

RELATIONAL VIEWS OF XML FOR THE SEMANTIC WEB

by

Shruti Atre

A thesis submitted to the School of Computing
In conformity with the requirements for
the degree of Master of Science

Queen's University
Kingston, Ontario, Canada
(September 2007)

©Shruti Atre, 2007

Abstract

The Semantic Web is the future of the Internet. It is the extension to the Internet in which information will be given well-defined meaning, enabling not only humans but also machines to find, share and combine information more easily. In the Semantic Web documents are not merely pages containing a set of words that form their content. They also encode the meaning and structure of those words. This enables various information retrieval techniques to be performed on the documents in addition to the ones restricted to keywords. The goal of this research is to explore a method for querying the Semantic Web using relational database theory and source transformation techniques. We take as input, documents annotated with XML mark-up and the information tags that we are interested in. We then extract and populate a relational view on the annotated XML documents using these tags and the implicit relations in the XML documents. We evaluate the feasibility of our system by testing on a variety of input and we also explore the kinds of queries that can be made on the extracted relational view.

Acknowledgements

It is my pleasure to thank all the people who made this thesis possible.

It is difficult to express completely, my gratefulness to both my supervisors, Dr. Jim Cordy and Dr. Pat Martin. If it were not for their enthusiasm, encouragement, sound advice and excellent ideas, this thesis would not have been possible. I would have been totally lost without their invaluable guidance.

I would like to thank all the teachers who have taught me since I was a child. This includes all the teachers from primary school through to my undergraduate years. You have all played a crucial role in bringing me to where I am.

I wish to thank all my close friends (you know who you are), for laughing at my poor jokes, for listening to me whine, for being there when I needed to vent, for providing me entertainment when I needed it most, for giving me all the good things (and the bad things) only a friend could provide. I wish to also thank my entire extended family for providing a loving and supportive environment even though they were thousands of miles away. I would especially like to mention my grandmother, Dr. Pratibha Mohgaonkar who has been one of the most influential people in my life.

Lastly and most importantly, I wish to thank my parents, Dr. Sandhya Atre and Mr. Nandu Atre for raising me, supporting me, teaching me and loving me. I want to thank my sister Dhanashri Atre and my very best friend Ravi Parupudi, for all the emotional support, camaraderie, and caring they have provided all through the years. It is to four of them that I dedicate this thesis.

Table of Contents

ABSTRACT	II
ACKNOWLEDGEMENTS	III
TABLE OF CONTENTS	IV
LIST OF FIGURES	VII
LIST OF TABLES	IX
CHAPTER 1 INTRODUCTION	1
1.1 MOTIVATION.....	1
1.2 PROBLEM / RESEARCH STATEMENT.....	2
1.3 CONTRIBUTIONS.....	3
1.4 OUTLINE OF THESIS.....	4
CHAPTER 2 BACKGROUND	5
2.1 SEMANTIC WEB.....	5
2.2 XML AND RELATED TECHNOLOGIES	8
2.2.1 XML Path Language (XPath)	8
2.2.2 XML Query Language (XQuery).....	9
2.3 RELATIONAL DATABASE SYSTEMS AND SQL	9
2.4 TXL.....	10
2.5 RELATED WORK.....	11
2.5.1 XML Publishing.....	12
2.5.2 XML Storage.....	13
2.5.3 XML Support In Major Database Vendors	15
2.5.4 CERNO, University Of Trento	18
CHAPTER 3 THE SYSTEM: ARCHITECTURAL OVERVIEW	20
3.1 ARCHITECTURE OF THE SYSTEM.....	20
3.2 OVERVIEW OF THE TRANSFORMATION.....	22
3.3 SUMMARY	24

CHAPTER 4 IMPLEMENTATION OF THE TRANSFORMATION PROCESS.....	25
4.1 OVERVIEW OF THE TRANSFORMATION STAGES	25
4.2 LANGUAGE GRAMMARS	25
4.3 DETAILED DESCRIPTION OF THE STAGES.....	28
4.3.1 Stage 1 – Marking Unique Identifiers.....	28
4.3.2 Stage 2 – Treating Implicit Relationships.....	32
4.3.3 Stage 3 – Tag Extraction.....	39
4.3.4 Stage 4 – Generation of Intermediate Representation.....	41
4.3.5 Stage 5 – Generation of SQL Statements.....	47
4.4 SUMMARY	51
CHAPTER 5 EVALUATION OF THE APPROACH	52
5.1 GOALS OF THE EXPERIMENT	52
5.1.1 Feasibility Of The Approach.....	52
5.1.2 Executing Different Kinds Of Queries	53
5.1.3 Test Data.....	54
5.2 CERNO OUTPUT.....	54
5.2.1 Advertisements	55
5.2.2 Biblio.....	59
5.2.3 HIPPA.....	64
5.2.4 Tourist Boards	69
5.3 STANDARD XML DATA SAMPLES	74
5.3.1 Create Relational View.....	75
5.3.2 Example Queries.....	77
5.4 BASIC STATISTICAL INFORMATION ABOUT THE EVALUATION.....	80
5.5 SUMMARY	83
CHAPTER 6 CONCLUSION.....	84
6.1 SUMMARY OF EXPERIMENTS.....	84
6.2 LIMITATIONS AND FUTURE WORK	85
6.3 THESIS CONCLUSION.....	87
REFERENCES.....	90

APPENDIX A TXL GRAMMARS	95
APPENDIX B TXL PROGRAMS.....	99

List of Figures

Figure 2.1: The Semantic Web Stack [15].....	6
Figure 2.2: The TXL process. [5]	10
Figure 2.3: The TXL program structure. [11].....	11
Figure 2.4: Example to illustrate lightweight semantic markup	19
Figure 3.1: System Diagram	21
Figure 3.2: Input XML data	23
Figure 3.3: Output SQL Statements for interesting tags ‘type’, ‘location’ and ‘price’.....	24
Figure 4.1: Part of the TXL grammar for XML data	26
Figure 4.2: Part of the grammar for the SQL statements	27
Figure 4.3: Example of grammar overrides in TXL.	28
Figure 4.4: Sample input data	29
Figure 4.5: Input data with all tags marked with a “tagId”.....	30
Figure 4.6: Rule to mark the tags with unique identifiers.....	31
Figure 4.7: Data after containment relationship has been made explicit	33
Figure 4.8: TXL function that adds a new tag to indicate containment.....	35
Figure 4.9: Data after sequence relationship has been made explicit	36
Figure 4.10: TXL rule that marks up the sequences in an input XML document.....	38
Figure 4.11: Data snippet from the tag files for extracted tags “type”, “price” and “location” from input shown in Figure 4.5.	39
Figure 4.12: Data snippet from the file for extracted tag ‘contained_elem’	40
Figure 4.13: Data snippet from the file for extracted tag ‘sequence_elem’	40
Figure 4.14: TXL rule to transform tags matching passed tag name to interesting elements.....	41
Figure 4.15: Intermediate representation of the relational schema in XML format for the extracted tags shown in Figures 4.11, 4.12 and 4.13.	43
Figure 4.16: General format of the intermediate database representation	44
Figure 4.17: TXL rule to convert each tag to the intermediate representation for a row in the relational table.....	47
Figure 4.18: SQL CREATE statements for <table_structure> tags shown in Figure 4.15	48
Figure 4.19: All the ‘type’ tags from the input shown in Figure 4.4	48
Figure 4.20: Intermediate representation for table data for “type” tag	49
Figure 4.21: SQL INSERT statements for data shown in Figure 4.20	49

Figure 4.22: TXL function to create an SQL INSERT statement.....	50
Figure 5.1: Extract of the annotated document for online advertisements in Rome.....	55
Figure 5.2: SQL statements generated for the extracted tags ‘ad’, ‘location’, ‘contact’, ‘facility’, ‘price’ and ‘type’.....	57
Figure 5.3: SQL query to find all advertisements and the query results.....	58
Figure 5.4: SQL query and result to find all accommodations of apartment type.....	59
Figure 5.5: Input extract for annotated research papers.....	60
Figure 5.6: SQL statements generated for extracting ‘Biblio_Abstract’, ‘Biblio_Citation’, ‘Biblio_cit_title’, ‘Biblio_cit_year’ and ‘Biblio_Email’ tag elements.....	62
Figure 5.7: SQL query and result to find all citation titles and years.....	63
Figure 5.8: Part of the annotated legal document from the HIPPA set.....	64
Figure 5.9: SQL statements generated for extracted tags ‘Event’, ‘Information’, ‘Actor’, ‘Obligation’ and ‘Continue_Obligation’ tag elements.....	66
Figure 5.10: SQL query and result to find all the continued parts of obligations.....	67
Figure 5.11: SQL query and result to find the obligations of actors that are related to health.....	69
Figure 5.12: Input snippet of the annotated tourist board for Monte Bondone in Trento.....	69
Figure 5.13: SQL statements for extracted tags ‘CAT_Accommodation’, ‘obj_phone’, ‘CAT_Services’, ‘CAT_Culture’, ‘obj_money’ and ‘CAT_FoodAndRefreshment’.....	72
Figure 5.14: SQL query and result to find all food and refreshments that include wine.....	73
Figure 5.15: SQL query and result to find the prices or costs for tourist artifacts that are related to culture.....	74
Figure 5.16: Part of the XML document for a Shakespeare play - Julius Caesar.....	75
Figure 5.17: SQL statements for extracted tags ‘PERSONA’ ‘SPEECH’, ‘SCENE’, ‘LINE’ and ‘SPEAKER’.....	77
Figure 5.18: SQL query and result to find all lines of a speaker in a speech.....	78
Figure 5.19: SQL query and result to find all the lines in speeches by Brutus.....	80

List of Tables

Table 1: Information About The Document Sets Used In Our Evaluation.....	81
Table 2: Summary Of The Input Files Used As Examples In This Chapter.....	82
Table 3: Summary Of The Various Types Of Queries Made On Different Input Documents	83

Chapter 1

Introduction

This chapter provides a preface to the research work we have undertaken. It outlines the incentive for investing our time in this work and also describes the problem that we have attempted to tackle. Section 1.1 provides the motivation for our work while Section 1.2 gives details about the research problem. Section 1.3 outlines the contributions made by our work and Section 1.4 gives an overview of the entire dissertation.

1.1 Motivation

Consider the manner in which query processing is performed on the World Wide Web (Web) [1] today. If the data embedded in the HyperText Mark-up of web pages is to be queried then special-purpose, page-specific parsers need to be used to parse the page before any meaningful queries can be made on the web pages. Ad-hoc queries are limited to simple key word based searches in which documents are treated as pure streams of words. Data stored in databases is queried on the Web using form-based interfaces and the queries used are fixed parameterized queries. Most of the search methods today are limited due to the unstructured nature of the data on the existing Web.

The Semantic Web [2] is an extension to the existing Web. It is a web in which information is given well-defined meaning and semantic relations are established between different pieces of data on the Web. Realization of the Semantic Web is marked with a number of challenges. A major challenge is that of creating efficient and high-precision methods to retrieve relevant information from the documents on the Semantic Web that are annotated according to different ontologies.

Semantic mark-up [3] is the annotation of the text of documents from the World Wide Web with tags that approximate the ontology of the domain. Annotation of all the documents in the World Wide Web in this manner generates a large number of documents with a lot of semantic information in them. In order to make querying these documents efficient, methods are needed to restrict querying to only those parts of the annotated documents that are necessary. In order to

discipline and tame the vast amount of information that will be made available by the Semantic Web, there is a growing need to explore different ways of performing ontology-based information retrieval on such a Semantic Web. This is the motivation for conducting our research.

1.2 Problem / Research Statement

The main aim of this research is to develop an approach to deriving relational views over documents annotated with mark-up in Extensible Markup Language (XML) [4] format in order to enable efficient querying of the information in the documents using Structured Query Language (SQL) [5][6]. We explore the application of source transformation techniques to obtain such a relational view.

Methods and tools that are highly efficient have already been used in the analysis of large amounts of text in the area of software design recovery and source transformation [7][8]. In this research, we intend to explore the use of such methods and tools as the basis of a new method for obtaining a relational view of documents with semantic markup that is in the form of XML. We intend to query the content of these documents using SQL.

Relational database systems have become irreplaceable in the world today. Many applications are built on top of relational database systems and so there is a considerable investment made in these systems. Over the years, relational databases have proved to be very successful when it comes to storing and querying data. SQL is popularly used for this purpose. As compared to query languages for XML such as XPath [9] and XQuery [10], SQL is much more mature and well established. The advantages offered by this well developed technology and the advantages offered by relational databases can be extended to information retrieval on the Semantic Web. This can be achieved by extracting only the data from the Semantic Web that is necessary for resolving a particular problem into a relational database, and then querying it.

For this, we first make explicit the implicit relationships in any XML document and then extract the information tags that we are interested in. The external user or application specifies these tags. We then create an intermediate representation for the relational view or tables from the extracted information. Finally the intermediate representation is transformed into corresponding SQL

statements to create the relational view or tables that are populated with appropriate information. This system is implemented using TXL [11]. It is a useful language for applications and tools that involve source transformation [8].

To understand the benefits of such a method, consider the following example of software companies and staffing agencies. The latter's interest lies in gaining access to job listings on the software companies' web pages and querying the information thus obtained. To obtain the information, staffing agencies would traditionally have to work their way through each company's website in order to compile their own comprehensive database of job listings. However with the Semantic Web, by using the mark-ups available, the process of searching the web pages is made easier. Since the web pages may contain a large number of mark-ups, it is necessary to filter out all the unnecessary mark-ups and the associated information to expedite the searching process. In other words, it is better to populate a relational database with only the data associated with the necessary mark-ups because then the task of querying becomes more focussed and efficient. It is focussed because it extracts only the relevant information and populates a relational database accordingly. It is efficient, not only because it harnesses the use of SQL, a well-established and a mature technology, but also because the information to be searched through is limited.

1.3 Contributions

This research project aims to use source transformation tools and techniques to transform documents annotated with XML markup into SQL statements that will create a relational view for the information tags in which the user or application is interested.

Specific contributions include:

- An automated method to explicate implicit relationships in XML documents
- An automated method to extract given tags as intermediate tables
- An automated method to build an SQL database given a set of such intermediate extracted tables
- A pipelined software architecture to combine the methods used for the transformation

1.4 Outline Of Thesis

This thesis is organized as follows:

- Background information regarding the Semantic Web, XML and related technologies, database management systems as well as TXL will be given in Chapter 2. In addition to this, related work is also discussed in this chapter.
- A brief overview of the implementation of our software system is provided in Chapter 3. This chapter also illustrates the architecture of our system and a short explanation of the different stages of the process is given in this chapter.
- Chapter 4 gives a comprehensive explanation of the transformation process. It also provides details about the implementation of the software system.
- Evaluation by means of different experiments is explained in Chapter 5. This chapter also shows the results of the experiments.
- Chapter 6 summarizes the thesis and touches upon its limitations and the future work that can be undertaken.

Chapter 2

Background

In this chapter, the information that is necessary to better understand the scope of this research is outlined. An introduction to the Semantic Web and the challenges that mark its realization are included in Section 2.2. In the vision of the Semantic Web, semantic-markup is done using XML. A brief overview of XML and related technologies is provided in Section 2.3. Our research looks at extracting a relational view over XML documents using source transformation techniques and tools. Therefore some of the main concepts of relational database theory and SQL are covered in Section 2.4. For the transformation we are using a tool called TXL [11], which is covered in Section 2.5. Finally Section 2.6 gives some details on other work that is related to what we have explored and implemented.

2.1 Semantic Web

The growth of the World Wide Web or Internet has been explosive in the last few years. The Internet is now a common source of information for many people and businesses. It is possible to find out information about almost everything on this network. The story of the Semantic Web starts with the invention of the World Wide Web by Tim Berners-Lee, an Oxford graduate, then a scientist at CERN [12]. Berners-Lee observed that the documents on the World Wide Web or the Internet are suitable for human consumption only. Using the information existing on the World Wide Web today, equipped with the understanding of the keywords to be searched, it is fairly easy for humans to search for and obtain the information they need. However, due to the complex nature of the documents on the World Wide Web and due to the inability of search engines of today to understand the meanings of keywords, it is very difficult for computers or machines to perform similar search tasks. In order to enable computers or machines to harness the large amount of information available on the World Wide Web, this information needs to be made machine-processable. Berners-Lee envisions that in the next stage of the World Wide Web, data available on the Internet will not only be in human-readable form but will also be machine-readable. This vision has led to the birth of the Semantic Web. The Semantic Web is not a replacement for the existing World Wide Web, but is an extension to it. It is an extension in

which a universal medium for information exchange is created by giving semantics to the information or data available in the form of documents on the World Wide Web.

The majority of the documents on the World Wide Web are written primarily in HyperText Markup Language (HTML). This language is mainly useful for describing the visual representation of the objects that make up the document. It is unable to classify or categorize the different parts of the document, especially blocks of text in the document, other than for merely achieving the desired visual display of the document. Another aspect of the existing Web is that there are pieces of information located in different documents that are related to each other in a number of different ways. Currently there is no way of expressing these relations and using these relations in a useful manner. The Semantic Web aims to overcome these issues. The Semantic Web also aims to make the web pages comprehensible to computers, so that newer and more sophisticated tasks can be performed and other tasks like searching and sorting can be done in a standardized manner. The architecture of the Semantic Web, put forth by the W3C, is shown in Figure 2.1 [13][14].

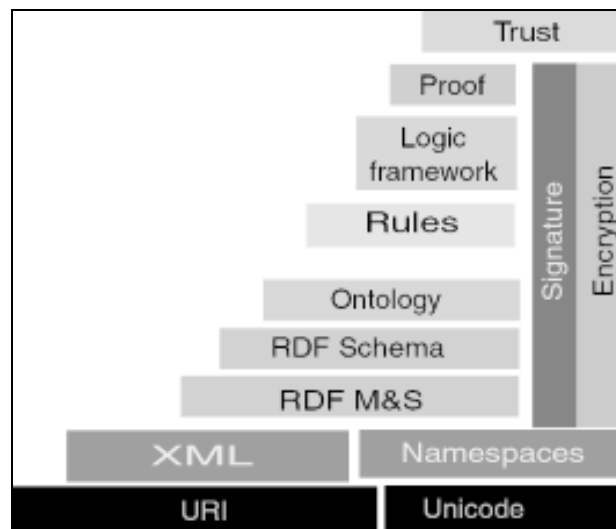


Figure 2.1: The Semantic Web Stack [15]

There are three concepts and related technologies that are important for the development of the Semantic Web. These are:

- Extensible Markup Language (XML)

XML is a technology that enables the creation of tags, that is, hidden labels that are useful for annotating the documents on the Web. These tags can represent the semantic meanings behind the different sections of text. Although XML is useful for adding arbitrary structure to the documents, it does not say anything about the meaning of the structure.

- Resource Description Framework (RDF) [16]

RDF represents the meaning of the structure of the documents as marked by XML. It is a language for representing information about resources on the Web. In RDF information is simply a collection of three tuples like statements with only a subject, verb and object. The triples of RDF form a network of information about related data. However, RDF is unable to relate similar concepts that have different names in different domains.

- Ontologies [2]

Another major component necessary for the successful implementation of the Semantic Web are ontologies to represent the different concepts in multiple domains and their inter-relations. This third component of the Semantic Web provides the mechanism to discover common meanings of concepts in different contexts on the Web.

Although the Semantic Web seems to be very promising, its realization is marked with a number of challenges. Some of these are as given below: [17] [18]

- Development of domain ontologies

Ontologies are an important aspect of the Semantic Web as they allow specification of the semantics of the content of the Semantic Web. Developing common domain ontologies is a social challenge due to the different perspectives for ontologies envisioned by experts in the same domain. Integrating these different ontologies into one huge common one is a major technical challenge.

- Migration of existing Web content to Semantic Web content

Currently very little Semantic Web content is available. One thing that is essential for the successful deployment of the Semantic Web is the migration of existing web content including HTML pages, dynamic content, as well as multimedia and web services, to Semantic Web content. Annotation of the existing Web content is the key to enabling this migration. However, annotation of the existing Web content involves the challenges of integrating domain ontologies and developing an annotation tool.

- Developing techniques for information retrieval on the Semantic Web
If one cannot search through the Semantic Web content (annotated data) then it is useless. A significant effort is needed to develop methods to query the information in addition to organizing the Semantic Web content systematically and storing it.

These challenges will need to be met successfully for a fruitful realization of the Semantic Web. In our research we hope to introduce a technique for information retrieval on the Semantic Web based on extraction of content into a relational database for querying. We want to use source transformation techniques because these techniques have proved to be efficient and fairly accurate for analysis and transformation of large amounts of text and source code.

