(\mathcal{P}_0, \mathcal{N}_0)$. The system could remember the set as used paths of each returned path in the set $\mathit{RtnPathSet}(\mathcal{P}_1)$ and control paths of each node in the set $\{N_j \mid j = 1, \ldots, l\}$. However, it is more efficient to remember the set as control paths of the newly created node eN. The following is the inference rule of element constructors with name expressions. Please refer to the previous subsection that discusses the copy judgement.

$$\frac{\begin{array}{l} Store_0, Env \vdash Expr_0 \Rightarrow \langle \mathcal{P}_0, \mathcal{N}_0 \rangle, Store_1 \\ Store_1, Env \vdash Expr_1 \Rightarrow \langle \mathcal{P}_1, \mathcal{N}_1 \rangle, Store_2 \\ \quad Store_2 \vdash \text{copy } \mathcal{N}_1 \text{ to } \{N_j \mid j = 1, \ldots, l\}, Store_3 \\ \quad\quad eN \notin \mathrm{dom}(Store_3) \text{ Create a new element node.} \\ \quad Store_4 = Store_3 + NodeName(eN \Rightarrow \textbf{unknown}) \\ \quad\quad\quad + NodeKind(eN \Rightarrow \text{"element"}) + Parent(eN \Rightarrow nil) \\ \quad\quad\quad + CtrlPathSet(eN \Rightarrow FinalizedPathSet(\mathcal{P}_0, \mathcal{N}_0)) \\ \quad\quad\quad + ConstructedChildren(eN \Rightarrow \{N_{j,} \mid j = 1, \ldots, l\}) \\ \quad\quad\quad + SourcePathPairSet(eN \Rightarrow \mathcal{P}_1) \\ \quad\quad\quad + \sum_{j=1}^{l} Parent(N_j \Rightarrow eN) \end{array}}{Store_0, Env \vdash \text{element } \{Expr_0\}\{Expr_1\} \Rightarrow \langle \emptyset, \{eN\} \rangle, Store_4}$$

3.5.13. Attribute Constructors with QNames

The system does not make a copy of the nodes constructed by the corresponding content expression since no further navigation can be done on an attribute node. Thus, there is no need to use the copy judgement in the inference rule of attribute constructors with QNames. The system needs to remember all the projection paths that are required to compute the content expression, using the following set.

$$\mathcal{P} \cup TreePathPairSet(\mathcal{N}) \cup \{\langle \varepsilon, AllCtrlPathSet(\mathcal{N}) \rangle\}$$

The above set could be replaced by the one below.

$$\mathcal{P} \cup \{\langle S, \mathcal{H} \cup AllCtrlPathSet(N) \rangle \mid \langle S, \mathcal{H} \rangle \in TreePathPairSet(\mathcal{N})\}$$

However, the former is more efficient, although the latter is more natural.

The inference rule for attribute constructors with QNames is as follows.

$$\frac{\begin{array}{l} \mathit{Store}_0, \mathit{Env} \vdash \mathit{Expr} \Rightarrow \langle \mathcal{P},\ \mathcal{N} \rangle,\ \mathit{Store}_1 \\ \qquad aN \notin \mathrm{dom}\,(\mathit{Store}_1) \text{ Create a new attribute node.} \\ \quad \mathit{Store}_2 = \mathit{Store}_1 + \mathit{NodeName}(aN \Rightarrow \mathit{QName}) \\ \qquad\qquad + \mathit{NodeKind}(aN \Rightarrow \text{"attribute"}) + \mathit{Parent}(aN \Rightarrow \mathit{nil}) \\ \qquad\qquad + \mathit{CtrlPathSet}(aN \Rightarrow \emptyset) \\ \qquad\qquad + \mathit{ConstructedChildren}\,(aN \Rightarrow \emptyset) \\ \qquad\qquad + \mathit{SourcePathPairSet}\,(aN \Rightarrow \mathcal{P} \cup \mathit{TreePathPairSet}(\mathcal{N}) \\ \qquad\qquad\qquad\qquad\qquad \cup \{\langle \varepsilon,\ \mathit{AllCtrlPathSet}(\mathcal{N}) \rangle\}) \end{array}}{\mathit{Store}_0, \mathit{Env} \vdash \text{attribute } \mathit{QName}\ \{\mathit{Expr}\} \Rightarrow \langle \emptyset,\ \{aN\} \rangle,\ \mathit{Store}_2}$$

Since no further navigation analysis should be done on an attribute node, the rule could be simplified as follows.

$$\frac{\begin{array}{l} \mathit{Store}_0, \mathit{Env} \vdash \mathit{Expr} \Rightarrow \langle \mathcal{P},\ \mathcal{N} \rangle,\ \mathit{Store}_1 \\ \qquad aN \notin \mathrm{dom}\,(\mathit{Store}_1) \text{ Create a new attribute node.} \\ \quad \mathit{Store}_2 = \mathit{Store}_1 + \mathit{NodeName}(aN \Rightarrow \mathit{QName}) \\ \qquad\qquad + \mathit{NodeKind}(aN \Rightarrow \text{"attribute"}) + \mathit{Parent}(aN \Rightarrow \mathit{nil}) \\ \qquad\qquad + \mathit{CtrlPathSet}(aN \Rightarrow \emptyset) \\ \qquad\qquad + \mathit{ConstructedChildren}\,(aN \Rightarrow \emptyset) \\ \qquad\qquad + \mathit{SourcePathPairSet}\,(aN \Rightarrow \{\langle \varepsilon,\ \mathit{FinalizedPathSet}(\mathcal{P},\ \mathcal{N}) \rangle\}) \end{array}}{\mathit{Store}_0, \mathit{Env} \vdash \text{attribute } \mathit{QName}\ \{\mathit{Expr}\} \Rightarrow \langle \emptyset,\ \{aN\} \rangle,\ \mathit{Store}_2}$$

3.5.14. Attribute Constructors with Name Expressions

Similar to the inference rule of element constructors with name expressions, it is generally impossible to know the name of the attribute node being constructed without actually executing the corresponding XQuery query. Again, the system assigns a special value **unknown** to the node name in this situation. No further navigation can be done on an attribute node. The inference rule of attribute constructors with name expressions is similar to the inference rule of attribute constructors with QNames. The difference is that system also needs to remember the set of all the projection paths that are needed to compute the name expression, which is *FinalizedPathSet*$(\mathcal{P}_0, \mathcal{N}_0)$, in addition to the set of the projection paths that are needed to compute the content expression.

The inference rule for attribute constructors with name expressions is as follows.

$$\frac{\begin{array}{l} \mathit{Store}_0, \mathit{Env} \vdash \mathit{Expr}_0 \Rightarrow \langle \mathcal{P}_0, \mathcal{N}_0 \rangle, \mathit{Store}_1 \\ \mathit{Store}_1, \mathit{Env} \vdash \mathit{Expr}_1 \Rightarrow \langle \mathcal{P}_1, \mathcal{N}_1 \rangle, \mathit{Store}_2 \\ \qquad aN \notin \mathrm{dom}\,(\mathit{Store}_2) \text{ Create a new attribute node.} \\ \quad \mathit{Store}_3 = \mathit{Store}_2 + \mathit{NodeName}(aN \Rightarrow \textbf{unknown}) \\ \qquad\qquad + \mathit{NodeKind}(aN \Rightarrow \text{"attribute"}) + \mathit{Parent}(aN \Rightarrow \mathit{nil}) \\ \qquad\qquad + \mathit{CtrlPathSet}(aN \Rightarrow \mathit{FinalizedPathSet}(\mathcal{P}_0, \mathcal{N}_0)) \\ \qquad\qquad + \mathit{ConstructedChildren}\,(aN \Rightarrow \emptyset) \\ \qquad\qquad + \mathit{SourcePathPairSet}\,(aN \Rightarrow \{\langle \varepsilon, \mathit{FinalizedPathSet}(\mathcal{P}_1, \mathcal{N}_1) \rangle\}) \end{array}}{\mathit{Store}_0, \mathit{Env} \vdash \text{element } \{\mathit{Expr}_0\}\{\mathit{Expr}_1\} \Rightarrow \langle \emptyset, \{aN\} \rangle, \mathit{Store}_3}$$

3.5.15. Document Constructors

The inference rule for document constructors is similar to the inference rule of element constructors with QNames. This rule needs to have the copy judgement for the same reason that the inference rule of element constructors needs it. A constructed document node has no name, so the system assigns an empty string as its node name.

$$\frac{\begin{array}{l} \mathit{Store}_0, \mathit{Env} \vdash \mathit{Expr} \Rightarrow \langle \mathcal{P}, \mathcal{N} \rangle, \mathit{Store}_1 \\ \quad \mathit{Store}_1 \vdash \text{copy } \mathcal{N} \text{ to } \{N_j \mid j = 1, \ldots, l\}, \mathit{Store}_2 \\ \quad\quad dN \notin \mathrm{dom}(\mathit{Store}_2) \text{ Create a new document node.} \\ \quad \mathit{Store}_3 = \mathit{Store}_2 + \mathit{NodeName}(dN \Rightarrow \text{""}) \\ \quad\quad\quad + \mathit{NodeKind}(dN \Rightarrow \text{"document"}) + \mathit{Parent}(dN \Rightarrow \mathit{nil}) \\ \quad\quad\quad + \mathit{CtrlPathSet}(dN \Rightarrow \emptyset) \\ \quad\quad\quad + \mathit{ConstructedChildren}(dN \Rightarrow \{N_{j,} \mid j = 1, \ldots, l\}) \\ \quad\quad\quad + \mathit{SourcePathPairSet}(dN \Rightarrow \mathcal{P}) \\ \quad\quad\quad + \sum_{j=1}^{l} \mathit{Parent}(N_j \Rightarrow dN) \end{array}}{\mathit{Store}_0, \mathit{Env} \vdash \text{document } \{\mathit{Expr}\} \Rightarrow \langle \emptyset, \{dN\} \rangle, \mathit{Store}_3}$$

3.5.16. Text Constructors

A text constructor creates a text node without using any source nodes from the data sources. Since a text node has no name, the system assigns an empty string as its node name. The inference rule of text constructors is as follows.

$$\frac{\begin{array}{l} \quad tN \notin \mathrm{dom}(\mathit{Store}_0) \text{ Create a new text node.} \\ \mathit{Store}_1 = \mathit{Store}_0 + \mathit{NodeName}(tN \Rightarrow \text{""}) \\ \quad\quad + \mathit{NodeKind}(tN \Rightarrow \text{"text"}) + \mathit{Parent}(tN \Rightarrow \mathit{nil}) \\ \quad\quad + \mathit{CtrlPathSet}(tN \Rightarrow \emptyset) \\ \quad\quad + \mathit{ConstructedChildren}(tN \Rightarrow \emptyset) \\ \quad\quad + \mathit{SourcePathPairSet}(tN \Rightarrow \emptyset) \end{array}}{\mathit{Store}_0, \mathit{Env} \vdash \text{text } \{\mathit{Stirng}\} \Rightarrow \langle \emptyset, \{tN\} \rangle, \mathit{Store}_1}$$

3.5.17. Wrapping Up

At the end of the inference procedure, the system encounters the following rule:

$$\mathit{Store}_0, \mathit{Env} \vdash \mathit{Expr} \Rightarrow \langle \mathcal{P}, \mathcal{N} \rangle, \mathit{Store}_1$$

The final projected document is represented by the set of all projection paths both in $\mathcal{P}$ and $\mathcal{N}$, i.e. $\mathit{FinalizedPathSet}(\mathcal{P}, \mathcal{N})$. Note that the system has appended a # to every $R \in \mathit{RtnPathSet}(\mathcal{P})$ and $S \in \mathit{RtnPathSet}(\mathit{TreePathPairSet}(\mathcal{N}))$ to signify that it needs to retrieve their descendants.

3.6. Soundness

The language used to present the extended XML doument projection algorithm is called structural operational semantics [67]. The inference rules are based on the syntax of the XQuery core expressions that are used by the system. The inference rules actually constitute a proof system. Every proof of this system verifies that some judgement holds in the form:

$$\mathit{Store}_0, \mathit{Env} \vdash \mathit{Expr} \Rightarrow \langle \mathcal{P}, \mathcal{N} \rangle, \mathit{Store}_1$$

The specific interpretation of this judgement is explained in the section 3.3 on page 47. In order to show the correctness of the extended projection algorithm, I need to prove the soundness of my proof system. That is, I need to show that any judgement proved by the proof system has the following properties:

(1) The projection path set $\mathit{FinalizedRtnPathSet}(\mathcal{P})$ includes all the source nodes that are part of the answer to the corresponding expression *Expr*.

(2) The constructed node set $\mathcal{N}$ includes all the constructed nodes that are part of the answer to the corresponding expression *Expr*.

(3) The projection path set $\mathit{FinalizedPathSet}(\mathcal{P}, \mathcal{N})$ includes all the source nodes that are needed to answer the corresponding expression *Expr*.

Note that the projection path sets might include more source nodes than necessary, but they should never omit any required nodes.

The technique I use to prove the soundness of the inference rules used by the system is structural induction [76]. The more familiar natural-number induction is a special form of structural induction, while structural induction is a special form of well-founded induction [56]. Well-founded induction is induction on a well-founded relation. A well-founded relation is a binary relation R on a set S with the following property: any non-empty subset of the set S has a minimal element under the relation R [56, 82, 41]. The subexpression relation on the set of all XQuery core expressions is a well-founded relation, as is the subproof relation on the set of all the proofs generated by my inference rules.

Marian and Siméon proved the correctness of their algorithm by induction on the inference rules for each expression. The final set of projection paths generated for an XQuery expression by the system described in this dissertation excludes some paths from the final set, only when the algorithm concludes that those paths will not affect the final results. In a sense, the algorithm used in the research system does a double projection. It does projection on both real input documents and virtual constructed elements or documents simultaneously, while the Marian and Siméon algorithm does projection on only real input documents. It is the projection on the virtual constructed elements or documents that enable the extended algorithm to exclude some unnecessary paths that are included by the Marian and Siméon algorithm. The extended algorithm is able to do projection on the virtual constructed elements or documents because their structures are represented by a forest of constructed nodes in memory. Intuitively, the fundamental ideas of the extended algorithm can never be wrong if the algorithm of Marian and Siméon is correct.

The inference rules by the system constitute a proof system. So the structural induction is induction on proofs or inference rules. It takes two steps to apply structural induction to prove a property. First, one must prove that the property holds for the base cases, and then, by the induction steps, show that the property is correct. The base cases for the extended proof system are the inference rules for atomic expressions. The induction steps deal with the inference rules for compound expressions.

Theorem. *If the inference rules of projection analysis prove that*

$$Store_0, Env \vdash Expr \Rightarrow \langle \mathcal{P}, \mathcal{N} \rangle, Store_1$$

then the following properties hold:

(1) *The projection path set FinalizedRtnPathSet($\mathcal{P}$) includes all the source nodes that are part of the answer to the corresponding expression Expr.*

(2) *The constructed node set $\mathcal{N}$ includes all the constructed nodes that are part of the answer to the corresponding expression Expr.*

(3) *The projection path set FinalizedPathSet($\mathcal{P}$, $\mathcal{N}$) includes all the source nodes that are needed to answer the corresponding expression Expr.*

Proof. I just need to prove the properties for every inference rule that is in the system. Note that every symbol in the following proof refers to the same entity as the corresponding inference rule presented in the section 3.5.