2.2 XML and Related Technologies

Extensible Markup Language (XML) [4] is a general language created to facilitate the sharing of data across different systems especially across the Internet. XML originated from the Standardized General Markup Language (SGML), a meta-language in which one can define markup languages for documents. XML can be viewed as a flexible text format, which can be used to identify structures in a document. The XML specification [4] defines a standard way to add markup tags to elicit the structures in a richly structured document. XML documents contain markup and content. In the Semantic Web, XML plays an important role because it provides an effective method to markup semantic information in the documents of the Semantic Web. There are many other languages or technologies that were created to work with XML. The two that are relevant to our research are XPath and XQuery.

2.2.1 XML Path Language (XPath)

XPath 1.0 [9] is a language useful for finding elements in an XML document. XPath models the XML document as a tree of different types of nodes. It is called XPath because of its nature to use paths similar to URLs (Universal Resource Locators) for navigating the XML document. XPath uses path expressions to select nodes or sets of nodes in an XML document that satisfy the path expression. It also has many built-in functions for manipulation of strings, numbers and booleans. XPath is very easy to use and a simple expression can solve many queries and computations. Since XPath views the XML document as a tree of nodes, it has major drawbacks. For example,

in the case of large documents, XPath expressions like //abc tend to be inefficient because they require a search of all elements in the document to see if they contain an element abc [19]. Also, it is found that XPath is slow for large documents and draws heavily on resources due the tree-like structure of its data model [20].

2.2.2 XML Query Language (XQuery)

XQuery 1.0 [10] is a language developed to intelligently query XML data sources. It is built on XPath expressions. It shares its data model with that of XPath, that is, the XML document is viewed as a tree of different types of nodes. XQuery borrows the SQL-like syntax for its “FLWOR” expression, which consists of five clauses: FOR, LET, WHERE, ORDER BY, RETURN. XQuery is good for querying data expressed only as XML. It is also useful to query a combination of XML and relational sources. However, this is possible only if there is a way to view the data from the relational sources in XML form. Although XQuery has become popular in the recent days, it does have some disadvantages. Firstly, XQuery does not include features for updating XML documents or databases. Secondly, it also lacks full-text search capability. Lastly, the main disadvantage of XQuery is that it is an immature technology [21]. Many existing applications are not ready to support it.

2.3 Relational Database Systems and SQL

Relational database systems are database management systems based on the relational model first introduced by Edgar Codd [6]. In these systems data is stored in the form of tables and the relations between the data are also stored in tabular form. Relational systems today are a very well recognized and mature technology. Most of them include high-performance query processors and optimizers. The language that is understood by these relational systems is SQL [5]. SQL is a long established standard when it comes to relational database systems. Many developers around the world are well versed with SQL and most of the data repositories and applications already support SQL. There are several development tools available for SQL. SQL has numerous advantages like high-speed query execution and data retrieval, compatibility with other third-party applications, data security, and an ability to perform operations like insert, delete, and updates [22]. However, relational databases have a flat structure. They do not easily support

hierarchy and sequence, which are the two main aspects of XML documents' logical structure. In our research we attempt to overcome this drawback of relational databases.

2.4 TXL

TXL is a language used mainly for source code analysis and source transformations. It is a functional and rule-based language. It uses three phases when transforming input - parsing input into a parse tree according to a specified grammar, transformations on the parse tree using specified transformation rules and un parsing of the new parse tree to generate the result of the transformation. This is shown in Figure 2.2.

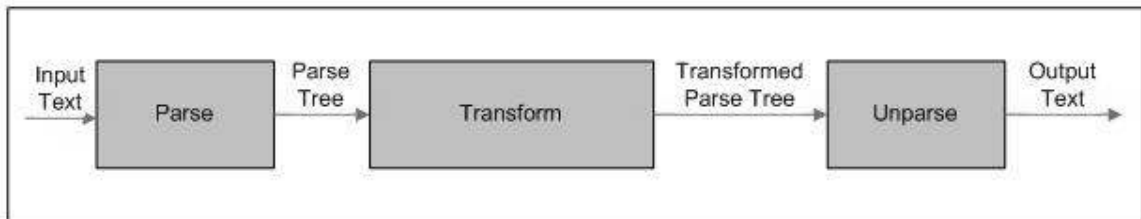


Figure 2.2: The TXL process. [5]

Every TXL program consists of two components. The first component is the grammar that defines how the input should be parsed. The second component is the set of rules and functions that define the transformations to be performed. The structure of a typical TXL program is shown in Figure 2.3. Grammar overrides are used to extend the base grammar so that the resulting output from the transformation can be parsed.

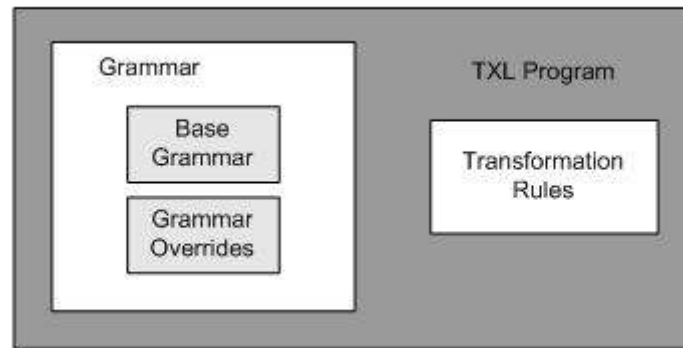


Figure 2.3: The TXL program structure. [11]

Transformations are defined as a set of rules and functions that are applied to the input parse tree. These rules are based on a match-and-replace principle to perform the transformations. The rules match patterns in the input parse tree and replace them with other patterns. The patterns to be matched and replaced are defined by combinations of terminal and non-terminal symbols. The programmer defines these in the transformation rules. The transformation rules can be designed to repeatedly search the scope for matches of the pattern and to replace them accordingly.

TXL is very useful for processing programming languages, program analysis and instrumentation, design recovery and architecture extraction from source, security analysis and prevention of buffer overflow exploits, database and document processing and many other applications [23][24][25][26][27][28].

2.5 Related Work

Storing and processing XML data has been a topic of great interest for a number of years. The database community has looked at reusing relational database management systems (RDBMS) for this because of two main reasons - firstly because adaptation is easier and secondly because hybrid XML-relational systems can store both structured (relational) and unstructured (XML) data [29]. Depending on the application area, the interaction between XML and the RDBMS can be classified into two main categories. One is XML publishing, in which an XML view of

relational data is defined and queries are posed over this view. The other category is XML storage in which a RDBMS is used to store and query existing XML data.

2.5.1 XML Publishing

A number of techniques have been proposed to handle XML publishing. Their approach to defining the XML view of the relational data is classified as local-as-view (LAV), or global-as-view (GAV), or both [30][31][32][33].

XPERANTO [30] is a middleware system working on top of any relational database system and takes the “pure XML”, single query language approach. In this approach XML is used to model both the relational data and the relational meta-data. Therefore the users need to know only XML and the XML query language. An XQuery query is converted into an internal query representation called the XML Query Graph Model (XQGM). View composition and rewrite optimizations are performed to modify the XQGM and then the modified XQGM is then separated into two parts – a SQL query to capture the memory and data intensive processing and a tagger graph structure that is used to construct the XML query result.

SilkRoute [31] is a relational to XML middleware system. SilkRoute differs from XPERANTO because it uses a declarative language called RXL to specify the mapping from the relational schema to the XML view. This mapping is in the form of an RXL query. The RXL query is decomposed into a set of optimal SQL queries over the relational data. The resulting tuples from the SQL queries are then tagged to generate the XML output. SilkRoute aims at publishing XML documents in an environment in which the middleware system has no control over the target RDBMS.

The Agora data integration system [32] is another XML publishing system that enables querying of data in relational as well as tree-structured data using a global XML schema and views of the local data sources. This is known as the LAV (local-as-view) approach. XPERANTO and SilkRoute work only on relational data sources and adopt the GAV (global-as-view) approach as

they generate XML from the tuples. In this system the queries in XQuery are translated into SQL and the results are returned in XML format.

MARS [33] is a system for publishing XML from mixed (relational | XML) storage. The MARS system combines both the LAV and GAV approaches. In MARS the input XQuery query is split into the navigation part and the tagging part by translating it into a set of decorrelated queries (XBind queries). This set of queries along with the integrity constraints on both the XML and the relational data are then compiled to produce relational queries and constraints. An algorithm called ChaseandBackChase (C&B) is then used to obtain a minimal reformulation of the relational queries under the relational integrity constraints.

Since all the above systems publish XML data from given relational data stores, they differ from our work.

2.5.2 XML Storage

Many techniques have been studied to store and query existing XML data using a RDBMS. The work in this area can be typically classified into two different categories. One category includes the techniques that are based on using the XML schema or a DTD (Document Type Definition) [34][35]. The other category includes techniques that use schema-less XML documents [36][37][38][39][40]. Our work falls in the latter category.

In the paper titled “Relational Databases for Querying XML Documents: Limitations and Opportunities” [34], the XML document is stored into relational tables using the corresponding DTD. In this work three techniques are suggested that can be used to choose a relational schema for an XML document based on its DTD – basic inlining, shared inlining and hybrid inlining. The main idea in all three is to embed all elements that have a one-to-one mapping with the parent element as a column in the table for the parent element. If an element has a one-to-many mapping with its sub-elements then a separate table is created for the sub-elements with a foreign key to the parent element.

ShreX [35] is an XML-to-relational mapping system that provides automatic document shredding and query translation capabilities. It is independent of the backend database system that is used. ShreX works by specifying the mapping using annotations in the XML schema of the input XML documents. The annotations specify how the individual elements and attributes in the XML documents are represented in the relational schema. The annotated XML schema is used to shred the XML documents into relational tables. A query translator is used to translate the XML queries to SQL queries over the relational schema. The resulting tuples are then transformed back into XML format.

In the work undertaken by D. Florescu and D. Kossman [36], the XML document is viewed as an ordered and labelled directed graph in which the elements of the XML document are the nodes and the edges represent the element-subelement relationships. An edge-based approach is used in this work and it describes three different ways to map the edges of such a graph along with two different ways for mapping the values of the XML elements.

In STORED [37], a technique to map semi-structured data to relational data is described. The mapping is specified in a declarative query language called STORED. The STORED mapping is generated automatically using data mining techniques and the queries and updates on the semi-structured data are rewritten as those over the corresponding relational data. This method requires you to learn a new language.

The work described in the paper “Storing and Querying Ordered XML Using a Relational Database System” [38] concludes that the relational data model can support XML’s ordered data model by using order as an additional data value in the relational schema. The paper describes three different ways in which order can be encoded into the relational schema and also describes algorithms to translate ordered XPath expressions into SQL using the three encoding methods.

XRel [39] is a path-based approach in which the XML documents are viewed as trees and the elements are stored in relational tables according to the node type and the root-to-node path information. In this method the XML documents are stored in a fixed relational schema.

D. DeHaan, D. Toman, M. P. Consens, and M. T. Özsu suggest a novel approach to storing XML documents in their work [40]. In this, the XML documents are encoded using dynamic intervals. XQuery queries are then translated into relational queries operating on this encoding.

Our method differs from all the above in some way or the other. Firstly, we do not introduce a new language or concept such as in XQGM, RXL or STORED. Secondly, we avoid the storage of XML elements that are not relevant to an application by only storing selective elements. This reduces search scope and makes querying more focused. Thirdly, we use source transformation techniques in our approach, which has not been used in any of the above approaches. Lastly, our approach can be applied to any relational database system. It is therefore independent of the backend database system.

2.5.3 XML Support In Major Database Vendors

In this section a brief overview of how major database vendors support XML is provided. The databases covered are IBM's DB2 Universal Database (UDB), Oracle Database 10g Release 2 and Microsoft's SQL Server 2005.

1. IBM's DB2 Universal Database [41]

IBM's DB2 Universal Database (UDB) has been enhanced with comprehensive native XML support. DB2 unifies native XML storage with relational data storage, indexing and query processing. The native XML support is known as 'pureXML' support. DB2 also provides an XML Schema repository (XSR) to maintain XML schemas that can be used to validate the XML documents. Schema validation is optional and may be performed on a per-document basis. The DB2 server uses a SAX (Simple API for XML) parser to check that incoming documents are well-formed and to perform optional validation.

For native XML support, DB2 introduces a new data type - the XML data type that can be used in CREATE TABLE statements to define one or more columns of type XML. Relational tables can contain any combination of XML columns and relational columns. Each column of type XML can hold one well-formed XML document for every row of the table. In DB2 the relational columns are stored in traditional row structures, while the XML data is stored in hierarchical structures. The text-search capabilities of DB2 have been extended to work with the new XML column type.

In DB2, querying of XML data is done using two languages - XQuery and SQL/XML. DB2 also provides a decomposition product that maps XML data into relational tables. The decomposition process uses annotations inside the XML Schema, similar to schema-annotated mappings in Microsoft SQL Server and Oracle. These annotations are added to the schema by the user and describe the mapping of XML elements and attributes to tables and columns. DB2 automates the decomposition process by using the annotated schema as input.

Thus IBM's DB2 UDB provides a hybrid solution that integrates native XML and XQuery with relational data and SQL.

2. Oracle Database 10g Release 2 [42]

Oracle provides native XML storage and retrieval support through its feature called Oracle XML DB. This feature was first introduced in Oracle Database 9i Release 2. The two main components of Oracle XML DB are the XMLType Storage and the Oracle XML DB Repository. XMLType is a new data type that has been introduced. It allows the database to create a column or table that will contain XML.

The XMLType Storage in Oracle XML DB consists of XMLType tables and views. Oracle XML DB supports both schema-based and schema-less XML storage. When XML schemas are registered with Oracle XML DB, a set of default tables are created and used to store XML instance documents associated with the schema. These can be accessed and viewed in Oracle

XML DB Repository. XMLType tables can be stored as Character Large Objects (CLOBs) or as a set of objects.

The second important component of Oracle XML DB is the Oracle XML DB Repository. This component is optimized for handling all kinds of data including XML data. It contains resources in the form of files and folders (directories, containers). The resources can be accessed in three ways. These are:

- Using SQL, through views RESOURCE_VIEW and PATH_VIEW
- Using PL/SQL, through the DBXML_XDB API
- Using Java, through the Oracle XML DB resource API for Java
- Oracle XML DB also provides a number of XML-specific methods that can operate on XMLType objects.

Thus, with Oracle XML DB you can get the all the advantages of relational database technology along with the advantages of XML.

3. Microsoft's SQL Server 2005 [43]

Microsoft's SQL Server 2005 extends the XML support provided by Microsoft's SQL Server 2000. In SQL Server 2000, XML support is provided at both the server as well as the client side.

At the server side, this support is in the form of "FOR XML" clause in the SELECT statement that can be used to generate XML data from the tables and query results. The converse functionality is rendered by a relational rowset generator function called OpenXML, which extracts values from XML into different fields of a rowset by evaluating XPath expressions.

At the client side XML support is provided through SQLXML, which enables you to create an XML view that is a bi-directional mapping of an XML schema to relational tables. This XML view allows you to work with your existing relational data as if it were an XML document. It also

enables you to convert XML data into relational data and load it into an existing SQL Server database.

SQL Server 2005 extends the XML support described above and also provides native XML support. A new data type called XML is introduced that can appear in the relational table as an additional column. The XML values are stored as large binary objects (BLOBs). An XML data type can be “typed” or “untyped”. The “typed” XML data type is one that conforms to an XML schema. SQL Server 2005 provides support for XML schema through XML schema collections. Built-in methods on the XML data type are provided to query and modify the XML data.

Thus using SQL Server 2005, the user is able to store both relational and XML data within the same database.

2.5.4 CERNO, University Of Trento

The research we have undertaken is based on some past work accomplished at the University of Trento, Italy. In this work, software design recovery techniques have been applied to the common task of semantic markup of documents [2]. The process consists of three steps – parsing, markup and mapping. The parsing step is the one in which a context-free grammar is used to obtain a structure parse of the document under consideration. The markup step involves an initial semantic markup of the document using a wordlist file that contains both positive and negative indicators for semantic categories. The wordlist is obtained from the ontology for that semantic domain. The text is marked up using XML tags. The mapping step is the last in which the XML marked-up text is used to populate a relational database according to a schema approximating the ontology for the domain. This schema is obtained manually from the ontology of the domain. This method of semantic mark-up is lightweight and proves to be fairly accurate. Figure 2.2 shows part of the results obtained after lightweight semantic markup of documents containing advertisements for accommodation in Rome drawn from an online newspaper.

<ad>STA71 AURELIO - PINETA SACCHETTI - <location>5 min walking to SUBWAY CORNELIA LINEA A (3 subway stop from St.Peter Vatican).</location> Small and safe <type>apartment</type> for rent from 5 days to 2 months. Ground Floor and Independent Entry that can accommodate up to three people with <facility>bedroom , bathroom, living room with Kitchen.</facility><price> Prices per day 2 people euro 80.00 - 3 People euro 100.00</price> ALL INCLUDED (<facility>Gas, electricity, Bedlinen, Towel,Tv, Weekly Cleaning Service).</facility> Discount on<term> long term rental. For more details check our web site.<term> Web: <contact>www.camerino-caselunghe.it - E-mail: inforoma@camerino-caselunghe.it.</contact></ad>

<ad>3083 Balduina, <type>studios</type> and 1 <facility>bedroom</facility> <type>apartments</type> available to rent<term> for short and long term periods.<term> All our <type>studios</type> and <type>apartments</type> are fully <facility>furnished with colour T.V.</facility> and direct dial telephone and<price> price</price> includes weekly <facility>linen change, electricity and water bills.</facility> <type>Studios</type> sleep 2, <type>apartments</type> sleep 3, all range in size. <type>Studios</type> from <price>EURO 800 per unit per month or Euro 70 per day for shorter periods.</price> <type>Apartments</type> from <price>EURO 1100 per unit per month or Euro 88 per day for shorter stays. Limited amount of single rooms also available at Euro 530 per month.</price> <facility>Parking and cleaning available upon request.</facility> Balduina is <location>situated in the North of Rome and is well serviced to the centre. The Vatican is a 20 minute walk away and the closest station is a 5 minute stroll away, connecting Balduina to St.Peters, Trastevere, the centre, Fiumicino airport and Temini station.</location> <contact>Contact us on: tel: +39 06 355771 - email: residence.balduina@tin.it.</contact></ad>

<ad>3031 <type>APARTMENT</type> ALBERICO IN <location>CENTRAL ROME</location>: The <type>apartment</type> is <location>located right in the heart of Rome. Within a few minutes walking distance it is possible to reach all of the center of the city. We are just around the corner from St Peter and the Vatican. More! Walking distance to all the major attractions in the city (St Peter, Piazza Navona, Spanish Steps, Colosseum,<location> Trevi Fountain) There are one double <facility>bedroom and one single bedroom. It can accommodate up to 3 guests plus one. Air conditioned. Satellite TV on request.</facility><price> Prices: From 110 euro a day</price> for the whole <type>apartment</type>! For pictures and<price> prices</price> for this <type>apartment</type> please visit: <contact>www.villaughi.com/latium/rome/alberico/index.html - E-mail@info@villaughi.com.</contact></ad>

<ad>3024 <price>850 ^ a month plus 2 months deposit.</price> Photos and map <contact>http://community.webshots.com/user/aurelia38</contact> - The <type>apartment</type> is in a elegant condo with swimming pool and doorman. Very good conditions, <facility>furnished, one bedroom (with big size double bed), living room with kitchen open plan, 2 bathrooms.</facility> <contact>prb@tele2.it.</contact></ad>

Figure 2.4: Example to illustrate lightweight semantic markup

Chapter 3

The System: Architectural Overview

The objective of our work is to develop an approach to construct relational views over documents annotated with mark-up in XML format in order to enable querying of the information in the documents using SQL. The motivation for this work was to research different ways of performing focussed information retrieval on the next generation of the World Wide Web - the Semantic Web. We have used source transformation techniques to obtain a relational view over given XML data. This chapter provides a general overview of the implementation of the software system that has been used for the transformation. A brief overview of the architecture of the system is given in section 3.1. This is followed by an overview of the steps taken in the transformation process in section 3.2. Finally, section 3.3 summarizes the chapter.

3.1 Architecture Of The System

The architecture of the software system is illustrated in Figure 3.1. The system diagram gives an overview of the transformation that is implemented in TXL. Each block shaded grey in the system diagram shown in Figure 3.1 corresponds to a source transformation program. Depending on the platform on which the program is run, the complete TXL transformation is run using a batch file or a shell script. The input given to the software system must be an XML document and a set of tags in which the application or user is interested. The XML document that is provided as input should be one that does not contain a Document Type Definition (DTD) and should be well-formed. A well-formed XML document for the purpose of our software system is one that is made up of elements that consist of a start tag, some tag content and an end tag. There should be no incomplete tags. The software takes the input and applies the different transformations on it to deliver output in the form of a set of SQL statements. The set of SQL statements when run on a database server create and populate relational tables. User queries can then be run on these tables to obtain other information.