Literal Values: A literal value expression evaluates to the literal value.

(1) The answer to a literal value expression includes no source node. Thus the projection path set *FinalizedRtnPathSet*($\emptyset$) = $\emptyset$ includes all the source nodes that are part of the answer to the literal value expression.

(2) The answer to a literal value expression includes no constructed node. Thus the constructed node set $\emptyset$ includes all the constructed nodes that are part of the answer to the literal value expression.

(3) The evaluation of a literal value expression does not need any source nodes. Thus the projection path set *FinalizedPathSet*($\emptyset$, $\emptyset$) = $\emptyset$ includes all the source nodes that are needed to answer the literal value expression.

Empty Sequence: An empty sequence expression evaluates to an empty sequence. Similar to the arguments given for a literal value expression, all three properties hold for an empty sequence expression.

Root Path: A root path expression evaluates to the root node.

(1) The answer to a root path expression includes just the root node. Thus the projection path set *FinalizedRtnPathSet*($\{\langle /, \emptyset\rangle\}$) = $\{/\#\}$ includes all the source nodes that are part of the answer to the root path expression.

(2) The answer to a root path expression includes no constructed node. Thus the constructed node set $\emptyset$ includes all the constructed nodes that are part of the answer to the root path expression.

(3) The evaluation of a root path expression needs only the whole source tree. Thus the projection path set

$$\mathit{FinalizedPathSet}(\{\langle /, \emptyset\rangle\}, \emptyset) = \{/\#\}$$

includes all the source nodes that are needed to answer the root path expression.

Sequence Expressions: A sequence expression evaluates to a sequence of the nodes returned by either subexpression.

(1) The answer to a sequence expression includes only the source nodes that are part of the answer to either subexpression. We have

$$\begin{aligned}\mathit{FinalizedRtnPathSet}(\mathcal{P}_1 \cup \mathcal{P}_2) &= \mathit{FinalizedRtnPathSet}(\mathcal{P}_1) \cup \\ & \mathit{FinalizedRtnPathSet}(\mathcal{P}_2)\end{aligned}$$

By the induction hypothesis, the sets $\mathit{FinalizedRtnPathSet}(\mathcal{P}_1)$ and $\mathit{FinalizedRtnPathSet}(\mathcal{P}_2)$ include all the source nodes that are part of the answers to the corresponding subexpressions Expr_1 and Expr_2 respectively. Thus the projection path set $\mathit{FinalizedRtnPathSet}(\mathcal{P}_1 \cup \mathcal{P}_2)$ includes all the source nodes that are part of the answer to the sequence expression.

(2) The answer to a sequence expression includes only the constructed nodes that are part of the answer to either subexpression. By the induction hypothesis, the sets $\mathcal{N}_1$ and $\mathcal{N}_2$ include all the constructed nodes that are part of the answers to the corresponding subexpressions Expr_1 and Expr_2 respectively. Thus the constructed node set $\mathcal{N}_1 \cup \mathcal{N}_2$ includes all the constructed nodes that are part of the answer to the sequence expression.

(3) The evaluation of a sequence expression needs only the source data required by its subexpressions. By the induction hypothesis, the projection path sets

$$\mathit{FinalizedPathSet}(\mathcal{P}_1, \mathcal{N}_1)$$

and

$$\mathit{FinalizedPathSet}(\mathcal{P}_2, \mathcal{N}_2)$$

include all the source nodes that are needed to answer the subexpressions Expr_1 and Expr_2 respectively. The final expression is

$$\begin{aligned}\mathit{FinalizedPathSet}(\mathcal{P}_1 \cup \mathcal{P}_2, \mathcal{N}_1 \cup \mathcal{N}_2) &= \mathit{FinalizedPathSet}(\mathcal{P}_1, \mathcal{N}_1) \cup \\ &\quad \mathit{FinalizedPathSet}(\mathcal{P}_2, \mathcal{N}_2)\end{aligned}$$

Thus the above projection path set includes all the source nodes that are needed to answer the sequence expression.

Function Calls:

Calls to User Defined Functions: The evaluation of a call to a user defined function without any parameter is equivalent to the evaluation of the expression inside the function body. Thus, if the three properties hold for the judgement corresponding to the function body expression Expr_0, the three properties hold for the judgement corresponding to the function call.

In a previous section 3.5.5.1 on page 56 , I show that a user defined function with parameters is equivalent to a series of nested `let` expressions in the subsection. The reader should refer to the proof for the inference rule of `let` expressions for the proof for a call to a user defined function with parameters.

Input Functions: An input function expression evaluates to the corresponding document node.

(1) The answer to an input function expression includes just the document node. Thus the projection path set

$$\mathit{FinalizedRtnPathSet}(\{\langle doc(URL), \emptyset\rangle\}) = \{doc(URL)\#\}$$

includes all the source nodes that are part of the answer to the input function expression.

(2) The answer to an input function expression includes no constructed node. Thus the constructed node set $\emptyset$ includes all the constructed nodes that are part of the answer to the input function expression.

(3) The evaluation of an input function expression needs only the document tree. Thus the projection path set

$$\mathit{FinalizedPathSet}(\{\langle doc(URL), \emptyset\rangle\}, \emptyset) = \{doc(URL)\#\}$$

includes all the source nodes that are needed to answer the input function expression.

Comparison Expressions: A comparison expression evaluates to a Boolean value. For convenience, let

$$\mathcal{A} = \mathit{FinalizedPathSet}(\mathcal{P}_1, \mathcal{N}_1) \cup \mathit{FinalizedPathSet}(\mathcal{P}_2, \mathcal{N}_2)$$

(1) The answer to a comparison expression includes no source node. Thus the projection path set $\mathit{FinalizedRtnPathSet}(\{\langle \varepsilon, \mathcal{A}\rangle\}) = \{\varepsilon\#\} = \emptyset$ includes all the source nodes that are part of the answer to the comparison expression.

(2) The answer to a comparison expression includes no constructed node. Thus the constructed node set $\emptyset$ includes all the constructed nodes that are part of the answer to the comparison expression.

(3) The evaluation of a comparison expression needs only the source data needed by its subexpressions. By the induction hypothesis, the projection path sets

$$\mathit{FinalizedPathSet}(\mathcal{P}_1, \mathcal{N}_1)$$

and

$$\mathit{FinalizedPathSet}(\mathcal{P}_2, \mathcal{N}_2)$$

include all the source nodes that are needed to answer the subexpressions Expr_1 and Expr_2 respectively. The final expression is

$$\begin{aligned} & \mathit{FinalizedPathSet}(\{\langle \varepsilon, \mathcal{A} \rangle\}, \emptyset) \\ = & \{\varepsilon\#\} \cup \mathcal{A} \\ = & \mathcal{A} \\ = & \mathit{FinalizedPathSet}(\mathcal{P}_1, \mathcal{N}_1) \cup \mathit{FinalizedPathSet}(\mathcal{P}_2, \mathcal{N}_2) \end{aligned}$$

Thus the above projection path set includes all the source nodes that are needed to answer the comparison expression.

Arithmetic Expressions: An arithmetic expression evaluates to an atomic value, e.g. an integer, a double, a date, etc.. Similar to the arguments given for a comparison expression, all three properties hold for an arithmetic expression.

Union Expressions: A union expression evaluates to the union of the nodes returned by either subexpression. Similar to the arguments given for a sequence expression, all three properties hold for a union expression.

Variable References: A variable reference expression evaluates to the source nodes and constructed nodes that are bound to the variable. A variable may be bound in two types of expressions: `for` expressions or `let` expressions. It is bound to the nodes in the answers of the binding subexpressions Expr_1's of `for` expressions or `let` expressions. Both rules bind the variable to the set of projection paths $\mathit{RtnPathSet}(\mathcal{P}_1)$ and the set of constructed nodes $\mathcal{N}_1$. By the induction hypothesis, the sets $\mathit{RtnPathSet}(\mathcal{P}_1)$ and $\mathcal{N}_1$ include all the nodes that are part of the answers to the binding subexpressions Expr_1's. In the inference rule of variable references, those nodes are retrieved. Thus,

- The set $\{R_i | i = 1, \ldots, n\}$ in the inference rule of variable references is equivalent to the set $\mathit{RtnPathSet}(\mathcal{P}_1)$ in the inference rules of `for` expressions or `let` expressions.
- The set $\mathcal{N}$ in the inference rule of variable references is equivalent to the set $\mathcal{N}_1$ in the inference rules of `for` expressions or `let` expressions.

(1) The projection path set $\mathit{FinalizedRtnPathSet}(\{\langle R_i, \emptyset\rangle \mid i = 1, \ldots, n\}) = \{R_i\# \mid i = 1, \ldots, n\}$ includes all the source nodes that are part of the answer to the variable reference expression.

(2) The constructed node set $\mathcal{N}$ includes all the constructed nodes that are part of the answer to the variable reference expression.

(3) The projection path set $\mathit{FinalizedPathSet}(\{\langle R_i, \emptyset\rangle \mid i = 1, \ldots, n\}, \mathcal{N})$ includes all the source nodes that are needed to answer the variable reference expression.

for **Expressions:** A for expression evaluates to a sequence of the nodes returned by the subexpression in the return clause. I will prove the properties hold for the optimized rule of for expressions. If that is successful, the properties should hold for the unoptimized rule of for expressions, since the unoptimized rule always includes more projection paths or nodes. For convenience, let

$$\mathcal{A} = \mathit{UsedPathSet}(\mathcal{P}_1) \cup \mathit{UpCtrlPathSet}(\mathcal{N}_1)$$

(1) The answer to a for expression includes only the source nodes that are part of the answer to the subexpression in the return clause. This produces

$$\begin{aligned}&\mathit{FinalizedRtnPathSet}(\{\langle R_{2,i}, \mathcal{U}_{2,i} \cup \mathcal{A}\rangle \mid i = 1, \ldots, n_2\})\\ &= \{R_{2,i}\# \mid i = 1, \ldots, n_2\}\\ &= \mathit{FinalizedRtnPathSet}(\{R_{2,i} \mid i = 1, \ldots, n_2\})\end{aligned}$$

By the induction hypothesis, the set $\mathit{FinalizedRtnPathSet}(\{R_{2,i} \mid i = 1, \ldots, n_2\})$ includes all the source nodes that are part of the answer to the subexpression in the return clause. Thus the projection path set $\mathit{FinalizedRtnPathSet}(\{\langle R_{2,i}, \mathcal{U}_{2,i} \cup \mathcal{A}\rangle \mid i = 1, \ldots, n_2\})$ includes all the source nodes that are part of the answer to the for expression.

(2) The answer to a for expression includes only the constructed nodes that are part of the answer to the subexpression in the return clause. By the induction hypothesis, the set $\{N_{2,j} \mid j = 1, \ldots, l_2\}$ includes all the constructed nodes that are part of the answer to the subexpression in the return clause. Thus the constructed node set $\{N_{2,j} \mid j = 1, \ldots, l_2\}$ includes all the constructed nodes that are part of the answer to the for expression.

(3) By the induction hypothesis, the projection path sets

$$\mathit{FinalizedPathSet}(\mathcal{P}_1, \mathcal{N}_1)$$

and

$$\mathit{FinalizedPathSet}(\{\langle R_{2,i}, \mathcal{U}_{2,i}\rangle \mid i = 1, \ldots, n_2\}, \{N_{2,j} \mid j = 1, \ldots, l_2\})$$

include all the source nodes that are needed to answer the subexpressions $Expr_1$ and $Expr_2$ as *standalone* expressions respectively. This produces the expression

$$\mathit{FinalizedPathSet}(\{\langle R_{2,i}, \mathcal{U}_{2,i}\rangle \mid i = 1, \ldots, n_2\}, \{N_{2,j} \mid j = 1, \ldots, l_2\})$$

$\subseteq \mathit{FinalizedPathSet}(\{\langle R_{2,i}, \mathcal{U}_{2,i} \cup \mathcal{A}\rangle \mid i = 1, \ldots, n_2\}, \{N_{2,j} \mid j = 1, \ldots, l_2\})$

Thus the algorithm has included all the source nodes that are needed to answer the subexpression in the return clause. Now consider the source nodes that are needed to answer the binding subexpression. First, the path set $\mathit{RtnPathSet}(\mathcal{P}_1)$ can be omitted for the reasons given by Marian and Siméon [53]. Second, $\mathcal{N}_1$needs to remember only the paths in the set $\mathit{UpCtrlPathSet}(\mathcal{N}_1)$. If the binding variable does not appear in the the subexpression in the return clause, obviously other paths remembered by $\mathcal{N}_1$ are not needed. If the binding variable does appear in the the subexpression in the return clause, then that or the binding variable references will cause the necessary paths to enter the sets $\{R_{2,i} \mid i = 1, \ldots, n_2\}$ or $\{\mathcal{U}_{2,i} \mid i = 1, \ldots, n_2\}$ or $\{N_{2,j} \mid j = 1, \ldots, l_2\}$. Thus only the paths in the set $\mathit{UsedPathSet}(\mathcal{P}_1) \cup \mathit{UpCtrlPathSet}(\mathcal{N}_1)$ need to be remembered, and they are. Therefore the path set

$$\mathit{FinalizedPathSet}(\{\langle R_{2,i}, \mathcal{U}_{2,i} \cup \mathcal{A}\rangle \mid i = 1, \ldots, n_2\}, \{N_{2,j} \mid j = 1, \ldots, l_2\})$$

includes all the source nodes that are needed to answer the for expression.

`let` **Expressions:** A let expression evaluates to a sequence of the nodes returned by the subexpression in the return clause. Similar to the arguments given for a for expression, all three properties hold for a let expression.

XPath steps: An XPath step expression evaluates to the source nodes and constructed nodes returned by the step that is being applied to the current context node. Its already been proved that the variables are bound to the correct source nodes and constructed nodes by `for` expressions or `let` expressions. Thus the current context node is correctly bound to the projection path set $\{R_1, \cdots, R_n\}$ and constructed node set $\mathcal{N}_1$.

(1) The answer to an XPath step includes only the source nodes returned by the step that is being applied to the current context node. Thus the projection path set

$$FinalizedRtnPathSet(\cup_{i=1}^{n} \{\langle R_i/Axis\ NodeTest,\ \emptyset\rangle \mid R_i \text{ is not } \varepsilon\} \cup \mathcal{P}_2)$$

includes all the source nodes that are part of the answer to the XPath step expression.

(2) The answer to an XPath step expression includes only constructed nodes returned by the step that is being applied to the current context node. Applying the step to projection paths of $\{R_1, \cdots, R_n\}$ will not result in any constructed nodes. All the constructed nodes are the result of applying the step to the constructed nodes of the set $\mathcal{N}_1$. Thus the constructed node set $\mathcal{N}_2$ includes all the constructed nodes that are part of the answer to the XPath step expression.

(3) This algorithm does not exclude any paths in the set $\{R_1, \cdots, R_n\}$ that are not the special empty path ε. It does exclude some constructed nodes in the set $\mathcal{N}_1$, but only when it is absolutely certain that those nodes will produce only empty sequences when this step is applied. Thus the projection path set

$$FinalizedPathSet(\cup_{i=1}^{n} \{\langle R_i/Axis\ NodeTest,\ \emptyset\rangle \mid R_i \text{ is not } \varepsilon\} \cup \mathcal{P}_2,\ \mathcal{N}_2)$$

includes all the source nodes that are needed to answer the XPath step expression.

Conditional Expressions: A conditional expression evaluates to a sequence of the nodes returned by either the then-expression or the else-expression.The arguments for the then-expression and the else-expression are similar to the ones for sequence expressions. For convenience, let

$$\begin{aligned}\mathcal{P} &= \cup_{k=1}^{2} \cup_{i=1}^{n_k} \{\langle R_{k,i},\, \mathcal{U}_{k,i} \cup \mathit{RtnUsedPathSet}(\mathcal{P}_0) \cup \mathit{UpCtrlPathSet}(\mathcal{N}_0)\rangle\} \\ &\quad \cup \{\langle \varepsilon,\, \mathit{RtnUsedPathSet}(\mathcal{P}_0) \cup \mathit{UpCtrlPathSet}(\mathcal{N}_0)\rangle\} \\ \mathcal{N} &= \{N_{1,j} \mid j = 1, \ldots, l_1\} \cup \{N_{2,j} \mid j = 1, \ldots, l_2\}\end{aligned}$$

(1) The answer to a conditional expression includes only the source nodes that are part of the answer to either the then-expression or the else-expression. Thus, the expression is

$$\begin{aligned}&\mathit{FinalizedRtnPathSet}(\mathcal{P}) \\ &= \cup_{i=1}^{n_1} \{R_{1,i}\#\} \cup \cup_{i=1}^{n_2} \{R_{2,i}\#\} \cup \{\varepsilon\#\} \\ &= \mathit{FinalizedRtnPathSet}(\cup_{i=1}^{n_1} \{\langle R_{1,i},\, \mathcal{U}_{1,i}\rangle\}) \cup \mathit{FinalizedRtnPathSet}(\cup_{i=1}^{n_2} \{\langle R_{2,i},\, \mathcal{U}_{2,i}\rangle\})\end{aligned}$$

By the induction hypothesis, the sets $\mathit{FinalizedRtnPathSet}(\cup_{i=1}^{n_1} \{\langle R_{1,i},\, \mathcal{U}_{1,i}\rangle\})$ and $\mathit{FinalizedRtnPathSet}(\cup_{i=1}^{n_2} \{\langle R_{2,i},\, \mathcal{U}_{2,i}\rangle\})$ include all the source nodes that are part of the answers to the then-expression and the else-expression respectively. Thus the projection path set $\mathit{FinalizedRtnPathSet}(\mathcal{P})$ includes all the source nodes that are part of the answer to the conditional expression.

(2) The answer to a conditional expression includes only the constructed nodes that are part of the answer to either the then-expression or the else-expression. By the induction hypothesis, the sets $\{N_{1,j} \mid j = 1, \ldots, l_1\}$ and $\{N_{2,j} \mid j = 1, \ldots, l_2\}$ include all the constructed nodes that are part of the answers to the then-expression or the else-expression respectively. Thus the constructed node set $\mathcal{N}$ includes all the constructed nodes that are part of the answer to the conditional expression.

(3) The evaluation of a conditional expression needs all the source data needed by its subexpressions. By the induction hypothesis, the projection path sets

$$\mathit{FinalizedPathSet}(\{\langle R_{1,i}, \mathcal{U}_{1,i}\rangle \mid i = 1, \ldots, n_1\}, \{N_{1,j} \mid j = 1, \ldots, l_1\})$$

and

$$\mathit{FinalizedPathSet}(\{\langle R_{2,i}, \mathcal{U}_{2,i}\rangle \mid i = 1, \ldots, n_2\}, \{N_{2,j} \mid j = 1, \ldots, l_2\})$$

include all the source nodes that are needed to answer the then-expression or the else-expression respectively. The union of the above two sets are obviously included in the set $\mathit{FinalizedPathSet}(\mathcal{P}, \mathcal{N})$. It remains to show that the set $\mathit{FinalizedPathSet}(\mathcal{P}, \mathcal{N})$ also includes all the source nodes that are needed by the test-expression. If the test-expression is a stand-alone expression, one will need the path set $\mathit{FinalizedPathSet}(\mathcal{P}_0, \mathcal{N}_0)$. As part of a conditional expression, however, it is sufficient to include only the set $\mathit{RtnUsedPathSet}(\mathcal{P}_0) \cup \mathit{UpCtrlPathSet}(\mathcal{N}_0)$, because the test-expression returns only a Boolean value, i.e. it does not return source nodes. Thus the projection path set $\mathit{FinalizedPathSet}(\mathcal{P}, \mathcal{N})$ includes all the source nodes that are needed to answer the conditional expression.

Element Constructors with QNames: An element constructor with a QName evaluates to a constructed element node.

(1) The answer to an element constructor with a QName includes no source nodes at the top level. Thus the projection path set $\mathit{FinalizedRtnPathSet}(\emptyset) = \emptyset$ includes all the source nodes that are part of the answer to the element constructor with a QName.

(2) The answer to an element constructor with a QName includes just this constructed node. Thus the constructed node set $\{eN\}$ includes all the constructed nodes that are part of the answer to the element constructor with a QName.

(3) The evaluation of a element constructor with a QName needs only the source data needed by its content expression. By the induction hypothesis, the projection path set *FinalizedPathSet*$(\mathcal{P}, \mathcal{N})$ includes all the source nodes that are needed to answer the content expression. This algorithm copies $\mathcal{N}$ to $\{N_j \mid j = 1, \ldots, l\}$, which preserves all projection paths remembered by $\mathcal{N}$. The sets $\mathcal{P}$ and $\{N_j \mid j = 1, \ldots, l\}$ are remembered respectively by the sets *SourcePathPairSet* (*eN*) and *ConstructedChildren* (*eN*) of this constructed node. Therefore, the projection path set *FinalizedPathSet*$(\emptyset, \{eN\})$ includes all the source nodes that are needed to answer the content expression and, subsequently, the whole element constructor with a QName.

Element Constructors with Name Expressions: An element constructor with a name expression evaluates to a constructed element node. The arguments are similar to the ones given for an element constructor with a QName. The only difference is that one needs to deal with the name expression when proving the third property.

(1) The answer to an element constructor with a name expression includes zero source nodes at the top level. Thus the projection path set *FinalizedRtnPathSet*$(\emptyset)$ $= \emptyset$ includes all the source nodes that are part of the answer to the element constructor with a name expression.

(2) The answer to an element constructor with a name expression includes just the constructed node. Thus the constructed node set $\{eN\}$ includes all the constructed nodes that are part of the answer to the element constructor with a name expression.

(3) The arguments are the same as those for an element constructor with a QName, when proving the projection path set *FinalizedPathSet*$(\emptyset, \{eN\})$ includes all the source nodes that are needed to answer the content expression. And all the source nodes needed to answer the name expression are in the path set

FinalizedPathSet$(\mathcal{P}_0, \mathcal{N}_0)$, which are remembered by the set *CtrlPathSet* (eN) of the constructed node. Therefore, the projection path set *FinalizedPathSet*$(\emptyset, \{eN\})$ includes all the source nodes that are needed to answer the element constructor with a name expression.

Attribute Constructors with QNames: An attribute constructor with a QName evaluates to a constructed attribute node. The arguments are similar to the ones given for an element constructor with a QName. The only difference is that the source nodes that are required to answer the content expression are remembered in a different way. All the projection paths remembered in various places are collected and then remembered by the set *SourcePathPairSet* (aN) of this constructed node. This is done because there is no further navigation on an attribute node.

Attribute Constructors with Name Expressions: An attribute constructor with a name expression evaluates to a constructed attribute node. The arguments are similar to the ones given for an element constructor with a name expression. The difference between the two is the same as the one between an attribute constructor with a QName and an element constructor with a QName.

Document Constructors: A document constructor evaluates to a constructed document node. The arguments are the same as the ones given for an element constructor with a QName.

Text Constructors: A text constructor evaluates to a constructed text node.

(1) The answer to a text constructor includes no source node. Thus the projection path set *FinalizedRtnPathSet*$(\emptyset) = \emptyset$ includes all the source nodes that are part of the answer to the text constructor.

(2) The answer to a text constructor includes just the constructed node. Thus the constructed node set $\{tN\}$ includes all the constructed nodes that are part of the answer to the text constructor.

(3) No source nodes are required in the evaluation of a text constructor. Thus the projection path set *FinalizedPathSet*$(\emptyset, \{tN\}) = \emptyset$ includes all the source nodes that are needed to answer the text constructor.

$\square$

CHAPTER 4

COMPARISON WITH THE ORIGINAL XML DOCUMENT PROJECTION

4.1. Marian and Siméon's Projection Algorithm

An XML document projection algorithm was first published by Amélie Marian and Jérôme Siméon at VLDB 2003 [52]. Their technical report gives the complete analysis [53]. The intuition behind the research was that most XQuery queries can be answered by examining only part of the original XML document. The authors define this subset of the document as the projection document. The corresponding XPath paths that select the projection part of the original XML document are called projection paths. Only forward navigation without predicates are allowed in projection paths, so the syntax used for projection paths is a subset of XPath.

The authors also developed inference rules to produce the set of projection paths given in response to an XQuery query. Their loading algorithm scans original documents and generates projection documents for an XQuery query according to the corresponding set of projection paths. They then tested their algorithm on the benchmark XMark [75] and showed that their projection enabled main-memory XQuery engines to evaluate queries on significantly larger documents.

4.2. Improvement Achieved by my Extended Projection Algorithm

In 2003, this researcher thought of an idea that was similar to the projection concept defined by Amélie Marian and Jérôme Siméon prior to reading the papers in this area. The ideas presented in this dissertation are the result of work related to developing a procedure to decompose an outsourced user query into atomic and integrating queries for a mediation system. After reading the papers by Amélie Marian and Jérôme Siméon, this researcher was able to see how to extend the original algorithm and show how this could be applied to information integration problems.

Although Marian and Siméon's algorithm is very effective, it was not intended for query processing for information integration systems. Their algorithm deals with most XQuery features, including FLWR expressions and full support for XPath 1.0. In Amélie Marian and Jérôme Siméon's opinion [52], however,

> if a new element is constructed, further navigation does not apply to the input document and must not be taken into account.

The work presented in this dissertation indicates that this statement is incorrect. For example, the projection results from the three evaluation outsourced queries generated by the mediation system developed for this research demonstrate that further navigation analysis of constructed elements or documents can help make projection documents even smaller, although the algorithm may become more complex.

Note that if the 'extended' algorithm is run on XMark benchmark[75], it will get the same results as the Amélie Marian and Jérôme Siméon algorithm since those benchmark queries contain no projections on constructed elements. For outsourced queries in the system developed for this dissertation, however, the extended algorithm always generates better results than the Marian and Siméon algorithm, because the queries generate projections on constructed elements.

Below is a description of the analysis performed for three different sample queries. The first evaluation query is the one that was used throughout the dissertation to explain how the algorithm works. The second and third evaluation queries are designed to show how the algorithm operates when a different number of documents are required to answer the query (large vs. small). Each query was executed using both the Marian and Siméon algorithm (M&S) and the extended projection algorithm on three different data sources: two were relational databases (data was wrapped as lectures.xml and faculty.xml) and one was an

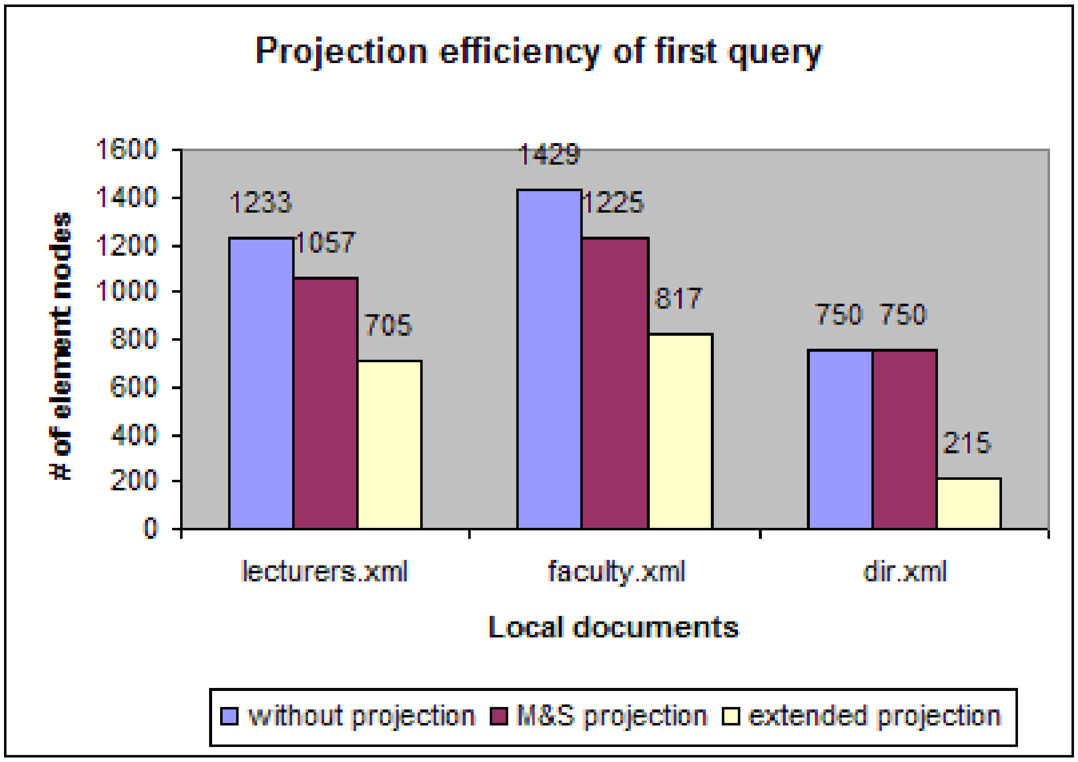


Figure 4.1. Comparing the Numbers of Element Nodes for the First Query

XML file (dir.xml). The analysis program then determined the projection efficiency for each algorithm by measuring the number of element and text nodes retrieved as well as the sizes of the projection documents. This analysis is described in more detail below.

4.2.1. Evaluation on the First Outsourced Query

The outsourced query presented in Listing 2.10 of Section 2.8.2 is the first evaluation query. The reader will recall that this query was presented in previous chapters, and that the exact listing was also presented at that time. The following figures compare the projection paths returned by Marian and Siméon's algorithm and by the extended algorithm. Figure 4.1 on page 97 compares the numbers of element nodes retrieved. Figure 4.2 on page 98 compares the numbers of text nodes retrieved. Figure 4.3 on page 99 compares the sizes of the projection documents retrieved.