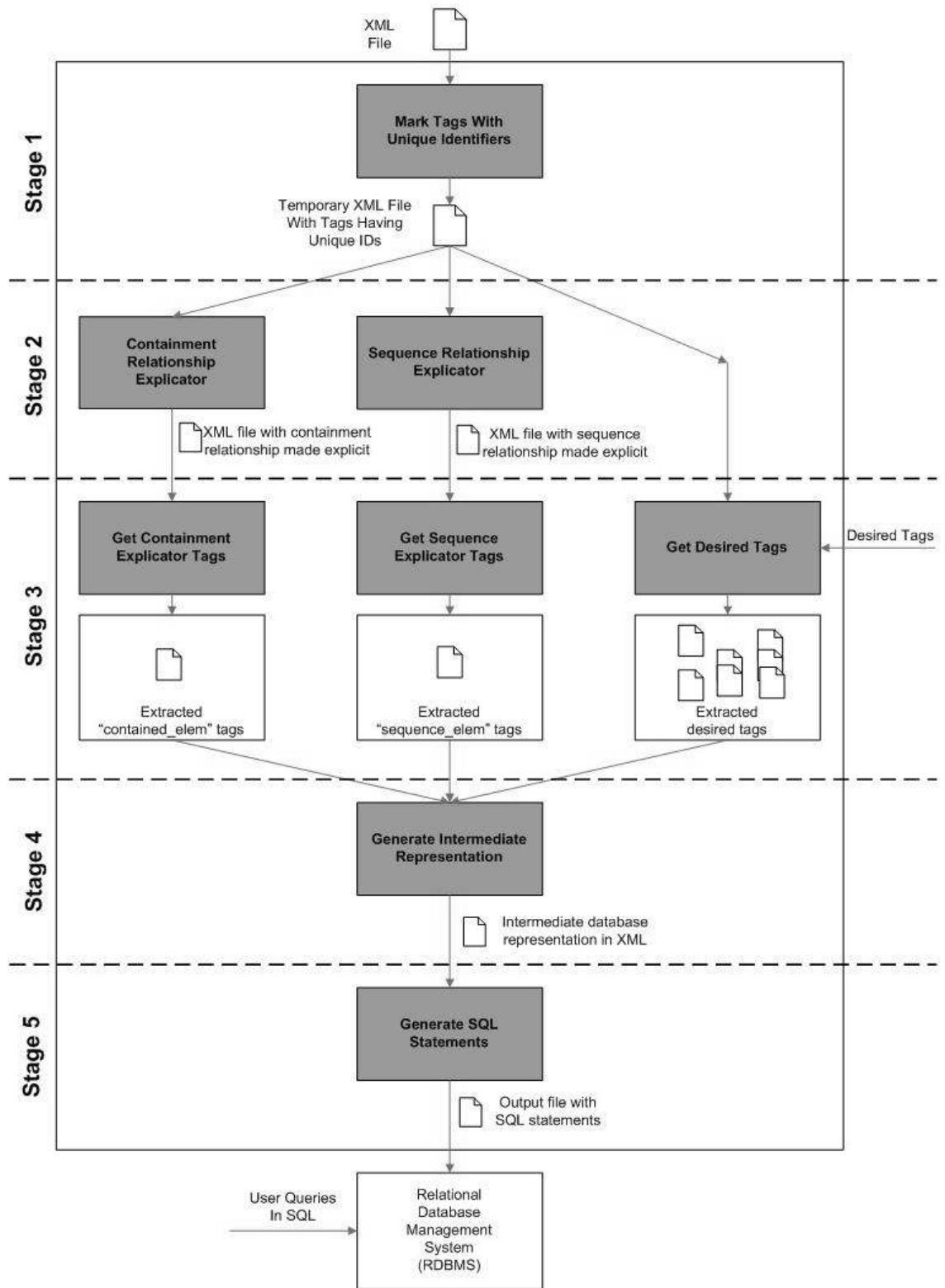


Figure 3.1: System Diagram

The system diagram shows that the architecture of our system is modular. Each different module can also be used independently. The system diagram also shows that, if multiple processors are available, our system can take advantage of parallel processing. The modules called ‘Contain Relation Explicator’, ‘Sequence Relation Explicator’ and ‘Get Interesting Tags’ (one with ‘Interesting Tags’ as input) can be run in parallel before the ‘Generate Intermediate Representation’ block combines their outputs.

3.2 Overview of The Transformation

The entire process is broken down into five different stages. These five stages are shown in Figure 3.1. Source transformations are used in each stage.

In the first stage the input XML file is pre-processed. In this stage all the XML tags in the input file are marked with unique identifiers. The output of this stage is then used as input in the next stages.

Elements in an XML document have certain inherent relationships. Specifically, these are the containment relationship and the sequence relationship. The containment relationship is seen in the nesting of the XML tags and the sequence relationship is seen in the document order, that is, the order in which the different XML tags appear in the document. The second stage of our process makes these implicit relationships explicit. More details about this are given in chapter 4.

The third stage is the stage in which the tags of interest are extracted. Information to keep a track of the implicit relationships is also extracted in this stage. Chapter 4 explains in detail how this is done.

In the fourth stage, an intermediate representation of the relational schema and data is generated from the information extracted in the third stage. This intermediate representation is also in XML format.

SQL statements that can realize the relational schema and data are generated in the fifth stage. This is the last stage. The SQL statements can then be used to create and populate relational

tables in an existing or new database. Currently the SQL statements generated are suitable for MySQL v5.0.41 database server.

For example, if input is the XML data shown in Figure 3.2 and the tags that are interesting are 'type', 'location' and 'price', then the resulting output of the software system would be as shown in Figure 3.3.

```
<ads>
  <ad>STA71 AURELIO - PINETA SACCHETTI - <location>5 min walking to SUBWAY CORNELIA LINEA
  A (3 subway stop from St.Peter Vatican ).</location> Small and safe <type>apartment</type> for rent
  from 5 days to 2 months. Ground Floor and Independent Entry that can accomodate up to three people
  with <facility>bedroom , bathroom, living room with Kitchen.</facility><price> Prices per day 2 people
  euro 80.00 - 3 People euro 100.00</price> ALL INCLUDED (<facility>Gas, electricity, Bedlinen,
  Towel,Tv, Weekly Cleaning Service).</facility> Discount on<term> long term rental. For more details
  check our web site.</term> Web: <contact>www.camerino-caselunghe.it - E-mail: inforoma@camerino-
  caselunghe.it.</contact></ad>

  <ad>3083 Balduina, <type>studios</type> and 1 <facility>bedroom</facility> <type>apartments</type>
  available to rent<term> for short and long term periods.</term> All our <type>studios</type> and
  <type>apartments</type> are fully <facility>furnished with colour T.V.</facility> and direct dial telephone
  and<price> price</price> includes weekly <facility>linen change, electricity and water bills.</facility>
  <type>Studios</type> sleep 2, <type>apartments</type> sleep 3, all range in size. <type>Studios</
  type> from <price>EURO 800 per unit per month or Euro 70 per day for shorter periods.</price>
  <type>Apartments</type> from <price>EURO 1100 per unit per month or Euro 88 per day for shorter
  stays. Limited amount of single rooms also available at Euro 530 per month.</price> <facility>Parking
  and cleaning available upon request.</facility> Balduina is <location>situated in the North of Rome and
  is well serviced to the centre. The Vatican is a 20 minute walk away and the closest station is a 5 minute
  stroll away, connecting Balduina to St.Peters, Trastevere, the centre, Fiumicino airport and Termini
  station.</location> <contact>Contact us on: tel: +39 06 355771 - email: residence.balduina@tin.it.</
  contact></ad>

  <ad>3024 <location>HISTORIC CENTER-SAN SABA Near FAO on Metro B Line</location><price>850
  ^ a month plus 2 months deposit.</price> Photos and map <contact>http://community.webshots.com/
  user/aurelia38</contact> - The <type>apartment</type> is in a elegant condo with swimming pool and
  doorman. Very good conditions, <facility>furnished, one bedroom (with big size double bed), living room
  with kitchen open plan, 2 bathrooms.</facility> <contact>prb@tele2.it.</contact></ad>
</ads>
```

Figure 3.2: Input XML data

```

CREATE TABLE type (
    tagId VARCHAR (50), tag_content TEXT, PRIMARY KEY (tagId));
CREATE TABLE location (
    tagId VARCHAR (50), tag_content TEXT, PRIMARY KEY (tagId));
CREATE TABLE price (
    tagId VARCHAR (50), tag_content TEXT, PRIMARY KEY (tagId));
CREATE TABLE table_elems (
    tableName VARCHAR (50), tagId VARCHAR (50), PRIMARY KEY (tagId));
CREATE TABLE contained_elem (
    childId VARCHAR (50), parentId VARCHAR (50), PRIMARY KEY (parentId, childId),
    FOREIGN KEY (childId) REFERENCES table_elems (tagId), FOREIGN KEY (parentId) REFERENCES
    table_elems (tagId));
CREATE TABLE sequence_elem (
    numInSeq VARCHAR (50), elemId VARCHAR (50), PRIMARY KEY (elemId),
    FOREIGN KEY (elemId) REFERENCES table_elems (tagId));
INSERT INTO type (tagId, tag_content) VALUES ("type1", "apartment");
...
INSERT INTO location (tagId, tag_content) VALUES ("location1", "5 min walking to SUBWAY CORNELIA LINEA A
(3 subway stop from St.Peter Vatican).");
...
INSERT INTO price (tagId, tag_content) VALUES ("price1", "Prices per day 2 people euro 80.00 - 3 People euro
100.00");
...
INSERT INTO table_elems (tableName, tagId) VALUES ("ads", "ads1");
...
INSERT INTO contained_elem (childId, parentId) VALUES ("ad1", "ads1");
...
INSERT INTO sequence_elem (numInSeq, elemId) VALUES ("2", "ad1");
...

```

Figure 3.3: Output SQL Statements for interesting tags ‘type’, ‘location’ and ‘price’

3.3 Summary

In this chapter we have seen a high-level view of the software system used in our research. The implementation of the software system is done using TXL. The next chapter contains an elaborate description of the different stages and also describes the implementation details.

Chapter 4

Implementation Of The Transformation Process

This chapter provides details about the different steps taken in the transformation process for obtaining a relational view over XML data. The transformation is broken down into five different stages. An overview of the stages is given in Section 4.1. The description of each stage along with the implementation detail is given in section 4.2. Finally, section 4.3 summarizes the transformation process.

4.1 Overview of The Transformation Stages

The transformation process consists of five stages that are illustrated in Figure 3.1 in Chapter 3.

The stages are:

- Marking the tags with unique identifiers,
- Treating implicit relationships,
- Desired tag extraction,
- Generation of intermediate representation and
- Generation of SQL statements.

Our transformation process makes use of the divide-and-conquer algorithm design paradigm. In this paradigm a given problem is broken down into smaller sub-problems and the sub-problems are first solved. The solutions to the sub-problems are then put together to produce the solution to the given problem. Accordingly, our transformation process is broken down into five separate stages and the outputs from the individual stages are used to obtain the final output. A source transformation program is associated with each stage. Our system is implemented using TXL - FreeTXL version 10.4e [4].

4.2 Language Grammars

An important aspect of programming source transformation projects is that grammars have to be defined for the languages of the input data as well as the output data of the transformation. TXL

is well suited for defining grammars. The input of our system is a well-formed XML document or a list of XML tags while the output is a sequence of SQL statements that can be executed on a database server. Thus for our system we define two grammars – one for the XML input and the other for the SQL output. An excerpt of the TXL grammar for XML is shown in Figure 4.1 and an excerpt of the TXL grammar for SQL is shown in Figure 4.2. The entire TXL grammars may be found in Appendix A.

```

define program          % The non-terminal that describes the whole input to the programs
  [xmldoc]
end define

define xmldoc
  [opt element]        % 'element' is optional to accommodate blank, empty documents
end define

define element          % An XML element could be tag with some content an empty tag
  [empty_elem_tag]
  | [tag_content]
end define

define tag_content      % XML tag with a start tag, some content and an end tag
  [NL]
  [stag]               [IN]
                       [NL][EX]
  [content]
  [etag]
end define

define content          % The content for the XML element
  [repeat sub_content]
end define

define sub_content      % Sub-content could be some character data, another element or a comment
  [repeat chardata]
  | [element]
  | [comment]
end define

```

Figure 4.1: Part of the TXL grammar for XML data

The TXL grammar that is used to parse the XML input is shown in this figure. The input data is parsed into tokens that are defined in the TXL grammar. In this figure ‘element’, ‘tag_content’, ‘content’, ‘sub_content’ are all non-terminals defined in the grammar. By convention the non-terminal [program] is the goal symbol, that is, the type as which the entire input must be parsed.

```

define sqlStatement                                % Grammar is limited to table definition and insert statements
    [tableDefinition] |
    [insertStatement]
end define

% SQL-Data Definition Language (DDL)
define tableDefinition
    CREATE TABLE [tableName] (                [IN]
        [list tableAttrDefinition]
        [opt primaryKeyDefinition]
        [opt foreignKeyDefinition] ) ;        [EX][NL]
end define

define tableAttrDefinition
    [NL] [tableAttribute] [dataType]
end define

define primaryKeyDefinition
    , [NL] PRIMARY KEY ( [tableAttributelist] )
end define

define foreignKeyDefinition
    , [NL] [list foreignKey] [NL]
end define

define foreignKey
    FOREIGN KEY ( [tableAttributelist] ) REFERENCES [tableName] ( [tableAttributelist] )
end define

% SQL-Data Manipulation Language (DML). Currently grammar for only INSERT is needed.
define insertStatement
    INSERT INTO [tableName] ( [list tableAttribute] ) VALUES ( [list stringlit] ) ; [NL]
end define

```

Figure 4.2: Part of the grammar for the SQL statements

The TXL grammar that is used to parse the SQL output is shown in this figure. The SQL statements are formed using the tokens that are defined here such as 'tableDefinition', 'tableAttrDefinition', 'primaryKeyDefinition', 'foreignKey', 'insertStatement', etc.

Grammar overrides are used in many of our transformation rules to bridge the output of one stage to the input of the next. A grammar override is used to extend the language defined by the base grammar for specific tasks. Grammar overrides can be used to perform certain tasks more efficiently. An example of a TXL grammar override is shown in Figure 4.3.

```

% Grammar overrides for marking interesting elements
redefine element
    ... | [not token] [interesting_element]
end redefine

define interesting_element
    [element]
end define

% Input and output is now either an XML doc or a sequence of interesting elements
redefine program
    ... |
    [repeat interesting_element]
end redefine

```

Figure 4.3: Example of grammar overrides in TXL.

In TXL, previously defined non-terminals can be overridden using the ‘redefine’ statement. For example, in this figure the non-terminals for ‘element’ and ‘program’ (previously defined in Figure 4.1) are overridden to include a repetition of XML tags that match the ‘interesting_element’ non-terminal.

4.3 Detailed Description Of The Stages

This section provides a detailed explanation of each stage in the transformation process. A description of what is accomplished in each stage is given along with the corresponding input and output for the stage. An explanation of how the different source transformations are implemented in TXL is also provided. The source code for the transformations can be found in Appendix B. In our explanation we will consider the input shown in Figure 3.2 as an example. The input is shown once again in Figure 4.4. The desired tags are ‘type’, ‘location’ and ‘price’.

4.3.1 Stage 1 – Marking Unique Identifiers

This is the very first stage of the transformation process. In order to be able to distinguish between the tags in the input XML document it is necessary to identify each tag uniquely. In this stage, the input file is run through a transformation program and all tags are identified. A unique identifier is associated with each tag in the form of an attribute in the XML tag. For example consider the input XML data shown in Figure 4.4.

```

<ads>
  <ad>STA71 AURELIO - PINETA SACCHETTI - <location>5 min walking to SUBWAY CORNELIA LINEA
  A (3 subway stop from St.Peter Vatican ).</location> Small and safe <type>apartment</type> for rent
  from 5 days to 2 months. Ground Floor and Independent Entry that can accomodate up to three people
  with <facility>bedroom , bathroom, living room with Kitchen.</facility><price> Prices per day 2 people
  euro 80.00 - 3 People euro 100.00</price> ALL INCLUDED (<facility>Gas, electricity, Bedlinen,
  Towel,Tv, Weekly Cleaning Service).</facility> Discount on<term> long term rental. For more details
  check our web site.</term> Web: <contact>www.camerino-caselunghe.it - E-mail: inforoma@camerino-
  caselunghe.it.</contact></ad>

  <ad>3083 Balduina, <type>studios</type> and 1 <facility>bedroom</facility> <type>apartments</type>
  available to rent<term> for short and long term periods.</term> All our <type>studios</type> and
  <type>apartments</type> are fully <facility>furnished with colour T.V.</facility> and direct dial telephone
  and<price> price</price> includes weekly <facility>linen change, electricity and water bills.</facility>
  <type>Studios</type> sleep 2, <type>apartments</type> sleep 3, all range in size. <type>Studios</
  type> from <price>EURO 800 per unit per month or Euro 70 per day for shorter periods.</price>
  <type>Apartments</type> from <price>EURO 1100 per unit per month or Euro 88 per day for shorter
  stays. Limited amount of single rooms also available at Euro 530 per month.</price> <facility>Parking
  and cleaning available upon request.</facility> Balduina is <location>situated in the North of Rome and
  is well serviced to the centre. The Vatican is a 20 minute walk away and the closest station is a 5 minute
  stroll away, connecting Balduina to St.Peters, Trastevere, the centre, Fiumicino airport and Termini
  station.</location> <contact>Contact us on: tel: +39 06 355771 - email: residence.balduina@tin.it.</
  contact></ad>

  <ad>3024 <location>HISTORIC CENTER–SAN SABA Near FAO on Metro B Line</location><price>850
  ^ a month plus 2 months deposit.</price> Photos and map <contact>http://community.webshots.com/
  user/aurelia38</contact> - The <type>apartment</type> is in a elegant condo with swimming pool and
  doorman. Very good conditions, <facility>furnished, one bedroom (with big size double bed), living room
  with kitchen open plan, 2 bathrooms.</facility> <contact>prb@tele2.it.</contact></ad>
</ads>

```

Figure 4.4: Sample input data

After identifying each tag, the output of this stage is as shown in Figure 4.5.

```

<ads tagId="ads1">
  <ad tagId="ad1"> STA71 AURELIO - PINETA SACCHETTI -<location tagId="location1"> 5 min walking
  to SUBWAY CORNELIA LINEA A (3 subway stop from St.Peter Vatican).</location> Small and
  safe<type tagId="type1"> apartment</type> for rent from 5 days to 2 months.Ground Floor and
  Independent Entry that can accomodate up to three people with<facility tagId="facility1"> bedroom,
  bathroom, living room with Kitchen.</facility><price tagId="price1"> Prices per day 2 people euro 80.00 -
  3 People euro 100.00</price> ALL INCLUDED (<facility tagId="facility2"> Gas, electricity, Bedlinen,
  Towel, Tv, Weekly Cleaning Service).</facility> Discount on<term tagId="term1"> long term rental.For
  more details check our web site.</term> Web:<contact tagId="contact1"> www.camerino-caselunghe.it -
  E-mail: inforoma @ camerino-caselunghe.it.</contact></ad>

  <ad tagId="ad2"> 3083 Balduina,<type tagId="type2"> studios</type> and 1<facility tagId="facility3">
  bedroom</facility><type tagId="type3"> apartments</type> available to rent<term tagId="term2"> for
  short and long term periods.</term> All our<type tagId="type4"> studios</type> and<type tagId="type5">
  apartments</type> are fully<facility tagId="facility4"> furnished with colour T.V.</facility> and direct dial
  telephone and<price tagId="price2"> price</price> includes weekly<facility tagId="facility5"> linen
  change, electricity and water bills.</facility><type tagId="type6"> Studios</type> sleep 2,<type
  tagId="type7"> apartments</type> sleep 3, all range in size.<type tagId="type8"> Studios</type>
  from<price tagId="price3"> EURO 800 per unit per month or Euro 70 per day for shorter periods.</
  price><type tagId="type9"> Apartments</type> from<price tagId="price4"> EURO 1100 per unit per
  month or Euro 88 per day for shorter stays.Limited amount of single rooms also available at Euro 530
  per month.</price><facility tagId="facility6"> Parking and cleaning available upon request.</facility>
  Balduina is<location tagId="location2"> situated in the North of Rome and is well serviced to the
  centre.The Vatican is a 20 minute walk away and the closest station is a 5 minute stroll away,
  connecting Balduina to St.Peters, Trastevere, the centre, Fiumicino airport and Termini station.</
  location><contact tagId="contact2"> Contact us on: tel: + 39 06 355771 - email: residence.balduina @
  tin.it.</contact></ad>

  <ad tagId="ad3"> 3024<location tagId="location3"> HISTORIC CENTER – SAN SABA Near FAO on
  Metro B Line</location><price tagId="price5"> 850 ^ a month plus 2 months deposit.</price> Photos and
  map<contact tagId="contact3"> http: / / community.webshots.com / user / aurelia38</contact> -
  The<type tagId="type10"> apartment</type> is in a elegant condo with swimming pool and
  doorman.Very good conditions.<facility tagId="facility7"> furnished, one bedroom (with big size double
  bed), living room with kitchen open plan, 2 bathrooms.</facility><contact tagId="contact4"> prb @
  tele2.it.</contact></ad>
</ads>

```

Figure 4.5: Input data with all tags marked with a “tagId”

The “tagId” attribute is used to exclusively identify each tag. The value of this attribute is uniquely derived from the tag name itself. The identifier is created by appending a unique number to the tag name. The resulting file is then used as input to the next stage.

4.3.1.1 Implementation

The transformation rule for marking the tags present in the input with unique identifiers is shown in Figure 4.6.

```

rule main
.
.
.
% Find every element node
replace $ [element]
    <Tag [id] TagAttrs [repeat attribute]>
        Contents [content]
    </Tag>
% Make sure it doesn't already have a tagId attribute
deconstruct not * [attribute] TagAttrs
    'tagId= Value [attvalue]
% Create a new tag id for it that is unique based on its tagname
construct UniqueTagId [id]
    Tag [! ]
construct UniqueTagIdStr [stringlit]
    _ [quote UniqueTagId]
.
.
.
% Replace the node with the tag id as a new attribute
by
    <Tag 'tagId=UniqueTagIdStr TagAttrs>
        Contents
    </Tag>
end rule

```

Figure 4.6: Rule to mark the tags with unique identifiers

In this program, every 'element' in the input is deconstructed to obtain the tag name. The tag name is then used to form the unique identifier using the '!' operator in TXL. Each element found is then replaced by the same tag but with a new attribute that has the unique identifier as its value.

In this rule every element that has a start tag, some content and an end tag is matched and it is replaced by the same element with an additional attribute. The additional attribute is the 'tagId' attribute and its value is the unique identifier for the tag. In Figure 4.6, the following TXL construct is seen:

```

construct UniqueTagId [id]
    Tag [!]

```

This is used for creating the unique identifier. The "!" operator in TXL creates a unique identifier from its operand by appending a unique number to the text of the operand.

4.3.2 Stage 2 – Treating Implicit Relationships

The structure of an XML document conveys semantics. These semantics of the document structure can be deduced from the inherent relationships between the different XML elements in the document. Once all the tags are uniquely identified, the next step is to make these inherent relationships obvious. The first relationship that can be seen is that of containment which is hidden in the nesting of XML elements. The second is that of document order which is hidden in the sequencing of the XML elements.

In this stage we use the technique of exploding the input with artifacts that will make identifying these implicit relationships possible. The artifacts we use are additional XML tags. By introducing new tags into the XML document we explicate the inherent relationships.

4.3.2.1 The Containment Relationship

Let us consider the first type of relationship, that is, the containment relationship. In XML, hierarchy is achieved by nesting tags inside each other. This hierarchy is implied by the containment relationship. The XML document can be viewed as a tree and so every tag in the XML document, except the root tag, can be viewed as a child tag that has a parent tag associated with it. To specify these parent-child relationships we annotate the input with additional XML tags. For example, consider the input data shown in Figure 4.5.

Upon identifying the containment relationship amongst the tags, the input is transformed into another XML document. The output corresponding to input shown in Figure 4.5 is shown in Figure 4.7.

```

<ads tagId="ads1">
  <contained_elem childId="ad1" parentId="ads1">
    <ad tagId="ad1"> STA71 AURELIO - PINETA SACCHETTI -
      <contained_elem childId="location1" parentId="ad1">
        <location tagId="location1"> 5 min walking to SUBWAY CORNELIA LINEA A (3 subway
stop from St.Peter Vatican).</location>
      </contained_elem> Small and safe
      <contained_elem childId="type1" parentId="ad1">
        <type tagId="type1"> apartment</type>
      </contained_elem> for rent from 5 days to 2 months.Ground Floor and Independent Entry that can
accomodate up to three people with
      <contained_elem childId="facility1" parentId="ad1">
        <facility tagId="facility1"> bedroom, bathroom, living room with Kitchen.</facility>
      </contained_elem>
      <contained_elem childId="price1" parentId="ad1">
        <price tagId="price1"> Prices per day 2 people euro 80.00 - 3 People euro 100.00</price>
      </contained_elem> ALL INCLUDED (
      <contained_elem childId="facility2" parentId="ad1">
        <facility tagId="facility2"> Gas, electricity, Bedlinen, Towel, Tv, Weekly Cleaning
Service).</facility>
      </contained_elem> Discount on
      <contained_elem childId="term1" parentId="ad1">
        <term tagId="term1"> long term rental.For more details check our web site.</term>
      </contained_elem> Web:
      <contained_elem childId="contact1" parentId="ad1">
        <contact tagId="contact1"> www.camerino-caselunghe.it - E-mail: inforoma @ camerino-
caselunghe.it.</contact>
      </contained_elem>
    </ad>
  </contained_elem>
  <contained_elem childId="ad2" parentId="ads1">
    <ad tagId="ad2"> 3083 Balduina,
  </contained_elem>
</ads>

```

Figure 4.7: Data after containment relationship has been made explicit

Every tag that is nested inside another tag is enclosed in a new tag that is the containment explicator tag. This new tag has two attributes. The first attribute indicates the identifier of the nested tag, that is, the child tag. The second attribute contains the identifier of the parent tag, that is, the identifier of the tag in which the child tag is nested.

4.3.2.2 Implementation of Containment Relationship

When identifying the containment relationship, the XML document is searched for all matches to the following pattern:

<Parent-Tag Parent-Attributes>

Child Tag 1

Child Tag 2

Child Tag 3

```
...
</Parent-Tag>
```

For every such pattern matched, we enclose each of the child tags in new tags with attributes to indicate the parent tag's id and the child tag's id. This new tag is called "contained_elem". This pattern is replaced by the following pattern:

```
<Parent-Tag Parent-Attributes>
  <contained_elem parentId="ParentTagId" childId="childTagId1">
    Child Tag 1
  </contained_elem>
  <contained_elem parentId="ParentTagId" childId="childTagId2">
    Child Tag 2
  </contained_elem>
  <contained_elem parentId="ParentTagId" childId="childTagId3">
    ChildTag 3
  </contained_elem>
  ...
</Parent-Tag>
```

The function in the TXL program that encloses each child element in a new tag is shown in Figure 4.8.

```

% This function encloses each child tag within a tag that indicates its id as well as the parent's id
function addContainedByTag ParentId [attvalue]
    replace [sub_content]
        <ChildTag [id] ChildAttrs [repeat attribute]>
            ChildContent [content]
        </ChildTag>
    % Extract the Id of the child tag
    deconstruct * [attribute] ChildAttrs
        'tagId= ChildId [attvalue]
    % Make a new child element with 2 new attributes:
    % 'childId' = child's id and 'parentId' = parent's id
    construct NewChild [sub_content]
        <contained_elem 'childId=ChildId 'parentId=ParentId >
            <ChildTag ChildAttrs >
                ChildContent
            </ChildTag>
        </contained_elem>
    by
        NewChild
end function

```

Figure 4.8: TXL function that adds a new tag to indicate containment

This TXL function operates on all the child tags of a parent tag, which is identified by the function's parameter – ParentId. In this function, every child tag found is replaced by a new child tag, which is the containment explicator tag. This new child tag encloses the original child tag.

4.3.2.3 The Sequence Relationship

The other implicit relationship between tags in an XML document is that of document order. The order or sequence in which the tags appear at each level in the document hierarchy plays an important role in understanding the data in the XML document. Similar to the way the containment relationship was explicated, this sequence relationship is also made explicit by exploding the input with additional XML tags, the sequence explicator tags. For example, the input data shown in Figure 4.5 is transformed into the output shown in Figure 4.9.

```

<sequence_elem numInSeq="1" elemId="ads1">
  <ads tagId="ads1">
    <sequence_elem numInSeq="2" elemId="ad1">
      <ad tagId="ad1"> STA71 AURELIO - PINETA SACCHETTI -
        <sequence_elem numInSeq="3" elemId="location1">
          <location tagId="location1"> 5 min walking to SUBWAY CORNELIA LINEA A (3
            subway stop from St.Peter Vatican).</location>
          </sequence_elem> Small and safe
          <sequence_elem numInSeq="4" elemId="type1">
            <type tagId="type1"> apartment</type>
          </sequence_elem> for rent from 5 days to 2 months.Ground Floor and Independent Entry
            that can accomodate up to three people with
          <sequence_elem numInSeq="5" elemId="facility1">
            <facility tagId="facility1"> bedroom, bathroom, living room with Kitchen.
            </facility>
          </sequence_elem>
          <sequence_elem numInSeq="6" elemId="price1">
            <price tagId="price1"> Prices per day 2 people euro 80.00 - 3 People euro
              100.00</price>
          </sequence_elem> ALL INCLUDED (
          <sequence_elem numInSeq="7" elemId="facility2">
            <facility tagId="facility2"> Gas, electricity, Bedlinen, Towel, Tv, Weekly Cleaning
              Service).</facility>
          </sequence_elem> Discount on
          <sequence_elem numInSeq="8" elemId="term1">
            <term tagId="term1"> long term rental.For more details check our web site.
            </term>
          </sequence_elem> Web:
          <sequence_elem numInSeq="9" elemId="contact1">
            <contact tagId="contact1"> www.camerino-caselunghe.it - E-mail: inforoma @
              camerino-caselunghe.it.</contact>
          </sequence_elem>
        </ad>
      </sequence_elem>
      <sequence_elem numInSeq="10" elemId="ad2">
        <ad tagId="ad2"> 3083 Balduina,
        .
        .
        .
      </ads>
    </sequence_elem>
  
```

Figure 4.9: Data after sequence relationship has been made explicit

In this stage, every tag is enclosed in a sequence explicator tag, which has two attributes - one to indicate the sequence number and the other to indicate the corresponding tag.

The “numInSeq” attribute contains the sequence number in the document order for the enclosed tag. Sequencing can be done in a couple of ways for an XML document. In the first way, every new tag seen in the document as you read the document from left to right and top to bottom is given a sequence number that increases in a consecutive manner. This is absolute sequencing. The other method of sequencing is one that takes into account the tree-structured nature of XML documents. It is hierarchical sequencing. In this thesis absolute sequencing is used at every level in the document tree. We use absolute sequencing in our work because then the resulting relation is simplified and contains only one column that indicates the sequence.

The “elemId” attribute contains the unique tag identifier of the enclosed tag element. This attribute is needed so that the sequence explicator tag itself can indicate which tag appears in what sequence order in the document.

4.3.2.4 Implementation of Sequence Relationship

To keep a track of where each element appears in the document order, we identify each element and enclose it within a new tag that has its own attributes. These attributes indicate the enclosed element’s position in the sequence of all elements and thus stores document order. The input is scanned for a simple pattern match as shown below:

```
<Tag Tag-Attributes>  
    Tag-Content  
</Tag>
```

Each pattern match is replaced by the following:

```
<sequence_elem "numInSeq=NumberInSeq" "elemId=TagIdentifier" >  
    <Tag Tag-Attributes>  
        Tag-Content  
    </Tag>  
</sequence_elem>
```

The sequence number value is incremented every time a pattern match is found. The TXL rule that does this replacement is shown in Figure 4.10.

```

rule markSequence
  skipping [element]
  % Find every element node
  replace $ [element]
    <Tag [id] TagAttrs [repeat attribute]>
      Contents [content]
    </Tag>
  % Make sure it is not the sequence_elem tag itself
  deconstruct not Tag
    'sequence_elem
  % Get the tag id for the node under consideration
  deconstruct * [attribute] TagAttrs
    'tagId= TagId [attvalue]

  % Get the sequence number that indicates the element's position in the document order
  import NumInSeq [number]
  construct NumInSeqStr [stringlit]
    _ [quote NumInSeq]
  % Export a new sequence number that indicates the next element's position in the document order
  export NumInSeq
    NumInSeq [+ 1]

  % Replace the original tag with the sequence element for the tag considered
  by
    <sequence_elem 'numInSeq=NumInSeqStr 'elemId= TagId >
      <Tag TagAttrs>
        Contents [markSequence]
      </Tag>
    </sequence_elem>
end rule

```

Figure 4.10: TXL rule that marks up the sequences in an input XML document

This TXL function takes every tag element in the input and replaces it by a new tag, which is the sequence explicator tag. The function imports the sequence number and uses that value in the attribute of the new tag. The original tag is enclosed in the new sequence explicator tag.

With these two implicit relationships explicitly identified it is possible to extract them as relations in a relational schema. It is important to extract these into the relational schema because the manner in which tags are contained in each other and the order in which they appear in a document suggests semantics of the document structure. For example, consider that we have an input document that contains health records of patients and the surgeries they have had in XML form. The containment relationship is used in these documents to associate a surgical procedure with a particular patient by nesting the tag related to the surgical procedure in the tag related to the patient. The sequencing relation within the tag related to the surgical procedure indicates the order in which the steps were taken to perform the surgical procedure. Once these implicit

relationships are extracted into relations, queries like finding out the second step in the surgical procedure for a particular patient can be answered.

Once the tags in an XML document have been uniquely identified and the inherent relationships present in the document have also been identified, the process moves into the next stage.

4.3.3 Stage 3 – Tag Extraction

In this stage only the data associated with the desired tags are extracted, thus reducing the search scope when querying the relational view on the input XML data. A simple scan of the input document is made and pattern matching is used to identify the interesting tags. Interesting tags are defined as the desired tags, that is, those that are relevant to the search, along with the tags that identify the implicit relationships. For our system, an external user or application provides the set of tags that are to be searched, that is, the desired tags. Each interesting tag is extracted into a separate file. Figure 4.11 shows snippets of the files for the extracted “type”, “price” and “location” tags for the input shown in Figure 4.4.

<p>File for “type” contains:</p> <pre><type tagId="type1"> apartment</type> <type tagId="type2"> studios</type> <type tagId="type3"> apartments</type> <type tagId="type4"> studios</type> . .</pre>	<p>File for “price” contains:</p> <pre><price tagId="price1"> Prices per day 2 people euro 80.00 - 3 People euro 100.00</price> <price tagId="price2"> price</price> <price tagId="price3"> EURO 800 per unit per month or Euro 70 per day for shorter periods.</price> <price tagId="price4"> EURO 1100 per unit per month or Euro 88 per day for shorter stays.Limited amount of single rooms also available at Euro 530 per month.</price> <price tagId="price5"> 850 ^ a month plus 2 months deposit.</price></pre>
<p>File for “location” contains:</p> <pre><location tagId="location1"> 5 min walking to SUBWAY CORNELIA LINEA A (3 subway stop from St.Peter Vatican).</location> <location tagId="location2"> situated in the North of Rome and is well serviced to the centre.The Vatican is a 20 minute walk away and the closest station is a 5 minute stroll away, connecting Balduina to St.Peters, Trastevere, the centre, Fiumicino airport and Termini station.</location> <location tagId="location3"> HISTORIC CENTER – SAN SABA Near FAO on Metro B Line</location></pre>	

Figure 4.11: Data snippet from the tag files for extracted tags “type”, “price” and “location” from input shown in Figure 4.5.

Along with the tags specified by the user or the external application, some additional tags also need to be extracted into separate files. These are the tags that are required to identify the containment and sequencing relationships and ones that are needed to maintain the referential integrity in the relational schema. Figure 4.12 shows the contents of the file corresponding to

extracted ‘contained_elem’ tags. Figure 4.13 shows the contents of the file corresponding to extracted ‘sequence_elem’ tags.

```
<contained_elem childId="location1" parentId="ad1"> 5 min walking to SUBWAY CORNELIA LINEA A (3 subway stop from St.Peter Vatican).</contained_elem>
<contained_elem childId="type1" parentId="ad1"> apartment</contained_elem>
<contained_elem childId="facility1" parentId="ad1"> bedroom, bathroom, living room with Kitchen.</contained_elem>
<contained_elem childId="price1" parentId="ad1"> Prices per day 2 people euro 80.00 - 3 People euro 100.00</contained_elem>
+
+
+
```

Figure 4.12: Data snippet from the file for extracted tag ‘contained_elem’.

```
<sequence_elem numInSeq="3" elemId="location1"> 5 min walking to SUBWAY CORNELIA LINEA A (3 subway stop from St.Peter Vatican).</sequence_elem>
<sequence_elem numInSeq="4" elemId="type1"> apartment</sequence_elem>
<sequence_elem numInSeq="5" elemId="facility1"> bedroom, bathroom, living room with Kitchen.</sequence_elem>
<sequence_elem numInSeq="6" elemId="price1"> Prices per day 2 people euro 80.00 - 3 People euro 100.00</sequence_elem>
+
+
+
```

Figure 4.13: Data snippet from the file for extracted tag ‘sequence_elem’.

The set of extracted tags produced by this step is then fed as input to the next stage in the form of temporary files.

4.3.3.1 Implementation

When looking for specific tags, we use the agile parsing paradigm. This paradigm is typically used in programming for software comprehension systems [44]. In agile parsing, the resultant grammar used is made up of the standard base grammar for the input and a set of explicit grammar overrides. This resultant grammar provides a parse of the input that is more convenient or efficient for accomplishing the transformation or analysis task at hand. TXL is well suited for agile parsing and this technique makes extraction of specific tags very simple.

A new non-terminal is introduced as a grammar override for marking out the elements in the input that are interesting. For each specific tag, the input is scanned for a pattern match of the following nature:

<Specific-Tag-Name Tag-Attributes>

Tag Content

</Specific-Tag-Name>

All tags that match are marked as interesting elements using grammar overrides. The TXL grammar override for this is shown in Figure 4.3. The interesting elements are cleaned to remove any other nested tags and are then output to a temporary file. This process is repeated for each tag that is to be searched for, that is, each desired tag. It is also repeated for the containment and sequence explicator tags marked in the second stage. The rule that transforms matching tags to interesting elements is shown in Figure 4.14.

```
% Rule to search for elements that match the passed tag name
rule lookForTag TagName [id]
  replace $ [sub_content]
    <TagName Attrs [repeat attribute]>
      Contents [content]
    </TagName>
  % If the tag matches your search tag name then mark it as an "interesting_element"
  construct InterestingElem [interesting_element]
    <TagName Attrs>
      Contents
    </TagName>
  % Replace the tag by its interesting version and appending contents to respective outputfile
  by
    InterestingElem
end rule
```

Figure 4.14: TXL rule to transform tags matching passed tag name to interesting elements
This TXL function takes a tag name as a parameter and looks for all the tags in the input that match this tag name. All the matching tags found are replaced by the same tag, but the replacement is marked to be of type 'interesting_element'.

The output of this stage includes one file for each of the extracted tags. These files are then passed on as input to the next stage.

4.3.4 Stage 4 – Generation of Intermediate Representation

In order to obtain SQL statements that realize the relational schema over the XML document, we need to first transform the extracted tags to an intermediate format. This intermediate format is also in XML.

In the relational schema that we generate, every interesting tag becomes a relation on its own and each attribute of the tag becomes a column in that relation. This includes relations extracted for the containment and sequence explicator tags. A separate relation called the ‘table_elems’ relation that contains the identifiers of all the tags and the table to which they belong is also created. This relation is created in order to maintain the referential integrity in the relational schema. It ensures that only the tags that are present in the input document appear in the relations corresponding to the containment and sequence explicator tags.

As an example, consider the extracted tags shown in figure 4.11, 4.12 and 4.13. The extracted tags are ‘type’, ‘price’, ‘location’, ‘table_elems’, ‘contained_elem’ and ‘sequence_elem’. The intermediate representation of the relational schema and the relational data corresponding to the extracted tags are shown in Figure 4.15.

```

<database name="db1">
  <table_structure name="type">
    <field Field="tagId"></field>
    <field Field="tag_content"></field>
  </table_structure>
  <table_data name="type">
    <row>
      <field name="tagId"> type1</field>
      <field name="tag_content"> apartment</field>
    </row>
    <row>
      <field name="tagId"> type2</field>
      <field name="tag_content"> studios</field>
    </row>
    ...
  </table_data>
  <table_structure name="location">
    <field Field="tagId"></field>
    <field Field="tag_content"></field>
  </table_structure>
  <table_data name="location">
    ...
  </table_data>
  <table_structure name="price">
    <field Field="tagId"></field>
    <field Field="tag_content"></field>
  </table_structure>
  <table_data name="price">
    ...
  </table_data>
  <table_structure name="table_elems">
    <field Field="tableName"></field>
    <field Field="tagId"></field>
  </table_structure>
  <table_data name="table_elems">
    ...
  </table_data>
  <table_structure name="contained_elem">
    <field Field="childId"></field>
    <field Field="parentId"></field>
  </table_structure>
  <table_data name="contained_elem">
    ...
  </table_data>
  <table_structure name="sequence_elem">
    <field Field="numInSeq"></field>
    <field Field="elemId"></field>
  </table_structure>
  <table_data name="sequence_elem">
    ...
  </table_data>
</database>

```

Figure 4.15: Intermediate representation of the relational schema in XML format for the extracted tags shown in Figures 4.11, 4.12 and 4.13.