(1) The projection paths for lecturers.xml at chemistry department wrapper

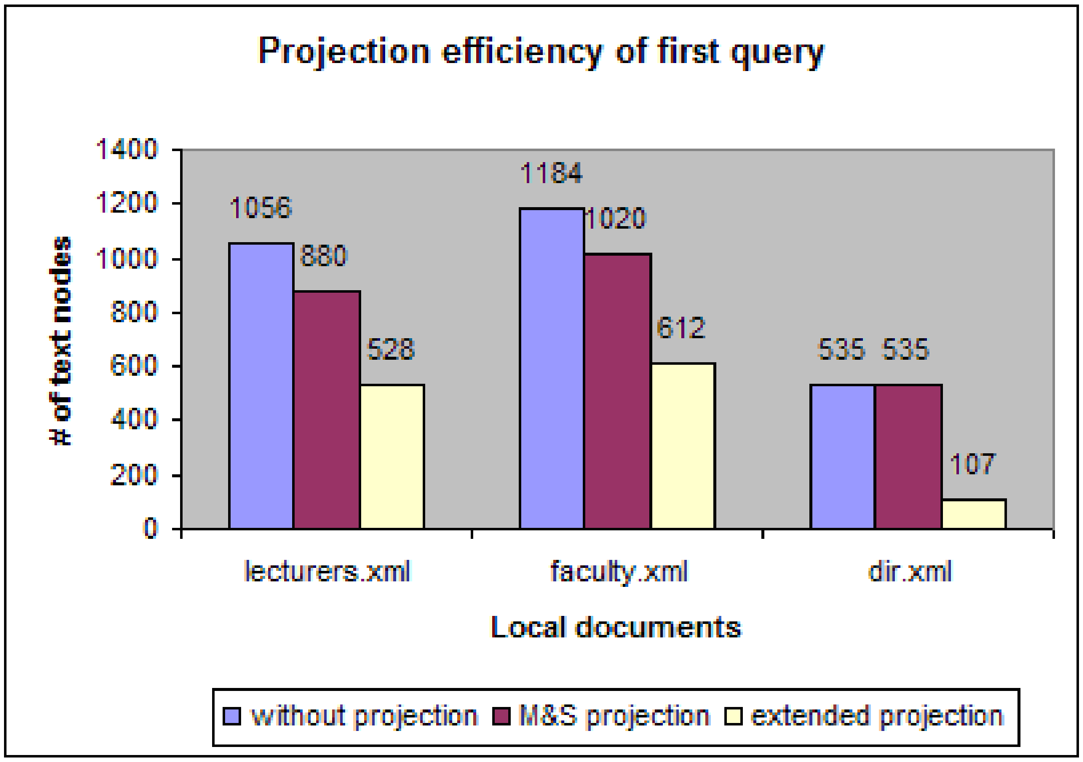


Figure 4.2. Comparing the Numbers of Text Nodes for the First Query

(a) Marian and Siméon's algorithm returns:

  (i) /Lecturers/Row
  (ii) /Lecturers/Row/Name#
  (iii) /Lecturers/Row/Pay_rate#
  (iv) /Lecturers/Row/Monthly_hours#
  (v) /Lecturers/Row/Rank#
  (vi) /Lecturers/Row/Room/text()#

(b) The extended algorithm returns:

  (i) /Lecturers/Row
  (ii) /Lecturers/Row/Name#
  (iii) /Lecturers/Row/Pay_rate#
  (iv) /Lecturers/Row/Monthly_hours#

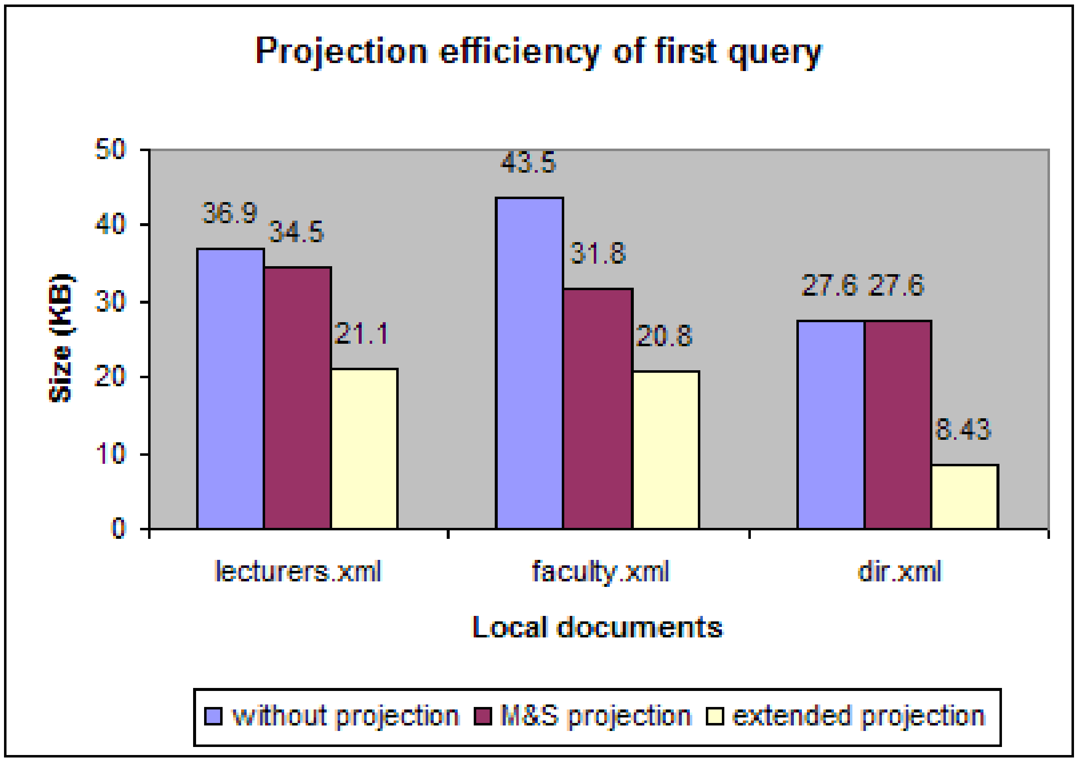


Figure 4.3. Comparing the Projection Sizes for the First Query

(2) The projection paths for faculty.xml at computer science department wrapper

(a) Marian and Siméon's algorithm returns:

(i) /Faculty/Row

(ii) /Faculty/Row/Fname#

(iii) /Faculty/Row/Lname#

(iv) /Faculty/Row/Salary#

(v) /Faculty/Row/Title/text()#

(vi) /Faculty/Row/Office#

(b) The extended algorithm returns:

(i) /Faculty/Row

(ii) /Faculty/Row/Fname#

(iii) /Faculty/Row/Lname#

        (iv) /Faculty/Row/Salary#

(3) The projection paths for dir.xml at the college wrapper

    (a) Marian and Siméon's algorithm returns:

        (i) /Directory/Person

        (ii) /Directory/Person/@Name

        (iii) /Directory/Person/Phone#

        (iv) /Directory/Person/Email#

        (v) /Directory/Person/Address#

    (b) The extended algorithm returns:

        (i) /Directory/Person

        (ii) /Directory/Person/@Name

        (iii) /Directory/Person/Phone#

The # at the end of a projection path is a shorthand notation that indicates that all the descendants of the corresponding nodes in the projection document should be included. If it is missing this symbol, then the descendants of the corresponding nodes are not included.

The following projection paths are the extra paths that are returned by Marian and Siméon's algorithm:

/Lecturers/Row/Rank#

/Lecturers/Row/Room/text()#

/Faculty/Row/Title/text()#

/Faculty/Row/Office#

/Directory/Person/Email#

/Directory/Person/Address#

However, the user is not interested in this information, so the extended algorithm excludes them. The total number of element nodes in the data sources is 3412. The number retrieved by the M&S algorithm is 3032, which is a saving of 11 percent. The number retrieved by

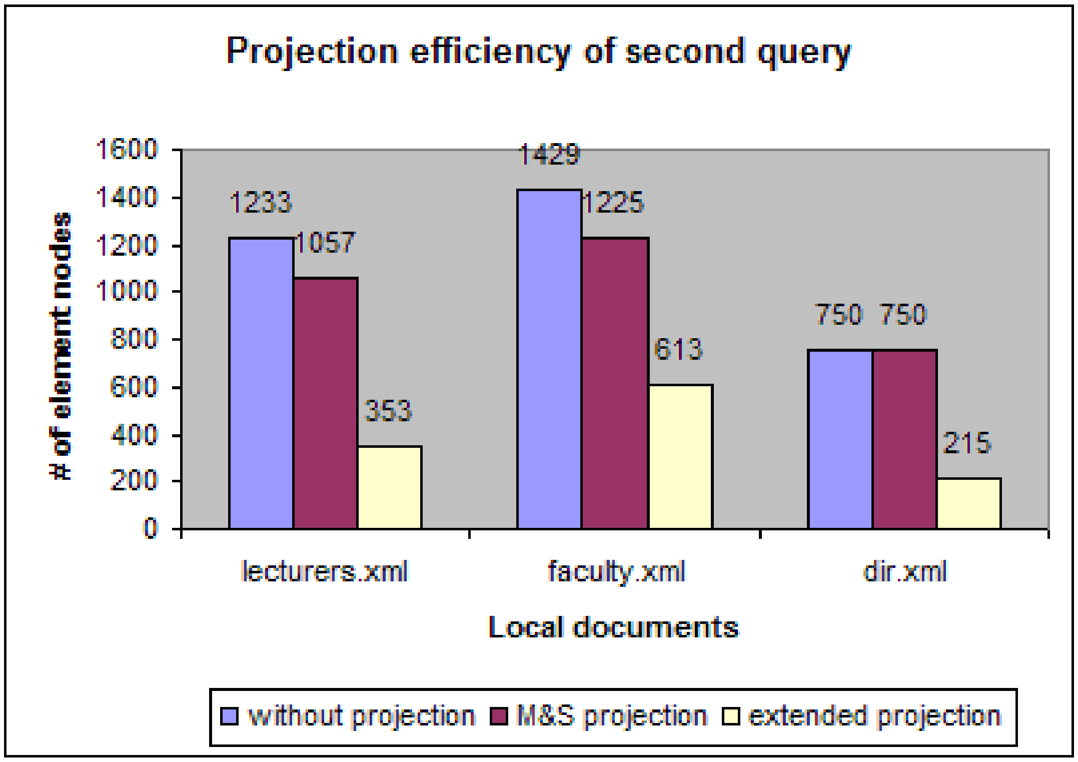


Figure 4.4. Comparing the Numbers of Element Nodes for the Second Query

the extended algorithm is 1737, which is a savings of 49 percent. Similar savings occur for the number of text nodes (12 percent versus 55 percent). For the figures showing the sizes of the projection documents for the M&S and the extended algorithm, the savings are 13 percent and 53 percent respectively.

4.2.2. Evaluation on the Second Outsourced Query

Consider the outsourced query in Listing 4.1 that show the names of all people living in Denton. This is a much simpler query and requires information from fewer documents. The following charts compare the projection paths returned by Marian and Siméon's algorithm and by the extended algorithm. The Figure 4.4 on page 101 compares the numbers of element nodes retrieved. The Figure 4.5 on page 102 compares the numbers of text nodes retrieved. The Figure 4.6 on page 103 compares the sizes of the projection documents retrieved.

(1) The projection paths for lecturers.xml at chemistry department wrapper

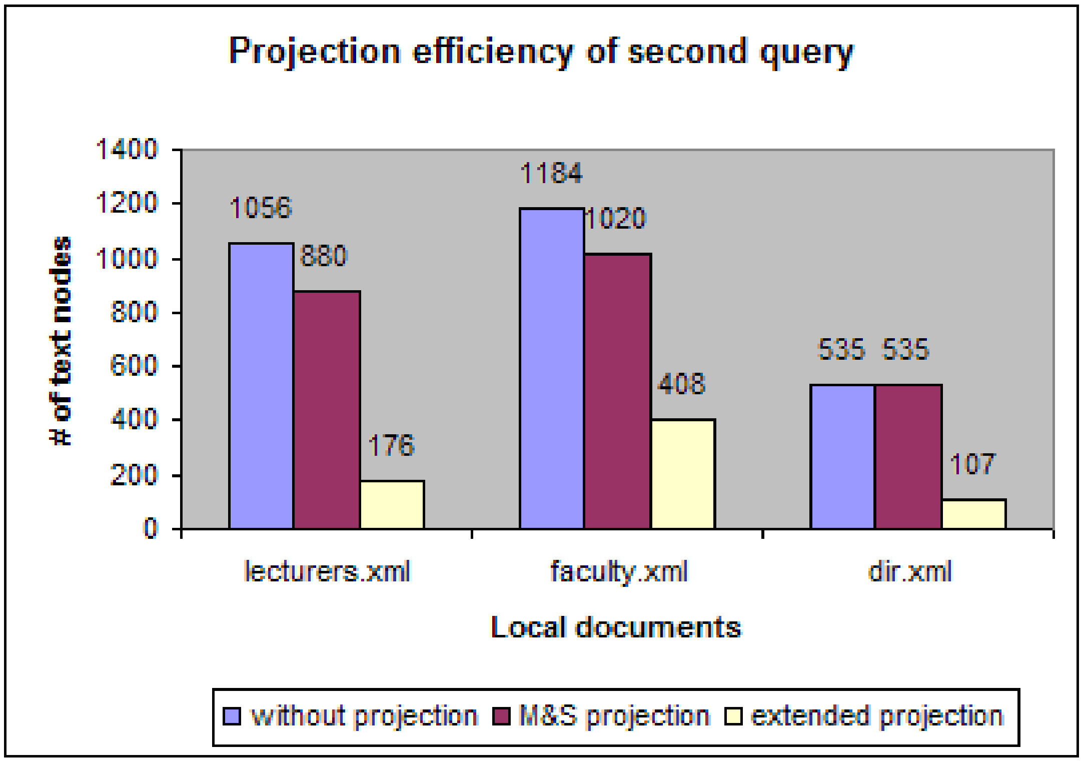


Figure 4.5. Comparing the Numbers of Text Nodes for the Second Query

(a) Marian and Siméon's algorithm returns:

(i) /Lecturers/Row

(ii) /Lecturers/Row/Name#

(iii) /Lecturers/Row/Pay_rate#

(iv) /Lecturers/Row/Monthly_hours#

(v) /Lecturers/Row/Rank#

(vi) /Lecturers/Row/Room/text()#

(b) The extended algorithm returns:

(i) /Lecturers/Row

(ii) /Lecturers/Row/Name#

(2) The projection paths for faculty.xml at the computer science department wrapper

(a) Marian and Siméon's algorithm returns:

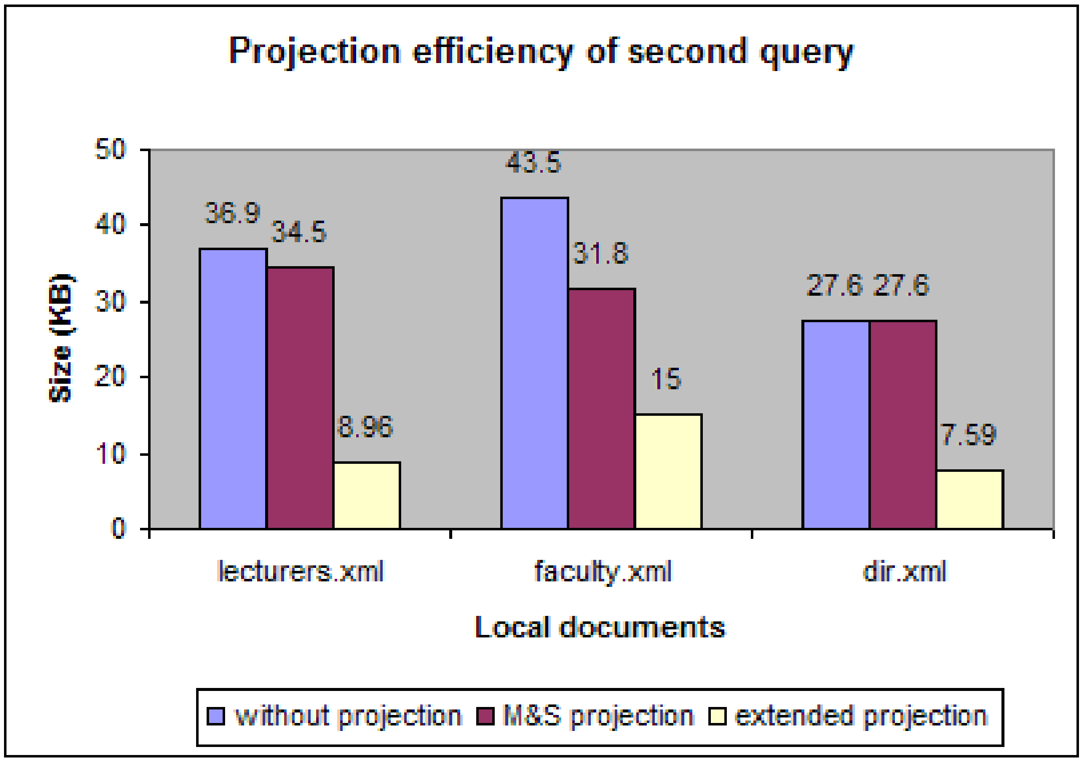


Figure 4.6. Comparing the Projection Sizes for the Second Query

(i) /Faculty/Row

(ii) /Faculty/Row/Fname#

(iii) /Faculty/Row/Lname#

(iv) /Faculty/Row/Salary#

(v) /Faculty/Row/Title/text()#

(vi) /Faculty/Row/Office#

(b) The extended algorithm returns:

(i) /Faculty/Row

(ii) /Faculty/Row/Fname#

(iii) /Faculty/Row/Lname#

(3) The projection paths for dir.xml at the college wrapper

(a) Marian and Siméon's algorithm returns:

(i) /Directory/Person

(ii) /Directory/Person/@Name

(iii) /Directory/Person/Phone#

(iv) /Directory/Person/Email#

(v) /Directory/Person/Address#

(b) The extended algorithm returns:

(i) /Directory/Person

(ii) /Directory/Person/@Name

(iii) /Directory/Person/Address/City#

The extended algorithm excludes many projection paths that are included by Marian and Siméon's algorithm, because there are not of interest to the user. Again, there are considerable differences in the number of documents retrieved by the two algorithms.

Listing 4.1. The Outsourced Query of Listing All People Living in Denton

```
<DentonPeople>
{
let $z := document {
<Personnel>
{ (
for $x in doc("lecturers.xml")/Lecturers/Row,
    $y in doc("dir.xml")/Directory/Person
where $x/Name = $y/@Name
return
  <Person>
    {$x/Name}
    <Department>CHEMISTRY</Department>
    <Salary> {($x/Pay_rate) * ($x/Monthly_hours) * 12}
    </Salary>
    {$x/Rank}
    <Office>{$x/Room/text()}</Office>
    <Contacts>
      {$y/Phone}
      {$y/Email}
    </Contacts>
    {$y/Address}
  </Person> )
union (
for $x in doc("faculty.xml")/Faculty/Row,
    $y in doc("dir.xml")/Directory/Person
let $xname := string-join(($x/FName, $x/LName), "␣")
where $xname = $y/@Name
return
  <Person>
    <Name>{$xname}</Name>
    <Department>COMPUTER SCIENCE</Department>
    {$x/Salary}
    <Rank>{$x/Title/text()}</Rank>
    {$x/Office}
    <Contacts>
      {$y/Phone}
      {$y/Email}
    </Contacts>
    {$y/Address}
  </Person> )
}
</Personnel>
}
for $x in $z/Personnel/Person
where $x/Adddress/City = "Denton"
return
    $x/Name
}
</DentonPeople>
```

4.2.3. Evaluation on the Third Outsourced Query

Consider the outsourced query in Listing 4.2 that list the names and email addresses of all full professors. This is a much more complex query and requires the examination of more documents than the previous query. The following figures compare the projection paths returned by Marian and Siméon's algorithm and by the extended algorithm. The Figure 4.7 on page 106 compares the numbers of element nodes retrieved. The Figure 4.8 on page 107 compares the numbers of text nodes retrieved. The Figure 4.9 on page 108 compares the sizes of the projection documents retrieved.

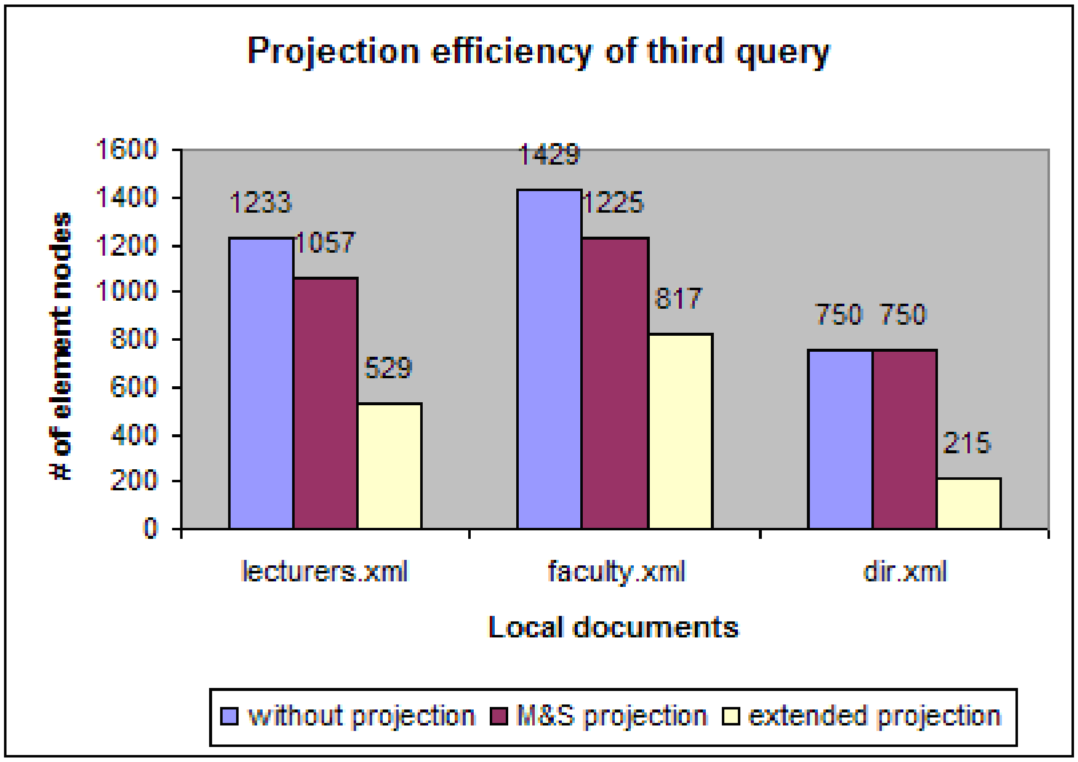


Figure 4.7. Comparing the Numbers of Element Nodes for the Third Query

(1) The projection paths for lecturers.xml at the chemistry department wrapper

(a) Marian and Siméon's algorithm returns:

(i) /Lecturers/Row

(ii) /Lecturers/Row/Name#

(iii) /Lecturers/Row/Pay_rate#

(iv) /Lecturers/Row/Monthly_hours#

(v) /Lecturers/Row/Rank#

(vi) /Lecturers/Row/Room/text()#

(b) My extended algorithm returns:

(i) /Lecturers/Row

(ii) /Lecturers/Row/Name#

(iii) /Lecturers/Row/Rank#

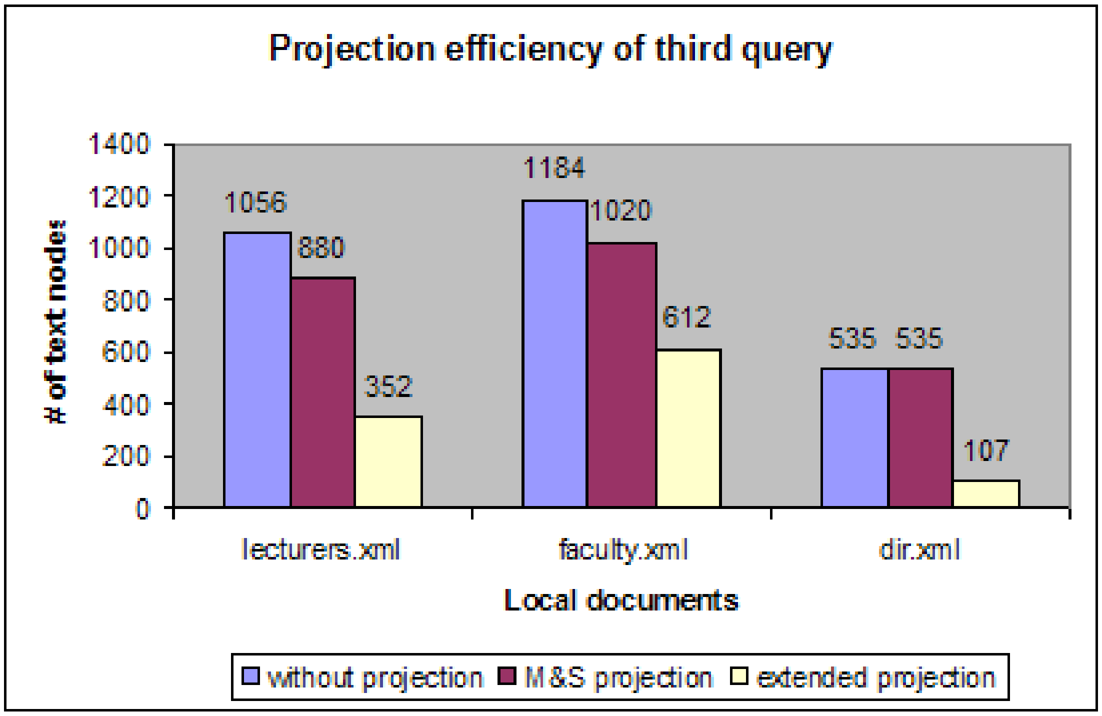


Figure 4.8. Comparing the Numbers of Text Nodes for the Third Query

(2) The projection paths for faculty.xml at the computer science department wrapper

(a) Marian and Siméon's algorithm returns:

(i) /Faculty/Row

(ii) /Faculty/Row/Fname#

(iii) /Faculty/Row/Lname#

(iv) /Faculty/Row/Salary#

(v) /Faculty/Row/Title/text()#

(vi) /Faculty/Row/Office#

(b) My extended algorithm returns:

(i) /Faculty/Row

(ii) /Faculty/Row/Fname#

(iii) /Faculty/Row/Lname#

(iv) /Faculty/Row/Title/text()#

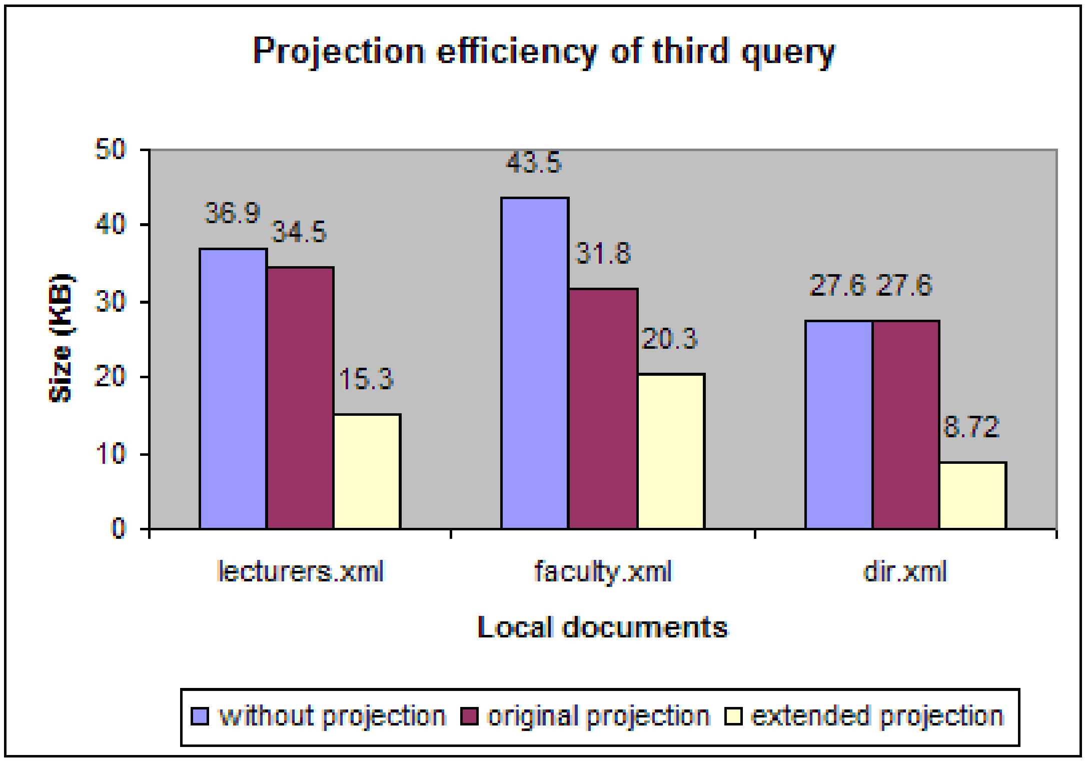


Figure 4.9. Comparing the Projection Sizes for the Third Query

(3) The projection paths for dir.xml at the college wrapper

(a) Marian and Siméon's algorithm returns:

(i) /Directory/Person

(ii) /Directory/Person/@Name

(iii) /Directory/Person/Phone#

(iv) /Directory/Person/Email#

(v) /Directory/Person/Address#

(b) My extended algorithm returns:

(i) /Directory/Person

(ii) /Directory/Person/@Name

(iii) /Directory/Person/Email#

The extended algorithm excludes many projection paths that are included by Marian and Siméon's algorithm, because they are not of interest to the user.