The intermediate representation is in XML. XML was chosen to keep both the input and output in the same language. This makes the task of source transformation easier because the same XML grammar can be used to parse both the input and output.

The schema and data for all the tables is enclosed in an outermost root tag called “database”. Within this tag, the “table_structure” tag contains the representation for the relational table schema. The “table_data” tag contains details of the data for the individual rows in each relational table. The general structure of the intermediate representation for the relational schema is shown in Figure 4.16.

```
<database name="nameOfDatabase">
  <table_structure name="TableName1">
    <field Field="FieldName1"></field>
    <field Field="FieldName2"></field>
    ...
  </table_structure>
  <table_data name="TableName1">
    <row>
      <field name="FieldName1"> ContentOfField1 </field>
      <field name="FieldName2"> ContentOfField2 </field>
    </row>
    <row>
      <field name="FieldName1"> ContentOfField1 </field>
      <field name="FieldName2"> ContentOfField2 </field>
    </row>
    ...
  </table_data>
  <table_structure name="TableName2">
    <field Field="FieldName3"></field>
    ...
  </table_structure>
  <table_data name="TableName2">
    <row>
      <field name="FieldName3"> ContentOfField3 </field>
    </row>
    ...
  </table_data>
  ...
</database>
```

Figure 4.16: General format of the intermediate database representation

4.3.4.1 Implementation

Typically in source transformation programs, the input language is first converted to an intermediate form and this intermediate form is then converted to the output language. In accordance with this technique, before generating the SQL statements an intermediate representation for the relational schema and the relational data is created. This is created using the matched tags' files generated in the earlier stages of the transformation process.

We generate a simplified schema in our system. In this schema every desired tag becomes a relation, that is a table, in the relational schema. This table has a minimum of two fields, one to store the tag identifiers and the other to store the tag content. Every other attribute of the extracted tag becomes an additional field in the relational table. The schema for each desired tag is as given below:

```
TagName(  
    tagId,  
    tag_content,  
    Other attributes as other fields,  
    primary key: tagId)
```

The other relations present in the schema are those that maintain referential integrity and those that are necessary to record the implicit relationships in the XML document, namely containment and sequencing. The schema for these tables is given below:

```
table_elems(  
    tableName,  
    tagId,  
    primary_key: tagId  
)  
contained_elem(  
    parentTagId,  
    childTagId,  
    primary_key: parentTagId, childTagId  
    foreign_key: parentTagId on table_elems(tagId) and childTagId on  
    table_elems(tagId)
```

```
)  
sequence_elem(  
    numberInSequence,  
    elementId,  
    primary_key: elementId,  
    foreign_key: table_elems(tagId)  
)
```

The method for generating the intermediate representation is the same for the desired tags, the tags that maintain referential integrity and the tags that record the implicit relationships. Each file corresponding to an extracted tag is read. In our database schema, every extracted tag becomes a relational table. Attributes of all the tags in the considered file are obtained. Each attribute becomes a field in the relational table created for the corresponding extracted tag. An intermediate representation for the relational table and all its fields is generated in this stage. The ‘table_structure’ tags in Figure 4.15 are examples of the intermediate representation for the relational table schema.

The intermediate representation for the data in the relational tables is also obtained in a similar manner. Each file corresponding to an extracted tag that has been read is considered. Every tag in this file corresponds to a row in the relational table for that tag. This relational data has an intermediate representation. The ‘table_data’ tags in Figure 4.15 are examples of the intermediate representation for the relational table data.

The rule that converts each tag to intermediate representation for the rows in the relational table is shown in Figure 4.17.

```

% Rule to form the rows of the table from the interesting elements
rule convertToRowFormat
  skipping [element]
  replace $ [element]
    <TagName [id] Attrs [repeat attribute]>
      Content [repeat sub_content]
    </TagName>
  construct AttrFields [repeat sub_content]
    _ [makeTableField 'name each Attrs ]

  % This field should be added ONLY for tags other than contained_elem and sequence_elem
  construct ContentField [sub_content]
    _ [getLastFieldContent TagName Content]

  construct AllFields [repeat sub_content]
    AttrFields [. ContentField]
  by
    <row> AllFields </row>
end rule

```

Figure 4.17: TXL rule to convert each tag to the intermediate representation for a row in the relational table

In this TXL rule, every tag element found is deconstructed to obtain the tag attributes. Each attribute's value becomes the data in the corresponding field of the relational table. The 'tag_content' field is created only for the relations corresponding to the desired tags. All the fields and their data are then enclosed in <row></row> tags.

The XML data that is the intermediate representation for the relational table schema and the corresponding relational table data becomes the input for the next stage.

4.3.5 Stage 5 – Generation of SQL Statements

The last stage of the transformation process is one in which the SQL statements are actually generated. These statements, when run on a database server, create and populate a relational schema, which can be queried using SQL. An SQL CREATE statement is generated for all the <table_structure> tags. The <field> tags in this tag specify the names of the columns of the table. Consider the <table_structure> tags shown in the data shown in Figure 4.15. This figure shows the XML representation for the table for the tags 'type', 'price' and 'location'. These are the desired tags extracted from the input document. It also shows the XML representation for the relational tables corresponding to the 'table_elems', 'contained_elem' and 'sequence_elem' tags. These are the tags that store the implicit relationships identified in the third stage.

To realize the relational tables corresponding to the <table_structure> tags in Figure 4.15 we need to execute the SQL statements shown in Figure 4.18.

```
CREATE TABLE type (  
    tagId VARCHAR (50), tag_content TEXT, PRIMARY KEY (tagId));  
CREATE TABLE location (  
    tagId VARCHAR (50), tag_content TEXT, PRIMARY KEY (tagId));  
CREATE TABLE price (  
    tagId VARCHAR (50), tag_content TEXT, PRIMARY KEY (tagId));  
CREATE TABLE table_elems (  
    tableName VARCHAR (50), tagId VARCHAR (50), PRIMARY KEY (tagId));  
CREATE TABLE contained_elem (  
    childId VARCHAR (50), parentId VARCHAR (50), PRIMARY KEY (parentId, childId),  
    FOREIGN KEY (childId) REFERENCES table_elems (tagId), FOREIGN KEY (parentId) REFERENCES  
    table_elems (tagId));  
CREATE TABLE sequence_elem (  
    numInSeq VARCHAR (50), elemId VARCHAR (50), PRIMARY KEY (elemId),  
    FOREIGN KEY (elemId) REFERENCES table_elems (tagId));
```

Figure 4.18: SQL CREATE statements for <table_structure> tags shown in Figure 4.15

After creating the relational tables it is necessary to populate them with relevant data. The data contained in the attribute fields are the values of the attributes. Consider the ‘type’ tags shown in Figure 4.19. These are extracted from the input shown in Figure 4.4. The corresponding intermediate representation is shown in Figure 4.20.

```
<type tagId="type1"> apartment</type>  
<type tagId="type2"> studios</type>  
<type tagId="type3"> apartments</type>  
<type tagId="type4"> studios</type>  
<type tagId="type5"> apartments</type>  
<type tagId="type6"> Studios</type>  
<type tagId="type7"> apartments</type>  
<type tagId="type8"> Studios</type>  
<type tagId="type9"> Apartments</type>  
<type tagId="type10"> apartment</type>
```

Figure 4.19: All the ‘type’ tags from the input shown in Figure 4.4

```

<table_data name="type">
  <row>
    <field name="tagId"> type1</field>
    <field name="tag_content"> apartment</field>
  </row>
  <row>
    <field name="tagId"> type2</field>
    <field name="tag_content"> studios</field>
  </row>
  <row>
    <field name="tagId"> type3</field>
    <field name="tag_content"> apartments</field>
  </row>
  ...
</table_data>

```

Figure 4.20: Intermediate representation for table data for “type” tag

Each row tag is transformed to an SQL INSERT statement. The resulting set of SQL INSERT statements created is shown in Figure 4.21.

```

INSERT INTO type (tagId, tag_content) VALUES ("type1", "apartment");
INSERT INTO type (tagId, tag_content) VALUES ("type2", "studios");
INSERT INTO type (tagId, tag_content) VALUES ("type3", "apartments");
INSERT INTO type (tagId, tag_content) VALUES ("type4", "studios");
INSERT INTO type (tagId, tag_content) VALUES ("type5", "apartments");
INSERT INTO type (tagId, tag_content) VALUES ("type6", "Studios");
INSERT INTO type (tagId, tag_content) VALUES ("type7", "apartments");
INSERT INTO type (tagId, tag_content) VALUES ("type8", "Studios");
INSERT INTO type (tagId, tag_content) VALUES ("type9", "Apartments");
INSERT INTO type (tagId, tag_content) VALUES ("type10", "apartment");

```

Figure 4.21: SQL INSERT statements for data shown in Figure 4.20

4.3.5.1 Implementation

The intermediate representation of the database in XML format is the input to the program that generates SQL statements. This program parses the input document and generates both SQL CREATE and INSERT statements that are needed to realize the relational schema and data.

The “table_structure” tags in the intermediate representation contains the information required to generate the CREATE statements. When forming the SQL CREATE statements, the data types of the individual table fields are hard coded to be variable character type. The only exception to this

is the field that store the tag's content, which is defined to be of type that can store a large amount of text. In MySQL this type is known as TEXT.

To generate SQL INSERT statements, the intermediate representation is scanned to find all the 'table_data' tags. Each of these tags contains row-by-row data for each table in the relational schema. Figure 4.22 shows the TXL function that creates an SQL INSERT statement.

```
function createInsertFor TableName [id] Row [sub_content]
  deconstruct Row
    <row>
      RowFields [repeat sub_content]
    </row>

  % Form the ( "attribute1" , "attribute2") part of the INSERT statement
  construct ListOfAttrs [list tableAttribute]
    _ [makeAttrList each RowFields]

  % Form the VALUES( "stringlit1" , "stringlit2") part of the INSERT statement
  construct ListOfValues [list stringlit]
    _ [makeValuesList each RowFields]

  replace * [repeat sqlStatement]
    % Nothing
  by
    INSERT INTO TableName ( ListOfAttrs ) VALUES ( ListOfValues ) ;
end function
```

Figure 4.22: TXL function to create an SQL INSERT statement

The TXL function shown here deconstructs the 'Row' parameter to obtain the different fields of the row. Each field and its value, which are both in XML form, are transformed into a form suitable for the SQL INSERT statement. The SQL INSERT statement is then appended to the end of the input in the function scope by replacing nothing by the newly formed SQL statement.

The SQL CREATE and the SQL INSERT statements generated in this last and final stage can be run on a database server to build the database that realizes a relational view over the input XML document. The relational view thus obtained is a focussed view as it only includes the XML tags, in which the user or external application is interested.

4.4 Summary

In this chapter we have covered the entire transformation process for obtaining a relational view over input XML document that is limited to the search tags specified by a user or an external application. Every stage in the transformation process was described in detail and the corresponding implementation was also explained. In the next chapter we will see how this software system performed on sample input data.

Chapter 5

Evaluation Of The Approach

In this chapter we discuss how we tested our transformation approach. Section 5.1 describes the goals of the experiment while section 5.2 elaborates on the results obtained when our transformation system was applied to different input data. In section 5.2, we also show the outcomes of different queries on the relational schema obtained by our transformation system. Section 5.3 summarizes the chapter.

5.1 Goals Of The Experiment

Ours is a proof of concept type of evaluation with two main goals. The first is to test the feasibility of the approach in terms of creating relational views of XML documents. The second goal is to see what kinds of queries could be posed on the relational data that we extract from the input XML documents.

5.1.1 Feasibility Of The Approach

The first goal of our evaluation is to try to see if source transformation techniques can be used to process XML data and obtain a relational view on it for querying using SQL. At the same time, we want to constrain the relational view obtained by limiting the XML tags that are to be included. An external user or application would specify these tags constraints. This enables queries to be more focused and thereby makes the querying process more efficient.

5.1.2 Executing Different Kinds Of Queries

The second goal of our evaluation is to test different kinds of queries on the relational view that we derive on the input XML document. After studying the usage scenarios for XQuery put forth by W3C [45], it was observed that an XML document has many features. The main features are that XML documents are made up text and they contain information, they are tree-structured in nature and they have an inherent document order. We classify the possible queries based on the features of an XML document they use and define four categories.

- Type 1 - Queries that illustrate information gathered from the XML document
- Type 2 - Queries based on the tree-structured nature of the XML document
- Type 3 - Queries based on the document order of the XML document
- Type 4 - Queries that search text in XML document elements

5.1.2.1 Type 1 - Queries that illustrate information gathered from the XML document

These are queries that gather data from the relational view that give information about the contents of the input XML document. For example, find all ‘x’ tag elements and ‘y’ tag elements from the input XML document.

5.1.2.2 Type 2 - Queries based on the tree-structured nature of the XML document

XML documents have a flexible structure and so the ways in which the tag elements are nested is usually quite important. The queries in this category use nesting to establish a relationship amongst the data that is to be retrieved by the queries. For example, find out all ‘x’ tag elements that are contained in ‘y’ tag elements.

5.1.2.3 Type 3 - Queries based on the document order of the XML document

Just as the way in which XML tag elements are nested in an XML document is important, the ways in which the tag elements in an XML document are ordered is also usually important. This category includes queries that take into consideration the order in which the tag elements appear in the input document. For example, find out all ‘x’ tag elements that follow a ‘y’ tag element.

5.1.2.4 Type 4 - Queries that search text in the XML document elements

These are queries that involve string searching and matching in the elements of the XML document. For example, find out all ‘x’ tag elements that contain the string “abc”

In the experimental evaluation we will see some concrete examples of queries from all four categories. Queries may belong to more than one of the categories described above.

5.1.3 Test Data

We applied our software system on a number of document sets with varying schema and markup. In each case, we show that our two goals are met by creating a relational view and then executing sample queries on that view. We use the output files from the work undertaken as part of the CERNO project at the University of Trento. We also test our approach on standard sets of XML files that are generally available. Section 5.2 shows the outcome of our tests on the CERNO output while Section 5.3 shows the outcome of the tests on the standard XML sample files.

5.2 CERNO Output

CERNO [2] is a semi-automatic annotation tool. This tool is used to annotate different text documents using a lightweight pattern-matching approach. The result from the CERNO tool is

typically a set of XML documents that are marked up with various tags that indicate semantics of the contents of the XML tag. We use the results of the CERNO tool applied to advertisements, research papers (Biblio), legal documents (HIPPA) and tourist boards.

5.2.1 Advertisements

Online advertisements for accommodation for some cities in Europe were annotated using CERNO and the results were used as input to our software system. Let us consider the annotated document for advertisements in Rome, Italy. Figure 5.1 shows an extract of the document.

```

<ads_doc>
<ad>
3884 <type><facility><location>Various apartment typologies, all furnished with care and immersed in the lush of
green of Monte Mario, just a few minutes away from downtown</location></facility></type>, <type><price>Hotel
services are included and all apartments have a common and private garden</price></type>, min. stay: 3nights.
<contact><location>St.Andrew's Garden - Via della Camilluccia 180/b - 00135-Rome-Italy - tel. +39.06.305.44.44 -
fax. +39.06.305.55.36 - info@residencestandrews.it - www.residencestandrews.it</location></contact>
</ad>
<ad>
3883 <type><location>Via del Babuino: Flat for rent on a weekly basis</location></type>. Newly refurbished.<facility>
1 bedroom,1 bathroom, living room, kitchen, big terrace</facility>. <facility>Can accommodate up to 4 people
</facility>.Children welcome. Pictures available. <contact>Elisa Call +39 3385227285 or e-mail
yourbabuino@yahoo.com</contact>
</ad>
<ad>
3868 <type><term><location>Flat available from April 1 in Prati, very close to the vatican</location></term></type>,
<location>The exact location is viale Giulio Cesare, corner of Via Ottaviano</location>. <facility>Top floor, two
bedrooms, two bathrooms, and very quiet, sunny</facility>. <price>2,000 euros a month, 800 euros a week</price>.
<contact>alinat@gmail.com</contact>
</ad>
<ad>
3861 <type><location>Julia penthouse is located in the historical center of Rome, between Santa Maria Maggiore
Basilica and Opera Theater in the 6th floor of a 18th neo-renaissance palace</location></type>. <facility><price>4/6
persons, starting from 230€/daily</price></facility>. <contact>info@hoteljulia.it</contact>
</ad>

```

Figure 5.1: Extract of the annotated document for online advertisements in Rome

5.2.1.1 Create Relational View

Let us consider that the tags we are interested in are ‘ad’, ‘location’, ‘contact’, ‘facility’, ‘price’ and ‘type’. We use our software system on the input shown in Figure 5.1 to get the output shown in Figures 5.2. Only a snippet of the output is shown in both figures. Figure 5.2 (a) shows the

SQL CREATE statements generated for the relational schema and Figure 5.2 (b) shows the SQL INSERT statements needed to populate the relational schema.

```
CREATE TABLE ad (  
    tagId VARCHAR (50),  
    tag_content TEXT,  
    PRIMARY KEY (tagId));  
CREATE TABLE location (  
    tagId VARCHAR (50),  
    tag_content TEXT,  
    PRIMARY KEY (tagId));  
CREATE TABLE contact (  
    tagId VARCHAR (50),  
    tag_content TEXT,  
    PRIMARY KEY (tagId));  
CREATE TABLE facility (  
    tagId VARCHAR (50),  
    tag_content TEXT,  
    PRIMARY KEY (tagId));  
CREATE TABLE price (  
    tagId VARCHAR (50),  
    tag_content TEXT,  
    PRIMARY KEY (tagId));  
CREATE TABLE type (  
    tagId VARCHAR (50),  
    tag_content TEXT,  
    PRIMARY KEY (tagId));  
CREATE TABLE table_elems (  
    tableName VARCHAR (50),  
    tagId VARCHAR (50),  
    PRIMARY KEY (tagId));  
CREATE TABLE contained_elem (  
    childId VARCHAR (50),  
    parentId VARCHAR (50),  
    PRIMARY KEY (parentId, childId),  
    FOREIGN KEY (childId) REFERENCES table_elems (tagId),  
    FOREIGN KEY (parentId) REFERENCES table_elems (tagId)  
);  
CREATE TABLE sequence_elem (  
    numInSeq VARCHAR (50),  
    elemId VARCHAR (50),  
    PRIMARY KEY (elemId),  
    FOREIGN KEY (elemId) REFERENCES table_elems (tagId)  
);
```

(a) SQL CREATE statements for realizing the relational schema

```

INSERT INTO ad (tagId, tag_content) VALUES ("ad1", "3884 Various apartment typologies, all furnished with care and
immersed in the lush of green of Monte Mario, just a few minutes away from downtown.Hotel services are included and
all apartments have a common and private garden.min.stay: 3 nights.St.Andrew ' s Garden - Via della Camilluccia 180 /
b - 00135 - Rome-Italy - tel.+ 39.06.305.44.44 - fax.+ 39.06.305.55.36 - info @ residencestandrews.it -
www.residencestandrews.it");
INSERT INTO ad (tagId, tag_content) VALUES ("ad2", "3883 Via del Babuino: Flat for rent on a weekly basis.Newly
refurbished.1 bedroom, 1 bathroom, living room, kitchen, big terrace.Can accommodate up to 4 people.Children
welcome.Pictures available.Elisa Call + 39 3385227285 or e-mail yourbabuino @ yahoo.com");
INSERT INTO ad (tagId, tag_content) VALUES ("ad3", "3868 Flat available from April 1 in Prati, very close to the
vatican.The exact location is viale Giulio Cesare, corner of Via Ottaviano.Top floor, two bedrooms, two bathrooms, and
very quiet, sunny.2, 000 euros a month, 800 euros a week.alinat @ gmail.com");
INSERT INTO ad (tagId, tag_content) VALUES ("ad4", "3861 Julia penthouse is located in the historical center of
Rome, between Santa Maria Maggiore Basilica and Opera Theater in the 6 th floor of a 18 th neo-renaissance palace.4
/ 6 persons, starting from 230 € / daily.info @ hoteljulia.it");
INSERT INTO ad (tagId, tag_content) VALUES ("ad5", "3857 Apartment to rent for holidays in a central area at only 5
minutes ' walk to S.Peter ' s Church and to metro station (3 stops to P.zza di Spagna).It is near to many interesting
touristic points, if you want you can reach also by walk P.zza di Spagna, Castel S.Angelo, Vatican Museums are in the
opposite street.Many restaurants, every kind of shops make of this area the best destination for an holiday.The
apartment is 60 mq, 11rd floor with elevator in a typical 40 th roman building, renewed few years ago, composed by 1
living room with a comfortable double sofa bed, 1 large double bedroom, a single bed if you need, equipped kitchen
with dishwashing machine and washing machine, bathroom with shower.At your disposal also 2 tv color, dvd, stereo,
hair dryer.I can send photos.sabrinaspaletra @ yahoo.it - tel.00393473602487");
INSERT INTO ad (tagId, tag_content) VALUES ("ad6", "3832 PENTHOUSE COLISEUM WITH TERRACE - Beautiful
luxury penthouse with sunny TERRACE and LIFT, in the very centre of Rome next to the COLISEUM, the FORUM and
the CAPITOL.Next to subway, bus, stores and restaurants.Very quiet and cosy, furnished with antiques, on two
floors.Lower floor: living room with kitchenette and a sofa bed for two, there ' s a balcony looking on a garden.Upper
floor: 1 bedroom (double), 1 bathroom (tub) and a beautiful sunny terrace with garden furniture, awning and
plants.Weekly rental: 800, 00 euros all included (gas, electricity, water, heating, bed linen & towels, air conditioning,
TV, cleaning at end); Security deposit 350 euros.Inquiries: serlius @ yahoo.it");
INSERT INTO ad (tagId, tag_content) VALUES ("ad7", "3810 Flat for rent in Rome.Sleeps 4 people, in Porta Pia area,
close to the noble villas of Galleria Borghese and Villa Torlonia.Prices from 70 Euros for one day.appartamentoroma

```

(b) INSERT statements for populating the relational schema

Figure 5.2: SQL statements generated for the extracted tags ‘ad’, ‘location’, ‘contact’, ‘facility’, ‘price’ and ‘type’

5.2.1.2 Example Queries

We executed the following queries on the relational view created for this case.