Listing 4.2. The Outsourced Query of Listing the Emails of All Full Professors

```
<ProfessorEmails>
{
let $z := document {
<Personnel>
{ (
for $x in doc("lecturers.xml")/Lecturers/Row,
    $y in doc("dir.xml")/Directory/Person
where $x/Name = $y/@Name
return
  <Person>
    {$x/Name}
    <Department>CHEMISTRY</Department>
    <Salary> {($x/Pay_rate) * ($x/Monthly_hours) * 12}
    </Salary>
    {$x/Rank}
    <Office>{$x/Room/text()}</Office>
    <Contacts>
      {$y/Phone}
      {$y/Email}
    </Contacts>
    {$y/Address}
  </Person> )
union (
for $x in doc("faculty.xml")/Faculty/Row,
    $y in doc("dir.xml")/Directory/Person
let $xname := string-join(($x/FName, $x/LName), "␣")
where $xname = $y/@Name
return
  <Person>
    <Name>{$xname}</Name>
    <Department>COMPUTER SCIENCE</Department>
    {$x/Salary}
    <Rank>{$x/Title/text()}</Rank>
    {$x/Office}
    <Contacts>
      {$y/Phone}
      {$y/Email}
    </Contacts>
    {$y/Address}
  </Person> )
}
</Personnel>
}
for $x in $z/Personnel/Person
where $x/Rank = "Full␣Professor"
return
   <Professor>
       {$x/Name}
       {$x/Contacts/Email}
   </Professor>
}
</ProfessorEmails>
```

4.2.4. Summary

In summary, the extended projection algorithm created for this dissertation performed consistently better on the three queries than either no projection or the Marian and Siméon algorithm was applied. The table 4.1 shows the performance and savings for each query for each type of algorithm on all three data sources. Because both the original and the Marian and Siméon algorithm retrieve the same numbers of nodes (or projection document sizes) for all three queries, these numbers are not repeated. The average savings on the element nodes is 11 percent for the M&S algorithm and 56 percent for the extended algorithm on the average for all three queries. These same savings can be seen in the case of text nodes, 12 percent versus a savings of 65 percent. Note that the Address element is a complex element, because it has three nested elements: Street, City and State. Therefore, the extended project algorithm is able to exclude more than two thirds of the document in the dir.xml data source in the first evaluation query, which results in savings of 53 percent. One can easily imagine other examples that would incur even larger savings. For simplicity and clarity, I selected only a few columns for the lecturer and faculty tables. If these data sources contained many more columns, the extended algorithm would exclude those extra columns while the Marian and

Element Nodes Returned	Lecturer.xml	Faculty.xml	Dir.xml	Total Nodes	Percent Savings
Original	1233	1429	750	3412	
M&S	1057	1225	750	3032	11
Extended Query 1	705	817	215	1735	49
Extended Query 2	353	613	215	1181	65
Extended Query 3	529	817	215	1561	54

Table 4.1. Savings Summary of the Number of Nodes Returned

Siméon algorithm would retain them. The case for dir.xml is similar. The conclusion is that the more projective a user query is on the mediated data the more the savings. In conclusion, the extended projection generates smaller projection documents and hence fetches less data from the original documents. The reason that this occurs is that my algorithm can remember and recall constructed elements or documents and apply projection analysis on them.

CHAPTER 5

CONCLUSION

The major goal of this research was to develop a more efficient method for translating user queries into atomic queries. The motivation for such a goal was to provide more efficient and easier access to multiple heterogeneous data sources through a single query. In this type of scenario, information is requested with a single query that may require data from many different data sources. The query must be decomposed into many so-called atomic queries; one for each data source. The emergence of XML standards and the development of a projection algorithm for XML documents helped solve some of the problems related to this research. This previous work, therefore, formed the basis of this dissertation by revealing a technique for processing the XQuery language. As a result, this research built a database system that uses the mediator approach for accessing distributed heterogeneous data sources. This means that the atomic queries are distributed by a mediator to different wrappers, each of which provides the query interface to a specific data source. The wrappers process the atomic query and return their results to the mediator, which integrates the results into a single answer to the user's original query. For efficiency's sake, it is important that only the data necessary to answer the user query be returned to the mediator. Therefore, it was necessary to develop an algorithm and representation language for the metadata that would allow the database system to decompose the original query into atomic queries so that only a subset of the data would be returned to the mediator. The Amélie Marian and Jérôme Siméon XML document projection algorithm is an example that allows a subset of data from an XML document to be used to answer a query, so that larger XML documents can be processed in memory. For the system used in this research, the Amélie Marian and Jérôme Siméon algorithm was extended so that it would work with an information integration application; It was determined that XQuery expressions could be used for the metadata and that an improved notation could be used to describe the resulting algorithm.

Following the completion of this task, a study was performed to test whether the extended algorithm was an improvement over the Marian and Siméon algorithm. Three different evaluation queries were run against the system. These queries were analyzed by determining the projection efficiency for each algorithm. This was done by counting the numbers of element and text nodes retrieved as well as measuring the sizes of the projection documents for each algorithm. The extended XML document projection algorithm created for this dissertation performed consistently better.

5.1. Contributions

The contributions of this dissertation are:

(1) The development of a query decomposition algorithm based on extended XML document projection.

    This algorithm generates atomic queries that fetch only relevant data from data sources. Thus network traffic is reduced and system performance is improved.

    Amélie Marian and Jérôme Siméon presented a static analysis on how to project XML documents [52]. However, their projection does not analyze navigations in constructed elements because they concluded that these navigations did not apply to the input documents. This research demonstrates that those navigations apply to the input documents and extend the projection analysis to constructors. First, this extension enables memory-based XQuery engines to process larger documents whenever there are projections on constructed elements or documents. Second, user queries in the mediation system developed for this research are based on virtual mediated documents which are all constructed by metaqueries. The extension enables the system to pinpoint the relevant data from the data sources.

This research also Improved the notation for presenting projection inference rules. Amélie Marian and Jérôme Siméon use the notation: "Paths_1 using Paths_2". While this notation appears to be appropriate for the algorithm presented in their research, it made the extended algorithm developed for this research more difficult to understand. The static analysis required in the extended XML document projection research is more complicated than theirs and requires a better notation. This new notation is based on ordered pairs, and it is better for describing more complicated XML document projection using inference rules.

(2) The use of XQuery expressions to represent mappings between global schema and local schemas.

This research shows that data integrators do not have to use proprietary or special languages to represent the mappings between data sources. The standard and powerful XQuery is sufficient to describe complicated mappings.

(3) The implementation of a prototype mediation system called Omphalos to demonstrate the extended algorithms.

The research system is based on XML related standards. It uses XML and W3C XML Schema as the data model and data exchange format, and XQuery to describe mapping expressions in the metadata. Developers need not learn proprietary technologies that are not widely used. Also XQuery is strongly typed, making it more user friendly and easier to debug.

The XML content model allows the creation of very flexible structures, and the W3C XML Schema supports very rich types. Because XML is rapidly expanding, there is a trend to take the data that is in other formats and wrap or transform them to XML. Therefore many data sources are already in XML format, which will allow information integrators to build mediation systems faster. And if a data source is wrapped, it is more likely to be reused by other applications.

A hierarchy of mediators can be easily built, since one can use XML and XQuery as the unified interface between mediators and wrappers.

(4) The implementation of a collaborative schema-matching tool.

The schema-matching tool running in a collaborative environment can improve the effectiveness and efficiency of the cooperation among data integration and domain experts when they are separated by distance. It also makes it easier to create the XML metaqueries. This tool is able to plug-in schema-matching algorithms to semi-automatically generate mappings.

5.2. Future Work

While this dissertation contributes to the problem of representing and processing metadata in mediator database systems, it does not provide a complete solution. The complexity of this problem implies that there is a vast amount of work that must be done before a final solution will be available. Some future areas of work are now outlined below.

(1) Automatic Metaquery Generation

The XQuery views used in the metadata for this research are called XQuery metaqueries. They are XQuery expressions that capture mappings/correspondences between mediated schema and data source schemas. Currently these XQuery metaqueries are created manually, which tends to be error prone and time consuming. Future work will include incorporating a back end to the collaborative schema matching tool in order to generate these XQuery views automatically.

The IBM®[1] Clio project has developed a tool to generate XQuery queries that can transform the source data in nested relational data model to the target data in a nested relational data model [68]. The Microsoft®[2] BizTalk™ Mapper can generate XSLT code to translate one XML document to another XML document

[1]IBM, www.ibm.com

[2]Microsoft Corporation, www.microsoft.com

according to their W3C XML Schema instances [54, 81]. Similar techniques can be applied and implemented in the collaborative tool.

The generation of metaqueries automatically will make the research system more flexible and it should scale well. It will be easier to build and maintain data integration systems. It will be easier to accommodate changes in data source schemas or mediated schemas. Moreover, it will be easier to add or remove data sources. Productivity will increase since there will be fewer human errors.

(2) A graphical query language.

A graphical query language will broaden the user base. It will not only enable non-professionals to use the system, but it will also lead to more efficient development.

(3) A benchmark specially designed for testing mediation systems.

There are a number of XML and data integration benchmarks or testbeds including The Amalgam Schema and Data Integration Test Suite [55], Clio Sample Schemas [20], The Michigan Benchmark [73], NIST XQuery Test Suite [57], THALIA [35], The UIUC web integration repository [12], XBench [89], XMach-1 [5], Xmark [75], XMLBench [77], and XOO7 [7]. These benchmarks, however, were not found to be suitable for evaluating mediation systems. A benchmark specially designed for testing mediation systems would be a real contribution to this field.

5.3. Summary

This research offers insight into representing a mapping between a global schema and local data source schemas and translating user queries into atomic queries in mediator database systems. It shows that it is possible to use XQuery to describe mappings between a global schema and local data sources. Moreover, it shows that it is possible to extend an XML document projection algorithm to be used in information integration applications and increase the accuracy and efficiency of the original algorithm. This research extended the projection analysis onto constructed elements and showed how it could operate on multiple data sources. A prototype mediation system called Omphalos was then developed to demonstrate the techniques and algorithms discussed in this thesis. Combining the capability of a metaquery generation tool, this system provides users with a powerful information integration system that allows flexible partial integration, scales well, and is easy to administer and maintain.

APPENDIX A

COLLEGE DIRECTORY DIR.XML

The college directory dir.xml.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Directory>
<Person Name="Frank Kerry">
<Phone>456-677-8007</Phone>
<Email>frank_kerry@unt.edu</Email>
<Address>
<Street>560 Bryan St</Street>
<City>Denton</City>
<State>Texas</State>
</Address>
</Person>
<Person Name="Mark Russell">
<Phone>456-677-8006</Phone>
<Email>mark_russell@unt.edu</Email>
<Address>
<Street>903 Ponder St</Street>
<City>Denton</City>
<State>Texas</State>
</Address>
</Person>
<Person Name="Logan Nixon">
<Phone>456-666-6595</Phone>
<Email>logan_nixon@unt.edu</Email>
<Address>
<Street>123 Scripture St</Street>
```

<City>Denton</City>

<State>Texas</State>

</Address>

</Person>

<Person Name="Darren Einstein">

<Phone>456-688-6596</Phone>

<Email>darren_einstein@unt.edu</Email>

<Address>

<Street>567 Oram St</Street>

<City>Dallas</City>

<State>Texas</State>

</Address>

</Person>

<Person Name="Samuel Justin">

<Phone>456-666-6592</Phone>

<Email>samuel_justin@unt.edu</Email>

<Address>

<Street>360 La vista St</Street>

<City>Dallas</City>

<State>Texas</State>

</Address>

</Person>

<Person Name="Daisy Henry">

<Phone>456-666-6591</Phone>

<Email>daisy_henry@unt.edu</Email>

<Address>

```
<Street>1400 Caroll Blvd</Street>
<City>Denton</City>
<State>Texas</State>
</Address>
</Person>
</Directory>
```

APPENDIX B

GLOBAL SCHEMA INSTANCE PERSONNEL.XSD

The global schema instance personnel.xsd.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema
targetNamespace="http://mars.csci.unt.edu/dbgroup/ditest/personnel"
xmlns:psn="http://mars.csci.unt.edu/dbgroup/ditest/personnel"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified"
attributeFormDefault="unqualified">
<xs:element name="Personnel">
<xs:annotation>
<xs:documentation xml:lang="en">
personnel.xml top level element.
</xs:documentation>
</xs:annotation>
<xs:complexType>
<xs:sequence>
<xs:element ref="psn:Person"
minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
<xs:key name="PersonName">
<xs:selector xpath="psn:Person"/>
<xs:field xpath="psn:Name"/>
</xs:key>
</xs:element>
<xs:element name="Person">
```

```
<xs:complexType>
<xs:annotation>
<xs:documentation xml:lang="en">
element Name represents a full name.
</xs:documentation>
</xs:annotation>
<xs:sequence>
<xs:element name="Name" type="xs:token"/>
<xs:element name="Department" type="psn:DepartmentType" />
<xs:element name="Salary" type="psn:NonNegativeDecimal" />
<xs:element name="Rank" type="psn:RankType" />
<xs:element name="Office" type="xs:token" />
<xs:element name="Contacts" type="psn:ContactsType" />
<xs:element name="Address" type="psn:AddressType" />
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:simpleType name="DepartmentType">
<!– 'COMPUTER SCIENCE' has a space and is not xs:NMTOKEN –>
<xs:restriction base="xs:token">
<xs:enumeration value="COMPUTER SCIENCE"/>
<xs:enumeration value="CHEMISTRY"/>
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="NonNegativeDecimal">
<xs:restriction base="xs:decimal">
```

```
<xs:minInclusive value="0"/>
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="RankType">
<xs:restriction base="xs:token">
<xs:enumeration value="Assistant Professor"/>
<xs:enumeration value="Assocate Professor"/>
<xs:enumeration value="Full Professor"/>
</xs:restriction>
</xs:simpleType>
<xs:complexType name="ContactsType">
<xs:sequence>
<xs:element name="Phone" type="psn:PhoneNumberType" />
<xs:element name="Email" type="psn:EmailType" />
</xs:sequence>
</xs:complexType>
<xs:simpleType name="PhoneNumberType">
<xs:annotation>
<xs:documentation xml:lang="en">
I don't care 2 hyphens or one hyphen or no hyphen.
</xs:documentation>
</xs:annotation>
<xs:restriction base="xs:string">
<xs:pattern value="[1-9][0-9]{2}(-| )?[0-9]{3}(-| )?[0-9]{4}"/>
</xs:restriction>
</xs:simpleType>
```

```
<xs:simpleType name="EmailType" >
<xs:restriction base="xs:token">
<xs:pattern value=
"([\.a-zA-Z0-9_-])+@([a-zA-Z0-9_-])+(([a-zA-Z0-9_-])*\.([a-zA-Z0-9_-])+)+"/>
</xs:restriction>
</xs:simpleType>
<xs:complexType name="AddressType">
<xs:sequence>
<xs:element name="Street" type="xs:token"/>
<!– 'El Paso' has a space. So City can not be xs:NMTOKEN –>
<xs:element name="City" type="xs:token"/>
<xs:element name="State" type="xs:NMTOKEN"/>
</xs:sequence>
</xs:complexType>
</xs:schema>
```

APPENDIX C

THE XML PARSE TREE FOR PERSONNEL.XSD

The XML parse tree for personnel.xsd.