1. Query 1

Find all the advertisements for accommodation. This query indicates that we need to extract the ‘ad’ tag elements from the input document and retrieve their contents.

The SQL query and the results are shown in Figure 5.3. This query falls under category “Type 1” described in section 5.1. Type 1 queries are those that retrieve information from the input document.

Query results:	
<i>Query:</i>	
SELECT Ad.tagId AS AdID, Ad.tag_content AS AdDetail	
FROM Ad;	
<i>Status:</i> Executed queries: 1 Error: 0	

(a) SQL query to find all advertisements

AdID	AdDetail
ad1	3884 Various apartment typologies, all furnished with care and immersed in the lush of green of Monte Mario, just a few minutes away from downtown. Hotel services are included and all apartments have a common and private garden. min. stay: 3 nights. St. Andrew ' s Garden - Via della Camilluccia 180 / b - 00135 - Rome-Italy - tel. + 39.06.305.44.44 - fax. + 39.06.305.55.36 - info @ residencestandrews.it - www.residencestandrews.it
ad10	3743 S.Maria Maggiore. From 15 May. Completely furnished, 60 sqm apartment, top floor, elevator, large kitchen – living area, cosy mezzanine bedroom, wood-beamed ceiling, TV, Carrier air-conditioner, quiet, sleeps 4 max, € 500 weekly, € 1.400 monthly (all inclusive). Tel. 064465914, marpali @ tiscali.it - www.geocities.com / romeapartment02
ad100	3571 B & B A sunny and huge doubleroom with private bathroom in attic apartment in the heart of Rome. You can share the green and typical Roman terrace and the kitchen of the apartment. Euro 50 daily. Rosemari.a @ libero.it
ad101	3562 SPANISH STEPS, B & B - Accommodation in the very heart of Historic Centre – huge double rooms in comfortable flat, tv, kitchenette, all comforts EURO: 65.00 Double p.N. all incl. TEL.: + 39 - 064818788 / 3495654606 - E-MAIL: arkan.d @ libero.it - http: // web.tiscali.it / accomodation
ad102	3561 HISTORIC CENTRE / NAZIONALE, B & B located in the heart of Rome in sunny top-floor flat, double rooms w.all comforts, tv, air condition, use of kitchenette EURO 65.00 p.N. Double all incl. TEL.: + 39 - 064818788 / 3495654606 - E-mail: arkan.d @ libero.it
ad103	3508 The B & B Oasis is situated in the center of Rome. With short walks you will arrive to the archaeological areas, museums, monuments, basilicas, churches, university, theatres and many restaurants. In a palace of the 1883 we have comfortable and quiet rooms with bath and TV. Familiar atmosphere. www.bboasi.net - E-mail: bboasi @ inwind.it
ad104	3501 B & B in Navona Square. Very comfortable sunny rooms with private bathroom. Single E. 80 double € 110. These prices are fully inclusive of breakfast, taxes and service charges. aniel74 @

(b) Result of the query shown in (a)

Figure 5.3: SQL query to find all advertisements and the query results

2. Query 2

Find all the accommodations that are of apartment type. This query indicates that we need to find all the ‘type’ tag elements that contain the string ‘apartment’.

The SQL query and the results are shown in Figure 5.4. This query falls under ‘Type 4’ because it uses string searching and matching as described in section 5.1.

Query results:	
<i>Query:</i>	SELECT type.tagId AS typeID, type.tag_content AS type FROM type WHERE type.tag_content LIKE '%apartment%' ORDER BY type;
<i>Status:</i>	Executed queries: 1 Error: 0

(a) SQL query to find all ‘type’ tag elements that contain the string “apartment”

typeID	type
type171) The apartment has every kind of shops all around, on the opposite street there are 3 supermarkets and in the back side there is the big market district, there are also banks, postal office, cinemas
type136	25 + sq.mtrs bedroom available in large apartment (120 sq.meters) consisting of living / dining room with large terrace, eat in kitchen (washing machine), office, hallway, entry way, 1.5 bathrooms and public terrace located on the roof
type12	A modern apartment located on the first floor of a nice building, facing a large green square with bus stop (10 minutes by bus to Colosseum)
type30	A modern apartment located on the first floor of a nice building, facing a large green square with bus stop (10 minutes by bus to Colosseum)
type179	A modern apartment located on the first floor of a nice building, facing a large green square with bus stop (10 minutes by bus to Colosseum)
type173	A modern fully equipped apartment located on the first floor of a nice building, facing to a green square with bus stop (10 minutes by bus to Colosseum)
type52	A stylish, modern apartment located on the first floor of a nice building, facing a large green square with bus stop (10 minutes by bus to Colosseum)
type46	A very nice apartment, full furnished and equipped, available for renting in the heart of Roma (Testaccio)
type34	All our studios and apartments are fully furnished with colour T.V.and direct dial telephone and price includes weekly linen change, electricity and water bills
type88	All our studios and apartments are fully furnished with colour T.V.and direct dial telephone and price includes weekly linen change, electricity and water bills
type181	All our studios and apartments are fully furnished with colour T.V.and direct dial telephone and price includes weekly linen change, electricity and water bills

(b) Results of the query shown in (a)

Figure 5.4: SQL query and result to find all accommodations of apartment type

5.2.2 Biblio

Biblio is a collection of documents obtained after running the CERNO annotation tool on research papers. An extract of one of the documents is shown in Figure 5.5.

```

<Biblio_Doc><Biblio_Head><Biblio_Title>Knowledge and Information Systems (2004) 6: 380-401 DOI
10.1007/s10115-003-0135-4</Biblio_Title><Biblio_Authors>Igor Jurisica</Biblio_Authors>
</Biblio_Head><Biblio_Abstract>Knowledge management research focuses on concepts, methods, and tools
supporting the management of human knowledge. The main objective of this paper is to survey basic concepts that
have been used in computer science for the representation of knowledge and summarize some of their advantages
and drawbacks. A secondary objective is to relate these techniques to information science theory and practice. The
survey classifies the concepts used for knowledge representation into four broad ontological categories. Static
ontologies describe static aspects of the world, i.e., what things exist, their attributes and relationships. A dynamic
ontology, on the other hand, describes the changing aspects of the world in terms of states, state transitions and
processes. Intentional ontologies encompass the world of things agents believe in, want, prove or disprove, and argue
about. Finally, social ontologies cover social settings - agents, positions, roles, authority, permanent organizational
structures or shifting networks of alliances and interdependencies. Keywords: Ontologies; Knowledge management;
Knowledge representation; Information science; Information systems </Biblio_Abstract><Biblio_Introduction>
Knowledge management is concerned with the representation, organization, acquisition, creation, use and evolution of
knowledge in its many forms. To build effective technologies for knowledge management, we need to further our
understanding of how individuals, groups and organizations use knowledge. Given that more and more knowledge is
represented in computer-readable forms, we also need to build tools</Biblio_Introduction><Biblio_Conclusion>The
technologies of information systems have been progressing at a rapid pace. Information systems are now being called
upon to support knowledge management and not just to process data or information. Many advances contribute to
taking information systems beyond mere data into the realm of knowledge. These include cooperative query
processing (Chu et al. 1996), similarity-based retrieval and browsing (Jurisica et al. 2000), data mining and knowledge
discovery (Jurisica et al. 2001), text understanding (Hahn et al. 2002; Riloff 1996), data translation services (Gruber
1993), and knowledge sharing (Orthner et al. 1994), to name a few. However, the key to providing useful support for
knowledge management is founded on how meaning is embedded in information models as defined in terms of
ontologies. In this paper, we have surveyed some of the basic concepts found under four ontological categories. We
outlined the benefits and limitations of the ontologybased approach and argued for the need to combine techniques
from information science and information systems.</Biblio_Conclusion><Biblio_Citation>Agostini A, De Michelis G,
Grasso MA, Patriarca S (<Biblio_cit_year>1993</Biblio_cit_year>) <Biblio_cit_title>Reengineering a business
process with an in- </Biblio_cit_title></Biblio_Citation><Biblio_Citation>novative workflow management system: a case
study. Proceedings of the Conference on Cooperative</Biblio_Citation><Biblio_Citation>Office Computing Systems
(COOC'S'93), Hayward, CA, November <Biblio_cit_year>1993</Biblio_cit_year>, pp 154-165</Biblio_Citation>

```

Figure 5.5: Input extract for annotated research papers

5.2.2.1 Create Relational View

Considering that the desired tags are ‘Biblio_Abstract’, ‘Biblio_Citation’, ‘Biblio_cit_title’, ‘Biblio_cit_year’ and ‘Biblio_Email’, the SQL statements generated after running our tool on the input shown in Figure 5.5 are shown in Figure 5.6.

```

CREATE TABLE Biblio_Abstract (
    tagId VARCHAR (50),
    tag_content TEXT,
    PRIMARY KEY (tagId));
CREATE TABLE Biblio_Citation (
    tagId VARCHAR (50),
    tag_content TEXT,
    PRIMARY KEY (tagId));
CREATE TABLE Biblio_cit_title (
    tagId VARCHAR (50),
    tag_content TEXT,
    PRIMARY KEY (tagId));
CREATE TABLE Biblio_cit_year (
    tagId VARCHAR (50),
    tag_content TEXT,
    PRIMARY KEY (tagId));
CREATE TABLE Biblio_Email (
    tagId VARCHAR (50),
    tag_content TEXT,
    PRIMARY KEY (tagId));
CREATE TABLE table_elems (
    tableName VARCHAR (50),
    tagId VARCHAR (50),
    PRIMARY KEY (tagId));
CREATE TABLE contained_elem (
    childId VARCHAR (50),
    parentId VARCHAR (50),
    PRIMARY KEY (parentId, childId),
    FOREIGN KEY (childId) REFERENCES table_elems (tagId),
    FOREIGN KEY (parentId) REFERENCES table_elems (tagId)
);
CREATE TABLE sequence_elem (
    numInSeq VARCHAR (50),
    elemId VARCHAR (50),
    PRIMARY KEY (elemId),
    FOREIGN KEY (elemId) REFERENCES table_elems (tagId)
);

```

(a) CREATE statements for realizing the relational schema

```

INSERT INTO Biblio_Abstract (tagId, tag_content) VALUES ("Biblio_Abstract1", "Knowledge management research focuses on concepts, methods, and tools supporting the management of human knowledge. The main objective of this paper is to survey basic concepts that have been used in computer science for the representation of knowledge and summarize some of their advantages and drawbacks. A secondary objective is to relate these techniques to information science theory and practice. The survey classifies the concepts used for knowledge representation into four broad ontological categories. Static ontologies describe static aspects of the world, i.e., what things exist, their attributes and relationships. A dynamic ontology, on the other hand, describes the changing aspects of the world in terms of states, state transitions and processes. Intentional ontologies encompass the world of things agents believe in, want, prove or disprove, and argue about. Finally, social ontologies cover social settings - agents, positions, roles, authority, permanent organizational structures or shifting networks of alliances and interdependencies. Keywords: Ontologies; Knowledge management; Knowledge representation; Information science; Information systems");
INSERT INTO Biblio_Citation (tagId, tag_content) VALUES ("Biblio_Citation1", "Agostini A, De Michellis G, Grasso MA, Patriarca S (1993) Reengineering a business process with an in-");
INSERT INTO Biblio_Citation (tagId, tag_content) VALUES ("Biblio_Citation2", "novative workflow management system: a case study. Proceedings of the Conference on Cooperative");
INSERT INTO Biblio_Citation (tagId, tag_content) VALUES ("Biblio_Citation3", "Office Computing Systems (COOCS '93), Hayward, CA, November 1993, pp 154 - 165");
INSERT INTO Biblio_Citation (tagId, tag_content) VALUES ("Biblio_Citation4", "Allen J (1984) Towards a general theory of action and time. Artif Intell 23 :123-154");
INSERT INTO Biblio_Citation (tagId, tag_content) VALUES ("Biblio_Citation5", "Ashburner M, Ball CA et al. (2000) Gene ontology: tool for the unification of biology. The Gene Ontology");
INSERT INTO Biblio_Citation (tagId, tag_content) VALUES ("Biblio_Citation6", "Consortium. Nat Genet 25 (1) :25-29");
INSERT INTO Biblio_Citation (tagId, tag_content) VALUES ("Biblio_Citation7", "Baker PG, Brass A, Bechhofer S, Goble C, Paton N, Stevens R (1998) TAMBIS: transparent access to multi-");
INSERT INTO Biblio_Citation (tagId, tag_content) VALUES ("Biblio_Citation8", "ple bioinformatics information sources. An overview. In: Proceedings of the 6 th international conference");
INSERT INTO Biblio_Citation (tagId, tag_content) VALUES ("Biblio_Citation9", "on intelligent systems for molecular biology (ISMB '98). AAAI Press, Menlow Park, CA, pp 25 - 34");
INSERT INTO Biblio_Citation (tagId, tag_content) VALUES ("Biblio_Citation10", "Baker PG, Goble CA, Bechhofer S, Paton NW, Stevens R, Brass A (1999) An ontology for bioinformatics");

```

(b) INSERT statements for populating the relational schema

Figure 5.6: SQL statements generated for extracting 'Biblio_Abstract', 'Biblio_Citation', 'Biblio_cit_title', 'Biblio_cit_year' and 'Biblio_Email' tag elements

5.2.2.2 Example Queries

We executed the following queries on the relational view created for this case.

1. Query 1

Find all the citation titles and the citation years from the research paper. This implies that we need to search for all 'Biblio_cit_title' and 'Biblio_cit_year' tag elements that appear in the same 'Biblio_Citation' tag element.

The SQL query for this and the results are shown in Figure 5.6. This query falls under 'Type 2' described in section 5.1 because we are making use of the contained relationship seen amongst the tag elements.

```

Query results:
Query:
SELECT biblio_cit_title.tag_content AS CitationTitle, biblio_cit_year.tag_content AS Year
FROM biblio_citation, biblio_cit_title, biblio_cit_year, biblio_email, contained_elem AS cont1,
contained_elem AS cont2
WHERE
((cont1.childId = biblio_cit_title.tagId) && (cont1.parentId = biblio_citation.tagId)) &&
((cont2.childId = biblio_cit_year.tagId) && (cont2.parentId = cont1.parentId))
Status: Executed queries: 1 Error: 0

```

(a) SQL query to find all 'Biblio_cit_title' and 'Biblio_cit_year' tag elements that appear in the same 'Biblio_Citation' tag element

CitationTitle	Year
Reengineering a business process with an in-	1993
The landscape of information science: the American Society for Information Science at	1999
b) Models for supporting the redesign of organizational work	1995
Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering	1997
Strategic modeling for enterprise integration	1999
Modelling Trust for System Design Using the i Strategic Actors Framework	2001
Modelling Strategic Actor Relationships to Support Intellectual Property Man-	2001
AI models for business process reengineering	1996
The logical structure of the world: pseudoproblems in philosophy	1967
general chair of the Very Large Databases	2003
From reaction to cognition	1993
CoBase: a scalable and extensible	1996
Representing and using non-functional requirements: a process-oriented approach	1993
From data to knowledge through concept-oriented terminologies: experience with the Med-	2000
glBIS: a hypertext tool for exploratory policy discussion	1988
Geographic information systems (GIS) : new perspectives in un-	1996
Commun ACM 35 (9) :75-90	1992
Goal directed requirements acquisition	1993
Towards a general theory of action and time	1984
Interfaces 25 (3) :42-57	1995
Ontology and terminology servers in agent-based health-care information	1997
Language support for the specification and derivation of concurrent systems	1987
Workshop on comparing description and frame logics	1998
Computer systems and the design of organizational	1988

(b) Result of the SQL query shown in (a)

Figure 5.7: SQL query and result to find all citation titles and years

5.2.3 HIPPA

HIPPA is a set of documents that have been obtained after annotating documents that come from the United States’ “Health Insurance Portability and Accountability Act”. These are the regulations governing health systems in the United States. A snippet of one of the documents that was used as an input for our system is shown in Figure 5.8.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<DOC><Section><Num_Section>§164.102</Num_Section><Title_section> Statutory basis</Title_section>.
<Sentence>The provisions of this part are adopted pursuant to the <Actor>Secretary</Actor>'s authority to prescribe
<Policy>standards</Policy>, requirements, and implementation <Policy>standards</Policy> under part C of title XI of
the <Policy>Act</Policy> and section 264 of Public Law 104-191</Sentence>.

</Section><Section><Num_Section>§164.104</Num_Section><Title_section> Applicability</Title_section>.
<Sentence>Except as otherwise provided, the provisions of this part apply to <Actor>covered entities</Actor>:
<Actor>health plans</Actor>, <Actor>health care clearinghouses</Actor>, and <Actor>health care providers</Actor>
who transmit <Information>health information</Information> in electronic form in connection with any
<Event>transaction</Event> referred to in section 1173(a)(1) of the <Policy>Act</Policy></Sentence>.

</Section><Section><Num_Section>§164.106</Num_Section><Title_section> Relationship to other
parts</Title_section>.
<Sentence>In complying with the requirements of this part, <Actor>covered entities</Actor> are required to comply
with the applicable provisions of parts 160 and 162 of this subchapter</Sentence>.

</Section><Section><Num_Section>§164.500</Num_Section><Title_section> Applicability</Title_section>.
<Sentence><Index>(a)</Index> Except as otherwise provided herein, the <Policy>standards</Policy>, requirements,
and <Policy>implementation specifications</Policy> of this subpart apply to <Actor>covered entities</Actor> with
respect to <Information>protected health information</Information></Sentence>.
<Sentence><Obligation><Index>(b)</Index> <Actor>Health care clearinghouses</Actor> must comply with the
<Policy>standards</Policy>, </Obligation> requirements, and <Policy>implementation specifications</Policy> as
follows:
<Index>(1)</Index> When a <Actor>health care clearinghouse</Actor> creates or receives <Information>protected
health information</Information> as a <Actor>business associate</Actor> of another <Actor>covered entity</Actor>,
the clearinghouse must comply with:
<Index>(i)</Index> Section <CrossRef>164.500</CrossRef> relating to applicability;
<Index>(ii)</Index> Section <CrossRef>164.501</CrossRef> relating to definitions;
<Index>(iii)</Index> Section <CrossRef>164.502</CrossRef> relating to uses and <Event>disclosures</Event> of
<Information>protected health information</Information>, except that a clearinghouse is prohibited from using or
```

Figure 5.8: Part of the annotated legal document from the HIPPA set

5.2.3.1 Create Relational View

The search tags we used in this experiment were ‘Event’, ‘Information’, ‘Actor’, ‘Obligation’ and ‘Continue_Obligation’. Using the data shown in Figure 5.8 as input, the resulting output is shown in Figure 5.9.