An XML parser reads in an XML document and produces a parse tree explicitly (DOM parser) or implicitly (SAX parser). The following is the XML parse tree for personnel.xsd. The empty text nodes between elements have been omitted for clarity.

DOCUMENT_NODE: #document

ELEMENT_NODE: xs:schema

ATTRIBUTE_NODE: targetNamespace="http://mars.csci.unt.edu/ ..."

ATTRIBUTE_NODE: xmlns:psn="http://mars.csci.unt.edu/ ..."

ATTRIBUTE_NODE: xmlns:xs="http://www.w3.org/2001/XMLSchema"

ATTRIBUTE_NODE: elementFormDefault="qualified"

ATTRIBUTE_NODE: attributeFormDefault="unqualified"

ELEMENT_NODE: xs:element

ATTRIBUTE_NODE: name="Personnel"

ELEMENT_NODE: xs:annotation

ELEMENT_NODE: xs:documentation

ATTRIBUTE_NODE: xml:lang="en"

TEXT_NODE: #text="personnel.xml top level element."

ELEMENT_NODE: xs:complexType

ELEMENT_NODE: xs:sequence

ELEMENT_NODE: xs:element

ATTRIBUTE_NODE: ref="psn:Person"

ATTRIBUTE_NODE: minOccurs="0"

ATTRIBUTE_NODE: maxOccurs="unbounded"

ELEMENT_NODE: xs:key

ATTRIBUTE_NODE: name="PersonName"

ELEMENT_NODE: xs:selector

```
ATTRIBUTE_NODE: xpath="psn:Person"

ELEMENT_NODE: xs:field

ATTRIBUTE_NODE: xpath="psn:Name"

ELEMENT_NODE: xs:element

ATTRIBUTE_NODE: name="Person"

ELEMENT_NODE: xs:complexType

ELEMENT_NODE: xs:annotation

ELEMENT_NODE: xs:documentation

ATTRIBUTE_NODE: xml:lang="en"

TEXT_NODE: #text="element Name represents a full name."

ELEMENT_NODE: xs:sequence

ELEMENT_NODE: xs:element

ATTRIBUTE_NODE: name="Name"

ATTRIBUTE_NODE: type="xs:token"

ELEMENT_NODE: xs:element

ATTRIBUTE_NODE: name="Department"

ATTRIBUTE_NODE: type="psn:DepartmentType"

ELEMENT_NODE: xs:element

ATTRIBUTE_NODE: name"Salary"

ATTRIBUTE_NODE: type="psn:NonNegativeDecimal"

ELEMENT_NODE: xs:element

ATTRIBUTE_NODE: name="Rank"

ATTRIBUTE_NODE: type="psn:RankType"

ELEMENT_NODE: xs:element

ATTRIBUTE_NODE: name="Office"

ATTRIBUTE_NODE: type="xs:token"
```

```
ELEMENT_NODE: xs:element
ATTRIBUTE_NODE: name="Contacts"
ATTRIBUTE_NODE: type="psn:ContactsType"
ELEMENT_NODE: xs:element
ATTRIBUTE_NODE: name="Address"
ATTRIBUTE_NODE: type="psn:AddressType"
ELEMENT_NODE: xs:simpleType
ATTRIBUTE_NODE: name="DepartmentType"
COMMENT_NODE: #comment
ELEMENT_NODE: xs:restriction
ATTRIBUTE_NODE: base="xs:token"
ELEMENT_NODE: xs:enumeration
ATTRIBUTE_NODE: value="COMPUTER SCIENCE"
ELEMENT_NODE: xs:enumeration
ATTRIBUTE_NODE: value="CHEMISTRY"
ELEMENT_NODE: xs:simpleType
ATTRIBUTE_NODE: name="NonNegativeDecimal"
ELEMENT_NODE: xs:restriction
ATTRIBUTE_NODE: base="xs:decimal"
ELEMENT_NODE: xs:minInclusive
ATTRIBUTE_NODE: value="0"
ELEMENT_NODE: xs:simpleType
ATTRIBUTE_NODE: name="RankType"
ELEMENT_NODE: xs:restriction
ATTRIBUTE_NODE: base="xs:token"
ELEMENT_NODE: xs:enumeration
```

```
ATTRIBUTE_NODE: value="Assistant Professor"

ELEMENT_NODE: xs:enumeration

ATTRIBUTE_NODE: value="Assocate Professor"

ELEMENT_NODE: xs:enumeration

ATTRIBUTE_NODE: value="Full Professor"

ELEMENT_NODE: xs:complexType

ATTRIBUTE_NODE: name="ContactsType"

ELEMENT_NODE: xs:sequence

ELEMENT_NODE: xs:element

ATTRIBUTE_NODE: name="Phone"

ATTRIBUTE_NODE: type="psn:PhoneNumberType"

ELEMENT_NODE: xs:element

ATTRIBUTE_NODE: name="Email"

ATTRIBUTE_NODE: type="psn:EmailType"

ELEMENT_NODE: xs:simpleType

ATTRIBUTE_NODE: name="PhoneNumberType"

ELEMENT_NODE: xs:annotation

ELEMENT_NODE: xs:documentation

ATTRIBUTE_NODE: xml:lang="en"

TEXT_NODE: #text="I don't care 2 hyphens or one hyphen ..."

ELEMENT_NODE: xs:restriction

ATTRIBUTE_NODE: base="xs:string"

ELEMENT_NODE: xs:pattern

ATTRIBUTE_NODE: value="[1-9][0-9]{2}(-| )?[0-9]{3}(-| )? ..."

ELEMENT_NODE: xs:simpleType

ATTRIBUTE_NODE: name="EmailType"
```

```
ELEMENT_NODE: xs:restriction
ATTRIBUTE_NODE: base="xs:token"
ELEMENT_NODE: xs:pattern
ATTRIBUTE_NODE: value=
"([\.a-zA-Z0-9_-])+@([a-zA-Z0-9_-])+(([a-zA-Z0-9_-])*\.([a-zA-Z0-9_-])+)+"
ELEMENT_NODE: xs:complexType
ATTRIBUTE_NODE: name="AddressType"
ELEMENT_NODE: xs:sequence
ELEMENT_NODE: xs:element
ATTRIBUTE_NODE: name="Street"
ATTRIBUTE_NODE: type="xs:token"
COMMENT_NODE: #comment
ELEMENT_NODE: xs:element
ATTRIBUTE_NODE: name="City"
ATTRIBUTE_NODE: type="xs:token"
ELEMENT_NODE: xs:element
ATTRIBUTE_NODE: name="State"
ATTRIBUTE_NODE: type="xs:NMTOKEN"
```

BIBLIOGRAPHY

[1] L. Afanasiev, M. Franceschet, M. Marx, and M. de Rijke, “Ctl model checking for processing simple xpath queries,” in *Proceedings of the 11th TIME*, 2004. [Online]. Available: citeseer.ist.psu.edu/678107.html

[2] C. Baru, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, P. Velikhov, and V. Chu, “XML-based information mediation with MIX,” in *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999*, 1999, pp. 597–599. [Online]. Available: citeseer.ist.psu.edu/baru99xmlbased.html

[3] C. K. Baru, B. Ludäscher, Y. Papakonstantinou, P. Velikhov, and V. Vianu, “Features and requirements for an XML view definition language: Lessons from XML information mediation,” in *Proceedings of QL’98 The Query Languages Workshop*, 1998. [Online]. Available: http://www.db.ucsd.edu/publications/xmas.html

[4] D. Beneventano, S. Bergamaschi, C. Castano, A. Corni, R. Guidetti, M. Malvezzi, M. Melchiori, and M. Vincini, “Information integration: The MOMIS project demonstration,” in *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10–14, 2000, Cairo, Egypt*, A. El Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K.-Y. Whang, Eds. Los Altos, CA 94022, USA: Morgan Kaufmann Publishers, 2000, pp. 611–614. [Online]. Available: http://www.vldb.org/dblp/db/conf/vldb/BeneventanoBCCGMMV00.html

[5] T. Böhme and E. Rahm, “XMach-1: A benchmark for XML data management,” in *In Proceedings of German database conference BTW2001, Oldenburg, 7.-9. March, 2001, Springer, Berlin.*, 2001.

[6] R. Brazile, K. Swigger, B. Harrington, B. Harrington, and X. Peng, “The international collaborative environment,” in *In Proceedings of the Computer Applications in Industry and Engineering (CAINE) 2002 Conference*, San Diego, California, November 2002.

[7] S. Bressan, M. L. Lee, Y. G. Li, Z. Lacroix, and U. Nambiar, “The xoo7 benchmark,” in *Proceedings of the first VLDB Workshop on Efficiency and Effectiveness of XML Tools, and Techniques (EEXTT)*, Hong Kong, China, August 2002.

[8] A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini, “Data integration under integrity constraints,” in *Proc. of the 14th Int. Conf. on Advanced Information Systems Engineering (CAiSE 2002)*, ser. Lecture Notes in Computer Science, vol. 2348. Springer, 2002, pp. 262–279.

[9] J. P. Callan and M. E. Connell, "Query-based sampling of text databases," *Information Systems*, vol. 19, no. 2, pp. 97–130, 2001. [Online]. Available: citeseer.ist.psu.edu/callan99querybased.html

[10] M. J. Carey, J. Kiernan, J. Shanmugasundaram, E. J. Shekita, and S. N. Subramanian, "XPERANTO: Middleware for publishing object-relational data as XML documents," in *The VLDB Journal*, 2000, pp. 646–648. [Online]. Available: citeseer.ist.psu.edu/article/carey00xperanto.html

[11] D. Chamberlin, D. Draper, M. Fernandez, M. Kay, J. Robie, M. Rys, J. Simeon, J. Tivy, and P. Wadler, *XQuery from the Experts: A Guide to the W3C XML Query Language*, 1st ed., H. Katz, Ed. Addison-Wesley, September 2003.

[12] K. C.-C. Chang, B. He, C. Li, and Z. Zhang, "The UIUC web integration repository," Computer Science Department, University of Illinois at Urbana-Champaign. http://metaquerier.cs.uiuc.edu/repository, 2003.

[13] A. B. Chaudhri, A. Rashid, and R. Zicari, Eds., *XML data management : native XML and XML-enabled database systems*, 1st ed. Addison Wesley Professional, 2003.

[14] A. B. Chaudhri, R. Unland, C. Djeraba, and W. Lindner, Eds., *XML-Based Data Management and Multimedia Engineering - EDBT 2002 Workshops, EDBT 2002 Workshops XMLDM, MDDE, and YRWS, Prague, Czech Republic, March 24-28, 2002, Revised Papers*, ser. Lecture Notes in Computer Science, vol. 2490. Springer, 2002.

[15] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. D. Ullman, and J. Widom, "The TSIMMIS project: Integration of heterogeneous information sources," in *In Proceedings of the 16th Meeting of the Information Processing Society of Japan*, Tokyo, Japan, 1994, pp. 7–18. [Online]. Available: citeseer.ist.psu.edu/chawathe94tsimmis.html

[16] V. Christophides, S. Cluet, and J. Simèon, "On wrapping query languages and efficient XML integration," in *In Proceedings of ACM SIGMOD Conference on Management of Data*, Dallas, Texas, May 2000, pp. 141–152. [Online]. Available: citeseer.ist.psu.edu/christophides00wrapping.html

[17] J. Clark, "RELAX NG home page," http://www.relaxng.org/. [Online]. Available: http://www.relaxng.org/

[18] ——, "TREX - tree regular expressions for XML," http://www.thaiopensource.com/trex/. [Online]. Available: http://www.thaiopensource.com/trex/

[19] C. Clifton, E. Housman, and A. Rosenthal, "Experience with a combined approach to attribute-matching across heterogeneous databases," in *DS-7*, 1997, pp. 0–. [Online]. Available: citeseer.ist.psu.edu/clifton97experience.html

[20] "Clio sample schemas," http://www.cs.toronto.edu/db/clio/testSchemas.html, Clio Group at the University of Toronto and IBM's Almaden Research Center. [Online]. Available: http://www.cs.toronto.edu/db/clio/testSchemas.html

[21] I. F. Cruz and K. M. James, "A user interface for distributed multimedia database querying with mediator supported refinement," in *Proceedings of ACM SIGMOD*, Philadelphia, Apr. 16 1999, pp. 590–592. [Online]. Available: http://citeseer.ist.psu.edu/371510.html;http://www.cs.wpi.edu/~beez/Papers/IDEAS99.ps

[22] O. Duschka and M. Genesereth, "Infomaster - an information integration tool," in *In Proceedings of the International Workshop on Intelligent Information Integration*, Freiburg, Germany, September 1997. [Online]. Available: citeseer.ist.psu.edu/duschka97infomaster.html

[23] O. M. Duschka, M. R. Genesereth, and A. Y. Levy, "Recursive query plans for data integration," *Journal of Logic Programming*, vol. 43, no. 1, pp. 49–73, 2000. [Online]. Available: citeseer.ist.psu.edu/duschka99recursive.html

[24] L. Fegaras, "The joy of sax," in *Informal Proceedings of the 1st XIME-P*, 2004. [Online]. Available: citeseer.ist.psu.edu/648806.html

[25] M. Fernandez, W. Tan, and D. Suciu, "SilkRoute: Trading between relational and xml," in *In Proc. of the Int. WWW Conf.*, May 2000.

[26] M. Friedman, A. Y. Levy, and T. D. Millstein, "Navigational plans for data integration," in *AAAI/IAAI*, 1999, pp. 67–73. [Online]. Available: citeseer.ist.psu.edu/article/friedman99navigational.html

[27] H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom, "Integrating and accessing heterogeneous information sources in tsimmis," in *In Proceedings of the AAAI Symposium on Information Gathering*, Stanford, California, Mar. 1995, pp. 61–64. [Online]. Available: citeseer.ist.psu.edu/article/garcia-molina95integrating.html

[28] H. Garcia-Molina, J. Ulman, and J. Widom., *Database Implementation*. New Jersey: Prentice Hall, 2000.

[29] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom, "The TSIMMIS approach to mediation: Data models and languages," *Journal of Intelligent Information Systems*, vol. 8, no. 2, pp. 117–132, 1997. [Online]. Available: citeseer.ist.psu.edu/article/garcia-molina97tsimmis.html

[30] H. Garcia-Molina, J. D. Ullman, and J. Widom, *Database Systems: The Complete Book*. Prentice Hall, 2002, pp. 1047–1100.

[31] G. Gardarin, A. Mensch, T. Tuyet Dang-Ngoc, and L. Smit, “Integrating heterogeneous data sources with XML and XQuery,” in *Proceedings of 13th International Workshop on Database and Expert Systems Applications, September 02 - 06, 2002, Aix-en-Provence, France*, Sept. 2002, pp. 839– 844.

[32] G. Gardarin, A. Mensch, T. Tuyet Dang-Ngoc, L. Smit, and e XMLMedia, “Integrating heterogeneous data sources with XML and XQuery,” http://www.e-xmlmedia.fr/HDI-XML&XQuery.pdf, e-XMLMedia. [Online]. Available: http://www.e-xmlmedia.fr/HDI-XML&XQuery.pdf

[33] 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*, vol. 17, no. 3, pp. 270–270, 1999. [Online]. Available: citeseer.ist.psu.edu/3111.html

[34] L. M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang, “Optimizing queries across diverse data sources,” in *Proceedings of the Twenty-third International Conference on Very Large Databases*. Athens, Greece: VLDB Endowment, Saratoga, Calif., 1997, pp. 276–285. [Online]. Available: citeseer.ist.psu.edu/article/haas97optimizing.html

[35] J. Hammer, M. Stonebraker, and O. Topsakal, “THALIA: Test harness for the assessment of legacy information integration approaches,” in *The 21st International Conference on Data Engineering (ICDE 2005)*. Tokyo, Japan: National Center of Sciences, Tokyo, Japan, April 2005. [Online]. Available: http://icde2005.is.tsukuba.ac.jp/

[36] E. R. Harold and W. S. Means, *XML in a nutshell*, 2nd ed. O’Reilly, 2002.

[37] “ISO/IEC 19757 - DSDL document schema definition languages,” http://dsdl.org/, ISO/IEC JTC 1/SC 34 WG 1. [Online]. Available: http://dsdl.org/

[38] R. Jakobovits, “Integrating heterogeneous autonomous information sources,” University of Washington,, Tech. Rep. TR-97-12-05, 1997.

[39] R. Jelliffe, “The schematron assertion language 1.5,” http://xml.ascc.net/resource/schematron/Schematron2000.html, Academia Sinica Computing Centre. [Online]. Available: http://xml.ascc.net/resource/schematron/Schematron2000.html

[40] ——, “The schematron home page,” http://xml.ascc.net/resource/schematron/, Academia Sinica Computing Centre. [Online]. Available: http://xml.ascc.net/resource/schematron/

[41] W. Just and M. Weese, *Discovering modern set theory*, ser. Graduate studies in mathematics ; v. 8, 18. American Mathematical Society, 1996, vol. 1, no. SBN/ISSN 0821802666, includes bibliographical references and indexes. 1. The basics – 2. Set-theoretic tools for every mathematician.

[42] R. King, M. Novak, C. Och, and F. Velez, "Sybil: Supporting heterogeneous database interoperability with lightweight alliance," in *Next Generation Information Technologies and Systems*, 1997, pp. 0–. [Online]. Available: citeseer.ist.psu.edu/king97sybil.html

[43] T. Kirk, A. Y. Levy, Y. Sagiv, and D. Srivastava, "The information manifold," in *In Proc. of the AAAI Spring Symposium on Information Gathering in Distributed Heterogeneous Environments*, C. Knoblock and A. Levy, Eds., Stanford University, Stanford, California, 1995. [Online]. Available: citeseer.ist.psu.edu/16242.html

[44] C. Koch, S. Scherzinger, N. Schweikardt, and B. Stegmaier, "Schema-based scheduling of event processors and buffer minimization for queries on structured data streams," in *Proceedings of the 30th VLDB*, 2004. [Online]. Available: citeseer.ist.psu.edu/article/koch04schemabased.html

[45] ——, "Fluxquery: An optimizing xquery processor for streaming xml data," in *Proceedings of the 30th VLDB*, 2004. [Online]. Available: citeseer.ist.psu.edu/koch04fluxquery.html

[46] M. Lenzerini, "Data integration: a theoretical perspective," in *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS02).* New York, NY, USA: ACM Press, 2002, pp. 233–246.

[47] A. Y. Levy, "Logic-based techniques in data integration," in *Workshop on Logic-Based Artificial Intelligence, Washington, DC, June 14–16, 1999*, J. Minker, Ed. College Park, Maryland: Computer Science Department, University of Maryland, 1999. [Online]. Available: citeseer.ist.psu.edu/391746.html

[48] ——, "Logic-based techniques in data integration," in *Logic-Based Artificial Intelligence*, J. Minker, Ed. Dordrecht: Kluwer Academic Publishers, 2000, pp. 575–595.

[49] A. Y. Levy, A. Rajaraman, and J. J. Ordille, "Querying heterogeneous information sources using source descriptions," in *Proceedings of the Twenty-second International Conference on Very Large Databases*. Bombay, India: VLDB Endowment, Saratoga, Calif., 1996, pp. 251–262. [Online]. Available: citeseer.ist.psu.edu/levy96querying.html

[50] L. Liu and C. Pu, "The distributed interoperable object model and its application to large-scale interoperable database systems," in *CIKM*, 1995, pp. 105–112. [Online]. Available: citeseer.ist.psu.edu/liu95distributed.html

[51] I. Manolescu, D. Florescu, and D. K. Kossmann, "Answering XML queries over heterogeneous data sources," in *Proceedings of 27th International Conference on Very Large Databases*, Roma, Italy, Sept. 2001, pp. 241–250. [Online]. Available: citeseer.ist.psu.edu/manolescu01answering.html

[52] A. Marian and J. Siméon, "Projecting XML documents," in *Proceedings of VLDB 2003*, 2003. [Online]. Available: citeseer.ist.psu.edu/marian03projecting.html

[53] ——, "Projecting XML documents," Computer Science Department, Columbia University, Tech. Rep., February 2003. [Online]. Available: http://www.cs.columbia.edu/~library/

[54] J. Matranga, S. Tranchida, and B. Preecs, *Understanding BizTalk*, 1st ed. SAMS, 2000, indianapolis, Ind.

[55] R. J. Miller, D. Fisla, M. Huang, D. Kalmuk, F. Ku, and V. Lee, "The amalgam schema and data integration test suite," http://www.cs.toronto.edu/ miller/amalgam/.

[56] J. C. Mitchell, *Foundations for Programming Languages*, ser. Foundations of Computing. Cambridge, Massachusetts: The MIT Press, 1996.

[57] C. Montanez, "NIST XQuery test suite," http://xw2k.sdct.itl.nist.gov/brady/xmlQuery.zip, NIST, Information Technology Laboratory, Software Diagnostics and Conformance Testing Division. [Online]. Available: http://xw2k.sdct.itl.nist.gov/BRADY/xmlquery/testSuite/NIST/files/readme.html

[58] M. MURATA, "RELAX (regular language description for XML)," http://www.xml.gr.jp/relax/. [Online]. Available: http://www.xml.gr.jp/relax/

[59] "RELAX NG specification," http://www.oasis-open.org/committees/relax-ng/spec-20011203.html, OASIS (Organization for the Advancement of Structured Information Standards), 2001, OASIS Technical Committee Specification: 3 December 2001. [Online]. Available: http://www.oasis-open.org/committees/relax-ng/spec-20011203.html

[60] D. Olteanu, H. Meuss, *et al.*, "Xpath: Looking forward." in *EDBT Workshops*, ser. Lecture Notes in Computer Science, A. B. Chaudhri, R. Unland, C. Djeraba, and W. Lindner, Eds., vol. 2490. Springer, 2002, pp. 109–127.

[61] D. Olteanu, H. Meuss, T. Furche, and F. Bry, "Xpath: Looking forward." in *EDBT Workshops*, ser. Lecture Notes in Computer Science, A. B. Chaudhri, R. Unland, C. Djeraba, and W. Lindner, Eds., vol. 2490. Springer, 2002, pp. 109–127.

[62] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina, "Object fusion in mediator systems," in *Proceedings of the Twenty-second International Conference on Very Large Databases*, 1996, pp. 413–424. [Online]. Available: citeseer.ist.psu.edu/papakonstantinou96object.html

[63] Y. Papakonstantinou, H. García-Molina, and J. Ullman, "Medmaker: A mediation system based on declarative specifications," in *Proceedings of the 12th International Conference on Data Engineering*, New Orleans, La., 1996. [Online]. Available: citeseer.ist.psu.edu/papakonstantinou95medmaker.html

[64] X. Peng, R. Brazile, and K. M. Swigger, "Using XQuery to describe mappings from global schemas to local data sources," in *In Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration*, A. M. Memon, Ed. IEEE Systems, Man, and Cybernetics Society, Novemeber 2004, pp. 97–102.

[65] ——, "Extending XML document projection for data integration," in *In Proceedings of the 2005 IEEE International Conference on Information Reuse and Integration*, D. Zhang, Ed. Las Vegas, Nevada, USA: IEEE Systems, Man, and Cybernetics Society, August 2005, pp. 138–143.

[66] B. C. Pierce, *Types and Programming Languages*. Cambridge, Massachusetts: The MIT Press, 2002.

[67] G. D. Plotkin, "A Structural Approach to Operational Semantics," University of Aarhus, Tech. Rep. DAIMI FN-19, 1981. [Online]. Available: citeseer.ist.psu.edu/plotkin81structural.html

[68] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin, "Translating web data," in *Proceedings of VLDB 2002*, Hong Kong SAR, China, 2002, pp. 598–609. [Online]. Available: citeseer.ist.psu.edu/popa02translating.html

[69] R. Pottinger and A. Halevy, "MiniCon: A scalable algorithm for answering queries using views," *VLDB Journal: Very Large Data Bases*, vol. 10, no. 2–3, pp. 182–198, 2001. [Online]. Available: citeseer.ist.psu.edu/pottinger01minicon.html

[70] X. Qian, "Query folding," in *12th Int. Conference on Data Engineering*, S. Y. Su, Ed., New Orleans, Louisiana, 1996, pp. 48–55. [Online]. Available: citeseer.ist.psu.edu/qian96query.html

[71] R. Ramakrishnan and A. Silberschatz, "Scalable integration of data collections on the web," Univ. of Wiscosin-Madison, Tech. Rep., 1998.

[72] T. Risch and V. Josifovski, "Distributed data integration by object-oriented mediator servers," *Concurrency and Computation: Practice and Experience*, vol. 13, no. 11, pp. 933–953, Sept. 2001. [Online]. Available: http://www3.interscience.wiley.com/cgi-bin/fulltext?ID=85012147&PLACEBO=IE.pdf;http://www3.interscience.wiley.com/cgi-bin/abstract/85012147/START

[73] K. Runapongsa, J. M. Patel, H. V. Jagadish, Y. Chen, and S. Al-Khalifa, "The michigan benchmark: Towards XML query performance diagnostics," in *Proceedings of the 29th VLDB Conference*, Berlin, Germany, 2003.

[74] A. Sahuguet and B. Alexe, "Sub-document queries over xml with xsquirrel," in *In Proceedings of the 14th International World Wide Web Conference*. Chiba, Japan: The International World Wide Web Conference Committee (IW3C2), May 2005. [Online]. Available: citeseer.ist.psu.edu/article/sahuguet05subdocument.html

[75] A. Schmidt, F. Waas, M. Kersten, M. Carey, I. Manolescu, and R. Busse, "Xmark: A benchmark for XML data management," in *Proceedings of International Conference on Very Large Databases (VLDB)*, Hong Kong, China, Aug. 2002, pp. 974–985. [Online]. Available: citeseer.ist.psu.edu/schmidt02xmark.html

[76] K. Slonneger and B. L. Kurtz, *Formal Syntax and Semantics of Programming languages: A Laboratory Based Approach*, T. Stone, Ed. Addison-Wesley, 1995.

[77] "XMLBench document model benchmark," Sosnoski Software Solutions, Inc. http://www.sosnoski.com/opensrc/xmlbench/, Sosnoski Software Solutions, Inc.

[78] V. Subrahmanian, S. Adali, A. Brink, J. J. Lu, A. Rajput, T. J. Rogers, R. Ross, and C. Ward, "HERMES: A heterogeneous reasoning and mediator system," University of Maryland, Tech. Rep., 1996. [Online]. Available: http://www.cs.umd.edu/projects/hermes/overview/paper/

[79] K. Swigger, R. Brazile, X. Peng, and B. Harrington, "Computer-supported collaboration and the effects of culture," in *In Proceedings of the 6th International Conference on the Design of Cooperative Systems, COOP'04*, Hyeres, France, May 2004.

[80] A. Tomasic, L. Raschid, and P. Valduriez, "Scaling access to heterogeneous data sources with DISCO," *Knowledge and Data Engineering*, vol. 10, no. 5, pp. 808–823, 1998. [Online]. Available: citeseer.ist.psu.edu/tomasic98scaling.html

[81] C. F. Vasters, *BizTalk Server 2000: A Beginner's Guide*. McGraw-Hill Companies, June 2001.

[82] "Well-founded relation," http://en.wikipedia.org/wiki/Well-founded_relation, The Wikimedia Foundation, Inc., July 2006, wikipedia, the free encyclopedia. [Online]. Available: http://en.wikipedia.org/wiki/Well-founded_relation

[83] G. Winskel, *The Formal Semantics of Programming Languages: An Introduction*, ser. Foundations of Computing. Cambridge, Massachusetts: The MIT Press, 1993.

[84] "Extensible markup language (XML) 1.1," http://www.w3.org/TR/2004/REC-xml11-20040204/, The world wide Web Consortium, w3C Recommendation. [Online]. Available: http://www.w3.org/TR/2004/REC-xml11-20040204/

[85] "XML schema part 0: Primer," http://www.w3.org/TR/xmlschema-0/, The world wide Web Consortium, w3C Recommendation. [Online]. Available: http://www.w3.org/TR/xmlschema-0/

[86] "XML syntax for XQuery 1.0 (XQueryX)," http://www.w3.org/TR/xqueryx/, The world wide Web Consortium, w3C working draft. [Online]. Available: http://www.w3.org/TR/xqueryx/

[87] "XQuery 1.0: An XML query language," http://www.w3.org/TR/xquery/, The world wide Web Consortium, w3C working draft. [Online]. Available: http://www.w3.org/TR/xquery/

[88] "XQuery 1.0 and xpath 2.0 formal semantics," http://www.w3.org/TR/xquery-semantics/, The world wide Web Consortium, w3C working draft. [Online]. Available: http://www.w3.org/TR/xquery-semantics/

[89] B. B. Yao, M. T. zsu, and N. Khandelwal, "Xbench benchmark and performance testing of XML DBMSs," in *In Proceedings of the 20th International Conference on Data Engineering*, Boston, MA, March 2004, pp. 621–632.

[90] G. Zhou, R. Hull, R. King, and J.-C. Franchitti, "Using object matching and materialization to integrate heterogeneous databases," pp. 59–76, 1999.