```

CREATE TABLE Event (
    tagId VARCHAR (50),
    tag_content TEXT,
    PRIMARY KEY (tagId));
CREATE TABLE Information (
    tagId VARCHAR (50),
    tag_content TEXT,
    PRIMARY KEY (tagId));
CREATE TABLE Actor (
    tagId VARCHAR (50),
    tag_content TEXT,
    PRIMARY KEY (tagId));
CREATE TABLE Obligation (
    tagId VARCHAR (50),
    tag_content TEXT,
    PRIMARY KEY (tagId));
CREATE TABLE Continue_Obligation (
    tagId VARCHAR (50),
    tag_content TEXT,
    PRIMARY KEY (tagId));
CREATE TABLE table_elems (
    tableName VARCHAR (50),
    tagId VARCHAR (50),
    PRIMARY KEY (tagId));
CREATE TABLE contained_elem (
    childId VARCHAR (50),
    parentId VARCHAR (50),
    PRIMARY KEY (parentId, childId),
    FOREIGN KEY (childId) REFERENCES table_elems (tagId),
    FOREIGN KEY (parentId) REFERENCES table_elems (tagId)
);
CREATE TABLE sequence_elem (
    numInSeq VARCHAR (50),
    elemId VARCHAR (50),
    PRIMARY KEY (elemId),
    FOREIGN KEY (elemId) REFERENCES table_elems (tagId)
);

```

(a) CREATE statements for realizing the relational schema

```

INSERT INTO Event (tagId, tag_content) VALUES ("Event1", "transaction");
INSERT INTO Event (tagId, tag_content) VALUES ("Event2", "disclosures");
INSERT INTO Event (tagId, tag_content) VALUES ("Event3", "disclosures");
INSERT INTO Event (tagId, tag_content) VALUES ("Event4", "Data aggregation");
INSERT INTO Event (tagId, tag_content) VALUES ("Event5", "health care operations");
INSERT INTO Event (tagId, tag_content) VALUES ("Event6", "payment");
INSERT INTO Event (tagId, tag_content) VALUES ("Event7", "treatment");
INSERT INTO Event (tagId, tag_content) VALUES ("Event8", "treatment");
INSERT INTO Event (tagId, tag_content) VALUES ("Event9", "treatment");
INSERT INTO Event (tagId, tag_content) VALUES ("Event10", "Disclosure");
INSERT INTO Event (tagId, tag_content) VALUES ("Event11", "Health care operations");
INSERT INTO Event (tagId, tag_content) VALUES ("Event12", "treatment");
INSERT INTO Event (tagId, tag_content) VALUES ("Event13", "treatment");
INSERT INTO Event (tagId, tag_content) VALUES ("Event14", "payment");
INSERT INTO Event (tagId, tag_content) VALUES ("Event15", "marketing");
INSERT INTO Event (tagId, tag_content) VALUES ("Event16", "treatment");
INSERT INTO Event (tagId, tag_content) VALUES ("Event17", "payment");
INSERT INTO Event (tagId, tag_content) VALUES ("Event18", "Marketing");
INSERT INTO Event (tagId, tag_content) VALUES ("Event19", "use");
INSERT INTO Event (tagId, tag_content) VALUES ("Event20", "Marketing");
INSERT INTO Event (tagId, tag_content) VALUES ("Event21", "payment");
INSERT INTO Event (tagId, tag_content) VALUES ("Event22", "treatment");
INSERT INTO Event (tagId, tag_content) VALUES ("Event23", "treatment");
INSERT INTO Event (tagId, tag_content) VALUES ("Event24", "treatment");
INSERT INTO Event (tagId, tag_content) VALUES ("Event25", "treatments");
INSERT INTO Event (tagId, tag_content) VALUES ("Event26", "marketing");
INSERT INTO Event (tagId, tag_content) VALUES ("Event27", "treatment");
INSERT INTO Event (tagId, tag_content) VALUES ("Event28", "Payment");
INSERT INTO Event (tagId, tag_content) VALUES ("Event29", "Payment");
INSERT INTO Event (tagId, tag_content) VALUES ("Event30", "payment");
INSERT INTO Event (tagId, tag_content) VALUES ("Event31", "Disclosure");

```

(b) SQL INSERT statements for populating the relational schema

Figure 5.9: SQL statements generated for extracted tags ‘Event’, ‘Information’, ‘Actor’, ‘Obligation’ and ‘Continue_Obligation’ tag elements

5.2.3.2 Example Queries

We executed the following queries on the relational view created for this case.

1. Query 1

Find all the continued parts of obligations seen in the input documents. This implies that we need to query for all ‘Continue_Obligation’ tag elements that follow an ‘Obligation’ tag element.

The SQL query to achieve this is shown in Figure 5.10. The figure also shows the results of this query. This query falls under the category ‘Type 3’ described in section 5.1 because we are making use of the document order to retrieve the result of the query.

```

Query results:
Query:
SELECT Obligation.tagId AS ObligationID, Obligation.tag_content AS Obligation,
Continue_Obligation.tagId AS Continue_ObligationID, Continue_Obligation.tag_content AS
Continue_Obligation
FROM Obligation, Continue_Obligation, sequence_elem AS seq1,
sequence_elem AS seq2
WHERE (seq1.elemId = Obligation.tagId) &&
(seq2.elemId = Continue_Obligation.tagId) &&
(seq1.numInSeq < seq2.numInSeq)
ORDER BY Continue_ObligationID;
Status: Executed queries: 1 Error: 0

```

(a) SQL Query to find all ‘Continue_Obligation Obligation’ tag elements that follow an ‘Obligation’ tag element

ObligationID	Obligation	Continue_ObligationID	Continue_Obligation
Obligation1	(b) Health care clearinghouses must comply with the standards,	Continue_Obligation1	Its health care component does not disclose protected health information to another component of the covered entity in circumstances in which this subpart would prohibit such disclosure if the health care component and the other component were separate and distinct legal entities;
Obligation10	a covered entity must treat such person as a personal representative under this subchapter,	Continue_Obligation1	Its health care component does not disclose protected health information to another component of the covered entity in circumstances in which this subpart would prohibit such disclosure if the health care component and the other component were separate and distinct legal entities;
	A covered health care provider or health plan must		Its health care component does not disclose protected health information to another

(b) Result of the query shown in (a)

Figure 5.10: SQL query and result to find all the continued parts of obligations

2. Query 2

Find all the obligations of actors that are related to health. This query implies that we need to find all ‘Actor’ element tags in ‘Obligation’ tags that contain the string “health”.

Figure 5.11 shows the SQL query and the results. This query falls under categories ‘Type 2’ and ‘Type 4’ as described in section 5.1 because we are making use of the containment relationship and we are also using string searching and matching to obtain the results.

Query results:	
<i>Query:</i>	
<pre>SELECT Actor.tagId AS ActorID, Actor.tag_content AS Actor, Obligation.tag_content AS Obligation FROM Actor,Obligation, contained_elem WHERE (contained_elem.childId = Actor.tagId && contained_elem.parentId = Obligation.tagId) && (Actor.tag_content LIKE '%health%') ORDER BY Actor;</pre>	
<i>Status:</i> Executed queries: 1 Error: 0	

- (a) SQL query to find all ‘Actor’ element tags in ‘Obligation’ tags that contain the string “health”

ActorID	Actor	Obligation
Actor1371	health care clearinghouse	A health care clearinghouse must comply with the applicable requirements of this subpart no later than February 26,
Actor8	Health care clearinghouses	(b) Health care clearinghouses must comply with the standards,
Actor217	health care provider	A covered health care provider or health plan must comply with the applicable requirements of § 164.522 (b) in communicating protected health information
Actor1364	health care provider	A covered health care provider must comply with the applicable requirements of this subpart no later than February 26,
Actor416	health care provider	a covered health care provider must obtain the individual ' s consent,
Actor564	health care provider	A covered health care provider must inform an individual of the protected health information that it may include in a directory and the persons to whom it may disclose such information (including disclosures to clergy of information regarding religious affiliation) and provide the individual with the opportunity to restrict or prohibit some or all of the uses or disclosures permitted by paragraph (a) (1) of this section
Actor572	health care provider	(ii) The covered health care provider must inform the individual and provide an opportunity to object to uses or disclosures for directory purposes as required by paragraph (a) (2) of this section when it becomes practicable to do so
Actor954	health care provider	A covered health care provider that has a direct treatment relationship with an individual must:
Actor971	health care provider	the covered health care provider must provide electronic notice automatically and contemporaneously in response to the individual ' s first

(b) Result of the query shown in (a)

Figure 5.11: SQL query and result to find the obligations of actors that are related to health

5.2.4 Tourist Boards

CERNO was also used to annotate different online tourist boards. We tested our software system on the resulting annotated documents. Consider the document snippet shown in Figure 5.12, which shows the annotated document for the online tourist board for Monte Bondone in Trento, Italy.

```
<Tourist_Doc>NomeAPT AZIENDA PER IL TURISMOTRENTE MONTE BONDONE Home page:
<obj_webaddress>http://www.apr.trento.it/English/ENG_Home.htm</obj_webaddress>
<CAT_Geography>Capital of Trentino, not far from the Dolomites and the numerous lakes that can be found near by,
Trentis a city of art that...</CAT_Geography><CAT_LocalProducts>Visitors throughout the various seasons can
admire its many folded facets: Città in giardino after the awakening of spring, its tasty products in the autumn period...
</CAT_LocalProducts><CAT_Geography><Mountain>Monte Bondone</Mountain>, a few kilometres away from the
city, is an important nature reserve of specific botanic nature that makes this mountain unique. This lead...
</CAT_Geography><CAT_Culture>Home the Council of Trent (1545 1563), it has always been a bridge between the
Italian and Central European culture. The city offers ...</CAT_Culture><CAT_Geography>Capital of Trentino, not far
from the Dolomites and the numerous lakes that can be found near by, Trentis a city of art that has a strong
Renaissance mark, which characterises it for its colours, its buildings and make it unique in the entire Alpine
arc.</CAT_Geography><CAT_Culture><CAT_LocalProducts>Trentis able tsurprise as it always presents itself in a
new form, maintaining at the same time close links ttradition. Visitors throughout the various seasons can admire its
many folded facets: Città in giardino after the awakening of spring, its tasty products in the autumn period, kept alive by
its historical reenactments during the S Vigillifestival at the end of June or illuminated by the warm <Market>Christmas
market</Market> lights during Advent.</CAT_LocalProducts></CAT_Culture>
<CAT_SportActivities><CAT_ArtisticHeritage><CAT_Geography><Mountain>Monte Bondone</Mountain>, a few
kilometres away from the city, is an important nature reserve of specific botanic nature that makes this mountain
unique. This lead tthe opening of the <Park>Botanical Alpine Garden</Park>, one of the most important in the Alps,
and the Nature reserve of the Tre Cime del <Mountain>Monte Bondone</Mountain>. This tourist destination offers
numerous possibilities for walks in the summer, whilst in the winter it turns into paradise for lovers of skiing and winter
sports.</CAT_Geography></CAT_ArtisticHeritage></CAT_SportActivities>
<CAT_ArtisticHeritage><CAT_Culture>Home tthe Council of Trent (1545 1563), it has always been a bridge between
the Italian and Central European culture. The city offers a lot tthe keen visitor from its many historical tartistic aspects:
from the underground archaeological site of the Tridentum dating back tthe Roman period, tthe Cathedral's
palaeochristian Basilica in the basement and its beautiful square, walking amongst Renaissance palazzi, churches and
the Castelldel Buonconsiglio, home for many centuries tthe Princebishops of
Trento.</CAT_Culture></CAT_ArtisticHeritage><CAT_Culture> Art and Culture </CAT_Culture>/TrentCard
<obj_webaddress>http://www.apr.trento.it/English/ENG_Trasporti.htm</obj_webaddress>
<CAT_Services>Transport</CAT_Services><CAT_Services><CAT_SportActivities>Getting around with public
transport is very easy. The urban buses link various parts of the city and the surrounding villages. A free shuttle bus
```

Figure 5.12: Input snippet of the annotated tourist board for Monte Bondone in Trento

5.2.4.1 Create Relational View

The desired tags used are 'CAT_Accommodation', 'obj_phone', 'CAT_Services', 'CAT_Culture', 'obj_money' and 'CAT_FoodAndRefreshment'. The SQL CREATE and the SQL INSERT statements produced by our software system are shown in Figure 5.13.

```

CREATE TABLE CAT_Accommodation (
    tagld VARCHAR (50),
    tag_content TEXT,
    PRIMARY KEY (tagld));
CREATE TABLE obj_phone (
    tagld VARCHAR (50),
    tag_content TEXT,
    PRIMARY KEY (tagld));
CREATE TABLE CAT_Services (
    tagld VARCHAR (50),
    tag_content TEXT,
    PRIMARY KEY (tagld));
CREATE TABLE CAT_Culture (
    tagld VARCHAR (50),
    tag_content TEXT,
    PRIMARY KEY (tagld));
CREATE TABLE obj_money (
    tagld VARCHAR (50),
    tag_content TEXT,
    PRIMARY KEY (tagld));
CREATE TABLE CAT_FoodAndRefreshment (
    tagld VARCHAR (50),
    tag_content TEXT,
    PRIMARY KEY (tagld));
CREATE TABLE table_elems (
    tableName VARCHAR (50),
    tagld VARCHAR (50),
    PRIMARY KEY (tagld));
CREATE TABLE contained_elem (
    childId VARCHAR (50),
    parentId VARCHAR (50),
    PRIMARY KEY (parentId, childId),
    FOREIGN KEY (childId) REFERENCES table_elems (tagld),
    FOREIGN KEY (parentId) REFERENCES table_elems (tagld)
);
CREATE TABLE sequence_elem (
    numInSeq VARCHAR (50),
    elemId VARCHAR (50),
    PRIMARY KEY (elemId),
    FOREIGN KEY (elemId) REFERENCES table_elems (tagld)
);

```

(a) CREATE statements for realizing the relational schema

```

INSERT INTO CAT_Accommodation (tagId, tag_content) VALUES ("CAT_Accommodation1", "Accommodation");
INSERT INTO CAT_Accommodation (tagId, tag_content) VALUES ("CAT_Accommodation2", "B & B");
INSERT INTO CAT_Accommodation (tagId, tag_content) VALUES ("CAT_Accommodation3", "Holiday apartments");
INSERT INTO CAT_Accommodation (tagId, tag_content) VALUES ("CAT_Accommodation4", "Hotels");
INSERT INTO CAT_Accommodation (tagId, tag_content) VALUES ("CAT_Accommodation5", "Rooms trent");
INSERT INTO CAT_Accommodation (tagId, tag_content) VALUES ("CAT_Accommodation6", "The TrentCard is
available at the TrentInformation Office (via Mancini 2) and on Monte Bondone (loc.Vaneze), Casteldel Buonconsiglio,
Diocesan Tridentine museum, Tridentine Museum of Natural Sciences, Gianni Caproni Aeronautics Museum,
Underground Archaeological Sass Area, Aquarium, Modern and Contemporary Art Museum of Trentand Roveret
(Palazzdelle Albere in Trentand CorsBettini in Rovereto), Museum of Uses and Customs of the Trentinpeople in San
Michele all ' Adige and at the associated accommodation facilities.");
INSERT INTO CAT_Accommodation (tagId, tag_content) VALUES ("CAT_Accommodation7", "Accommodation");
INSERT INTO CAT_Accommodation (tagId, tag_content) VALUES ("CAT_Accommodation8", "In Trentvisitors will
find a wide range of hotels of different classes and types.");
INSERT INTO CAT_Accommodation (tagId, tag_content) VALUES ("CAT_Accommodation9", "Otherwise, tourists
can stay in residences, apartments, youth hostels, agriturs or in a campsite located on the near Monte Bondone, which
offers further possibilities of lodging in hotels, residences and apartments situated at different altitudes.");
INSERT INTO CAT_Accommodation (tagId, tag_content) VALUES ("CAT_Accommodation10", "Residence of the
Princebishops from the end of the 13 th century until the secularisation of the episcopate in 1803, it was originally built
for purely defensive purposes.The military aspect of the Castle underwent huge modifications over the centuries that
transformed it intone of the most complex fortified urban residences in the Alps.It was mainly thanks tthe
Princebishops (GiorgiLichtenstein, BernardClesio, CristoforMadruzzo, FrancescAlbertiPoja) that the castle took on its
present day appearance: starting from the end of the fourteenth century until the mid seventeenth century its original
layout, the PalazzVecchi (Old castle) and the Torre d ' Augusttower had other parts added (MagnPalazzo, Giunta
Albertiana).At the same time a few of the best artists of the period arrived in Trento, whwere not allowed twork at the
castle yet have left their mark in other buildings around the city.");
INSERT INTO CAT_Accommodation (tagId, tag_content) VALUES ("CAT_Accommodation11", "The interior is a
single room, barrel vaulted, without a transept yet with a series of lateral chapels that hold marble baroque altars, a
series of altar pieces and a baroque sarcophagus containing the relics attributed tSt Clement.A series of paintings the
admired on the barrel vault depicting some moments from the Council and the protagonists of the Counterreformation
are very interesting from an historical point of view.");

```

(b) INSERT statements for populating the relational schema

Figure 5.13: SQL statements for extracted tags ‘CAT_Accommodation’, ‘obj_phone’, ‘CAT_Services’, ‘CAT_Culture’, ‘obj_money’ and ‘CAT_FoodAndRefreshment’

5.2.4.2 Example Queries

We executed the following queries on the relational view created for this case.

1. Query 1

Find all food and refreshments that include wine. This implies that we need to find all ‘CAT_FoodAndRefreshment’ tag elements that contain the string “wine”.

The SQL query and result for this are shown in Figure 5.14. This query falls under the ‘Type 4’ category described in section 5.1 because it includes string searching and matching.

Query results:
Query: SELECT CAT_FoodAndRefreshment.tagId AS CAT_FoodAndRefreshmentID, CAT_FoodAndRefreshment.tag_content AS CAT_FoodAndRefreshment FROM CAT_FoodAndRefreshment WHERE CAT_FoodAndRefreshment.tag_content LIKE '%wine%' ORDER BY CAT_FoodAndRefreshment;
Status: Executed queries: 1 Error: 0

(a) SQL query to find all ‘CAT_FoodAndRefreshment’ tag elements that contain the string “wine”

CAT_FoodAndRefreshmentID	CAT_FoodAndRefreshment
CAT_FoodAndRefreshment1	Food and Wine
CAT_FoodAndRefreshment2	Food and Wine
CAT_FoodAndRefreshment10	Food and Wine
CAT_FoodAndRefreshment37	Guided tour and tasting at the Cantine Ferrari winery in Trent
CAT_FoodAndRefreshment39	Guided tour and tasting at the farms and wineries in the Valle dei Laghi
CAT_FoodAndRefreshment4	Guided tours of the Wine Cellars with wine tasting
CAT_FoodAndRefreshment38	Refreshment with mulled wine and optional typical local dinner
CAT_FoodAndRefreshment8	The following services are also available for the TrentCard holders (some are only seasonal) : entertainment in Trent (guided tours of the Castello del Buonconsiglio with wine tasting every Saturday, guided tours of the historical centre every Saturday, visits to Villa Margone, the Eighteenth Century Garden of Villa De Mersi, the Gardens of Villa Garbari and Villa Tambosi) and on Monte Bondone (excursions with Alpine guides, visits to Botanical Alpine Garden and the Ecological Alpine Centre, rock climbing practise walls and excursions with snow shoeing accompanied by Alpine guides, visits to the mountain dairies and the honey hut, Bimbocard)
CAT_FoodAndRefreshment11	The numerous Food and Wine events that take place during the year do not only represent an opportunity to delight the palate, but also to discover traditions, tastes, aromas we believed were lost forever. Linked traditions such as the Casolara, fair of cheeses that dates back to the Middle Ages, when it was forbidden to eat meat during lent and therefore dairy products was

(b) Result of the query shown in (a)

Figure 5.14: SQL query and result to find all food and refreshments that include wine.

2. Query 2

Find the prices or costs of the services for tourist artifacts that are related to culture. This means that we need to find all the “obj_money” element tags that are in “CAT_Services” tag elements that follow a “CAT_Culture” tag.

The SQL query for this is shown in Figure 5.15 along with the results for the same. This query can be categorized as ‘Type 2’ and ‘Type 3’ as described in section 5.1 because it makes use of both the containment and sequencing relationships present in the input document.

```

Query results:
Query:
SELECT DISTINCT obj_money.tagId AS obj_moneyID, obj_money.tag_content AS obj_money,
CAT_Services.tag_content AS CAT_Services
FROM obj_money, CAT_Services, CAT_Culture, contained_elem, sequence_elem AS seq1,
sequence_elem AS seq2
WHERE ((contained_elem.childId = obj_money.tagId) &&
(contained_elem.parentId = CAT_Services.tagId)) &&
((seq1.elemId = CAT_Services.tagId) &&
(seq2.elemId = CAT_Culture.tagId) &&
(seq1.numInSeq > seq2.numInSeq))
ORDER BY obj_moneyID;
Status: Executed queries: 1 Error: 0

```

(a) SQL query to find all the “obj_money” element tags that are in “CAT_Services” tag elements that follow a “CAT_Culture” tag

obj_moneyID	obj_money	CAT_Services
obj_money1	9 Euros	The TrentCard is a prepaid card and twversions are proposed: valid for 24 hours (9 Euros) and 48 hours (14 Euros).It is complemented by an inkit tgive tourists an indication of the area and the services.
obj_money2	14 Euros	The TrentCard is a prepaid card and twversions are proposed: valid for 24 hours (9 Euros) and 48 hours (14 Euros).It is complemented by an inkit tgive tourists an indication of the area and the services.

(b) Result of query shown in (a)

Figure 5.15: SQL query and result to find the prices or costs for tourist artifacts that are related to culture

5.3 Standard XML Data Samples

We also consider a standard collection of XML files that contain plays by William Shakespeare marked up in XML [46]. We specifically processed the documents for three of the plays, namely, Hamlet, Julius Caesar and Macbeth. Figure 5.16 shows part of the XML file for Julius Caesar.

```

<?XML version="1.0"?><!DOCTYPE play PUBLIC "-//Free Text Project/DTD Play//EN"><PLAY><TITLE>The
Tragedy of Julius Caesar</TITLE></fm><p>Text placed in the public domain by Moby Lexical Tools, 1992.</p>
<p>SGML markup by Jon Bosak, 1992-1994.</p><p>XML version by Jon Bosak, 1996-1997.</p><p>This work may
be freely copied and distributed worldwide.</p></fm><PERSONAE><TITLE>Dramatis Personae</TITLE>
<PERSONA>JULIUS CAESAR</PERSONA><PGROUP><PERSONA>OCTAVIUS CAESAR</PERSONA>
<PERSONA>MARCUS ANTONIUS</PERSONA><PERSONA>M. AEMILIUS LEPIDUS</PERSONA>
<GRPDESCR>triumvirs after death of Julius Caesar.</GRPDESCR></PGROUP><PGROUP>
<PERSONA>CICERO</PERSONA><PERSONA>PUBLIUS</PERSONA><PERSONA>POPILIUS
LENA</PERSONA><GRPDESCR>senators.</GRPDESCR></PGROUP><PGROUP><PERSONA>MARCUS
BRUTUS</PERSONA><PERSONA>CASSIUS</PERSONA><PERSONA>CASCA</PERSONA>
<PERSONA>TREBONIUS</PERSONA><PERSONA>LIGARIUS</PERSONA><PERSONA>DECIUS
BRUTUS</PERSONA><PERSONA>METELLUS CIMBER</PERSONA><PERSONA>CINNA</PERSONA>
<GRPDESCR>conspirators against Julius Caesar.</GRPDESCR></PGROUP><PGROUP>
<PERSONA>FLAVIUS</PERSONA><PERSONA>MARULLUS</PERSONA>
<GRPDESCR>tribunes.</GRPDESCR></PGROUP><PERSONA>ARTEMIDORUS Of Cnidos, a teacher of rhetoric.
</PERSONA><PERSONA>A Soothsayer</PERSONA><PERSONA>CINNA, a poet. </PERSONA>
<PERSONA>Another Poet</PERSONA><PGROUP><PERSONA>LUCILIUS</PERSONA>
<PERSONA>TITINIUS</PERSONA><PERSONA>MESSALA</PERSONA><PERSONA>Young
CATO</PERSONA><PERSONA>VOLUMNIUS</PERSONA><GRPDESCR>friends to Brutus and
Cassius.</GRPDESCR></PGROUP><PGROUP><PERSONA>VARRO</PERSONA>
<PERSONA>CLITUS</PERSONA><PERSONA>CLAUDIUS</PERSONA><PERSONA>STRATO</PERSONA>
<PERSONA>LUCIUS</PERSONA><PERSONA>DARDANIUS</PERSONA><GRPDESCR>servants to
Brutus.</GRPDESCR></PGROUP><PERSONA>PINDARUS, servant to Cassius.</PERSONA>
<PERSONA>CALPURNIA, wife to Caesar.</PERSONA><PERSONA>PORTIA, wife to Brutus.</PERSONA>
<PERSONA>Senators, Citizens, Guards, Attendants, &amp;c.</PERSONA></PERSONAE><SCNDESCR>SCENE
Rome: the neighbourhood of Sardis: the neighbourhood of Philippi.</SCNDESCR><PLAYSUBT>JULIUS
CAESAR</PLAYSUBT><ACT><TITLE>ACT I</TITLE><SCENE><TITLE>SCENE I. Rome. A street.</TITLE>
<STAGEDIR>Enter FLAVIUS, MARULLUS, and certain Commoners</STAGEDIR><SPEECH>
<SPEAKER>FLAVIUS</SPEAKER><LINE>Hence! home, you idle creatures get you home:</LINE><LINE>Is this a
holiday? what! know you not,</LINE><LINE>Being mechanical, you ought not walk</LINE><LINE>Upon a labouring
day without the sign</LINE><LINE>Of your profession? Speak, what trade art thou?</LINE></SPEECH><SPEECH>

```

Figure 5.16: Part of the XML document for a Shakespeare play - Julius Caesar

5.3.1 Create Relational View

We created a view made up of the following tag elements – ‘PERSONA’, ‘SPEECH’, ‘SCENE’, ‘LINE’ and ‘SPEAKER’ for each of the plays. The resulting output for the input shown in Figure 5.16 is shown in Figure 5.17.

```

CREATE TABLE PERSONA (
    tagId VARCHAR (50),
    tag_content TEXT,
    PRIMARY KEY (tagId));
CREATE TABLE SPEECH (
    tagId VARCHAR (50),
    tag_content TEXT,
    PRIMARY KEY (tagId));
CREATE TABLE SCENE (
    tagId VARCHAR (50),
    tag_content TEXT,
    PRIMARY KEY (tagId));
CREATE TABLE LINE (
    tagId VARCHAR (50),
    tag_content TEXT,
    PRIMARY KEY (tagId));
CREATE TABLE SPEAKER (
    tagId VARCHAR (50),
    tag_content TEXT,
    PRIMARY KEY (tagId));
CREATE TABLE table_elems (
    tableName VARCHAR (50),
    tagId VARCHAR (50),
    PRIMARY KEY (tagId));
CREATE TABLE contained_elem (
    childId VARCHAR (50),
    parentId VARCHAR (50),
    PRIMARY KEY (parentId, childId),
    FOREIGN KEY (childId) REFERENCES table_elems (tagId),
    FOREIGN KEY (parentId) REFERENCES table_elems (tagId)
);
CREATE TABLE sequence_elem (
    numInSeq VARCHAR (50),
    elemId VARCHAR (50),
    PRIMARY KEY (elemId),
    FOREIGN KEY (elemId) REFERENCES table_elems (tagId)
);

```

(a) CREATE statements for realizing the relational schema

```

INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA1", "JULIUS CAESAR");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA2", "OCTAVIUS CAESAR");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA3", "MARCUS ANTONIUS");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA4", "M.AEMILIUS LEPIDUS");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA5", "CICERO");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA6", "PUBLIUS");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA7", "POPILIUS LENA");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA8", "MARCUS BRUTUS");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA9", "CASSIUS");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA10", "CASCA");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA11", "TREBONIUS");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA12", "LIGARIUS");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA13", "DECIUS BRUTUS");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA14", "METELLUS CIMBER");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA15", "CINNA");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA16", "FLAVIUS");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA17", "MARULLUS");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA18", "ARTEMIDORUS Of Cnidos, a teacher of
rhetoric.");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA19", "A Soothsayer");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA20", "CINNA, a poet.");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA21", "Another Poet");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA22", "LUCILIUS");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA23", "TITINIUS");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA24", "MESSALA");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA25", "Young CATO");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA26", "VOLUMNIUS");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA27", "VARRO");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA28", "CLITUS");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA29", "CLAUDIUS");
INSERT INTO PERSONA (tagId, tag_content) VALUES ("PERSONA30", "STRATO");

```

(b) INSERT statements for populating the relational schema

Figure 5.17: SQL statements for extracted tags ‘PERSONA’, ‘SPEECH’, ‘SCENE’, ‘LINE’ and ‘SPEAKER’

5.3.2 Example Queries

We executed the following queries on the resulting relational view.

1. Query 1

Find all the speakers and his or her lines in a speech for the play titled ‘Hamlet’. This means that we need to find all “LINE” element tags that follow a “SPEAKER” tag element and all these tag elements are in the same “SPEECH” tag element.

We extract the required tags from the XML document related to the play titled “Hamlet”, and run the resulting SQL statements on the database server. This creates the relational schema and populates the database with relevant information. The SQL query for this query is shown in

Figure 5.18 along with the results of the query. This query falls under categories ‘Type 2’ and ‘Type 3’ as described in section 5.1.

```

Query results:
Query:
SELECT speech.tagId AS SpeechID, speaker.tag_content AS Speaker, line.tag_content AS Line
FROM line, speaker, speech, contained_elem AS cont1, contained_elem AS cont2,
sequence_elem AS seq1
WHERE
((cont1.childId = line.tagId) && (cont1.parentId = speech.tagId)) &&
((cont2.childId = speaker.tagId) && (cont2.parentId = cont1.parentId)) &&
((seq1.elemId = line.tagId) && (seq1.numInSeq > (
SELECT numInSeq
FROM sequence_elem AS seq2
WHERE seq2.elemId = speaker.tagId)
))
ORDER BY speech.tagId;
Status: Executed queries: 1 Error: 0

```

(a) SQL query to find all the “LINE” element tags that are in “SPEECH” tag elements that follow a “SPEAKER” tag in the same “SPEECH” tag

SpeechID	Speaker	Line
SPEECH1	BERNARDO	Who ' s there ?
SPEECH10	FRANCISCO	Not a mouse stirring.
SPEECH100	HORATIO	Season your admiration for awhile
SPEECH100	HORATIO	With an attent ear, till I may deliver,
SPEECH100	HORATIO	Upon the witness of these gentlemen,
SPEECH100	HORATIO	This marvel to you.
SPEECH1000	HAMLET	Being thus be-netted round with villainies, - -
SPEECH1000	HAMLET	Ere I could make a prologue to my brains,
SPEECH1000	HAMLET	They had begun the play--I sat me down,
SPEECH1000	HAMLET	Devised a new commission, wrote it fair:
SPEECH1000	HAMLET	I once did hold it, as our statists do,
SPEECH1000	HAMLET	A baseness to write fair and labour ' d much
SPEECH1000	HAMLET	How to forget that learning, but, sir, now
SPEECH1000	HAMLET	It did me yeoman ' s service: wilt thou know
SPEECH1000	HAMLET	The effect of what I wrote ?
SPEECH1001	HORATIO	Ay, good my lord.
SPEECH1002	HAMLET	An earnest conjuration from the king,
SPEECH1002	HAMLET	As England was his faithful tributary,
SPEECH1002	HAMLET	As love between them like the palm might flourish,
SPEECH1002	HAMLET	As peace should stiff her wheaten garland wear
SPEECH1002	HAMLET	And stand a comma ' tween their amities,
SPEECH1002	HAMLET	And many such-like ' As ' es of great charge,
SPEECH1002	HAMLET	That on the view and knowing of these contents

(b) Result of query shown in (a)

Figure 5.18: SQL query and result to find all lines of a speaker in a speech

2. Query 2

Find all the lines in speeches by Brutus in the play titled 'Julius Caesar'. This means that from the XML document for the play titled 'Julius Caesar', find all 'LINE' element tags that are in a 'SPEECH' element and contain the 'SPEAKER' tag containing the string "BRUTUS".

After extracting the required tags, creating the relational schema and populating the database with relevant data, the SQL query shown in Figure 5.19 (a) is executed. The results of the SQL query are shown in Figure 5.19 (b). This query falls under categories 'Type 2' and 'Type 4' because it uses the containment relationship between the XML tag elements and string searching.

Query results:
<pre>Query: SELECT speech.tagId AS SpeechID, speaker.tag_content AS Speaker, line.tag_content AS Line FROM line, speaker, speech, contained_elem AS cont1, contained_elem AS cont2 WHERE ((cont1.childId = line.tagId) && (cont1.parentId = speech.tagId)) && ((cont2.childId = speaker.tagId) && (cont2.parentId = cont1.parentId)) && (speaker.tag_content LIKE '%brutus%') ORDER BY speech.tagId; Status: Executed queries: 1 Error: 0</pre>

(a) SQL query to find all "LINE" element tags that are in a "SPEECH" element and contain the 'SPEAKER' tag containing the string "BRUTUS"

SpeechID	Speaker	Line
SPEECH102	BRUTUS	What a blunt fellow is this grown to be !
SPEECH102	BRUTUS	He was quick mettle when he went to school.
SPEECH104	BRUTUS	And so it is.For this time I will leave you:
SPEECH104	BRUTUS	To-morrow, if you please to speak with me,
SPEECH104	BRUTUS	I will come home to you; or, if you will,
SPEECH104	BRUTUS	Come home to me, and I will wait for you.
SPEECH143	BRUTUS	What, Lucius, ho !
SPEECH143	BRUTUS	I cannot, by the progress of the stars,
SPEECH143	BRUTUS	Give guess how near to day.Lucius, I say !
SPEECH143	BRUTUS	I would it were my fault to sleep so soundly.
SPEECH143	BRUTUS	When, Lucius, when ? awake, I say ! what, Lucius !
SPEECH145	BRUTUS	Get me a taper in my study, Lucius:
SPEECH145	BRUTUS	When it is lighted, come and call me here.
SPEECH147	BRUTUS	It must be by his death: and for my part,
SPEECH147	BRUTUS	I know no personal cause to spurn at him,
SPEECH147	BRUTUS	But for the general.He would be crown ' d:
SPEECH147	BRUTUS	How that might change his nature, there ' s the question.
SPEECH147	BRUTUS	It is the bright day that brings forth the adder;
SPEECH147	BRUTUS	And that craves wary walking.Crown him ? - - that; - -
SPEECH147	BRUTUS	And then, I grant, we put a sting in him,
SPEECH147	BRUTUS	That at his will he may do danger with.
SPEECH147	BRUTUS	The abuse of greatness is, when it disjoins
SPEECH147	BRUTUS	Remorse from power: and, to speak truth of Caesar,
SPEECH147	BRUTUS	It doth become the可怕

(b) Result of the query shown in (a)

Figure 5.19: SQL query and result to find all the lines in speeches by Brutus

5.4 Basic Statistical Information About The Evaluation

To better understand the evaluation of our system, we have included the following statistical information about our test input and results. Table 1 shows the details of the document sets that we used in our test input. It shows the number of document files in each set, their total size and the total number of tags in all the documents of each set. It also shows the set of all possible tags that appear in the document set.

Table 1: Information About The Document Sets Used In Our Evaluation

Document Set	No. Of Files	Total Size (KB)	Total No. Of Tags	Set Of Possible Tags	Desired Set Of Tags	No. Of Tables Extracted
Ads	3	118	1684	ad, ads_doc, contact, facility, location, price, term, type	ad, location, contact, facility, price, type	9
Biblio	59	587	4104	Biblio_abs_contribution, Biblio_Abstract, Biblio_Authors, Biblio_cit_title, Biblio_cit_year, Biblio_Citation, Biblio_con_contribution, Biblio_con_fwork, Biblio_Conclusion, Biblio_Doc, Biblio_Head, Biblio_intro_contribution, Biblio_Introduction, Biblio_Title	Biblio_Abstract, Biblio_Citation, Biblio_cit_title, Biblio_cit_year, Biblio_Email	8
HIPPA	2	316	5788	DOC, Actor, AntiObligation, AntiRight, Constraint, Continue_AntiObligation, Continue_Obligation, Continue_Right, CrossRef, Date, Event, Exception, Index, Information, Num_Section, Obligation, Policy, Right, Section, Sentence, TempCondition, Title_section	Event, Information, Actor, Obligation, Continue_Obligation	8
TouristBoards	13	1565	13143	AccommodationType, Altitude, Availability, CableRoad, CarHire, Castle, CAT_Accommodation, CAT_ArtisticHeritage, CAT_Culture, CAT_FoodAndRefreshment, CAT_Geography, CAT_LocalProducts, CAT_Services, CAT_SportActivities, CAT_Wellness, Church, Cinema, Climate, Costume, Court, Cuisine, CulturalEvent, CulturalPath, Degustation, GastronomicEvent, Lake, Library, Market, Mountain, Museum, obj_date, obj_discount, obj_distance, obj_email, obj_money, obj_people, obj_phone, obj_temperature, obj_time, obj_webaddress, Park, Products, ProfLevel, Rental, Restaurant, Ruins, School, SportCenter, SportEvent, Stadium, Station, SwimmingPool, Ticket, Tourist_Doc, TouristOffice, Track, Tradition, Transport, Valley, WellnessCenter, WellnessFacility, WellnessService	CAT_Accommodation, obj_phone, CAT_Services, CAT_Culture, obj_money, CAT_FoodAndRefreshment	9
Shakespeare's Plays	3	614	15046	act, epilogue, fm, grpdescr, induct, line, p, persona, personae, pgroup, play, playsubt, prologue, scene, scnddescr, speaker, speech, stagedir, subhead, subtitle, title	persona, speech, scene, line, speaker	8

Table 2 shows information about the specific files that were used in the examples shown in this chapter. It shows data related to the resulting relational schema, such as the row count for each table created in the derived relational view, the contained relationships inferred and the sequence relationships inferred.

Table 2: Summary Of The Input Files Used As Examples In This Chapter

Category	File Name: newroma1.xml		TouristBoards	File Name: trentomontebondone.xml	
	Tag Count: 1375			Tag Count: 1620	
	Desired Tag	Row Count		Desired Tag	Row Count
Ads	ad	175	CAT_Accommodation	91	
	contact	199	CAT_Culture	170	
	facility	369	CAT_FoodAndRefreshment	47	
	location	152	CAT_Services	112	
	price	214	obj_money	17	
	type	196	obj_phone	233	
	contained_elem	1374	contained_elem	1619	
	sequence_elem	1374	sequence_elem	1619	
	Biblio	File Name: 178.xml		File Name: hamlet.xml	
Tag Count: 448		Tag Count: 6636			
Desired Tag		Row Count	Desired Tag	Row Count	
Biblio_Abstract		1	line	4014	
Biblio_cit_title		110	persona	26	
Biblio_cit_year		111	scene	20	
Biblio_Citation		219	speaker	1150	
Biblio_Email		1	speech	1138	
contained_elem		447	contained_elem	6635	
sequence_elem	447	sequence_elem	6635		
HIPPA	File Name: h_164.xml		File Name: jcaesar.xml		
	Tag Count: 5089		Tag Count: 4455		
	Desired Tag	Row Count	Desired Tag	Row Count	
	Actor	1371	line	2596	
	Continue_Obligation	55	persona	36	
	Event	578	scene	18	
	Information	368	speaker	798	
	Obligation	124	speech	795	
	contained_elem	5088	contained_elem	4454	
sequence_elem	5088	sequence_elem	4454		

5.5 Summary

In this chapter we present the results obtained from our transformation software system on various input documents. We also show different kinds of queries that can be made on the relational schemas and the retrieved data. The various input files used were from different sources. Table 3 summarizes the input files used in our experiments and the different types of queries that were posed on them.

Table 3: Summary Of The Various Types Of Queries Made On Different Input Documents

	Query - Type 1 Queries that illustrate information gathered from the XML document	Query - Type 2 Queries based on the tree-structured nature of the XML document	Query - Type 3 Queries based on the document order of the XML document	Query - Type4 Queries that search text in XML document elements
Ads Section 5.2.1.1.2	√			√
Biblio Section 5.2.1.2.2		√		
HIPPA Section 5.2.1.3.2		√	√	√
Tourist Boards Section 5.2.1.4.2		√	√	√
Shakespeare's Plays Section 5.2.2		√	√	√

In Chapter 6, we will give a summary of the conclusions that were reached from these experiments. We also talk about the limitations of the implementation and the future enhancements that can be made.

Chapter 6

Conclusion

In this chapter we sum up the thesis and the research work we have undertaken. We summarize the results from our experiments in Section 6.1. Section 6.2 talks about some of the limitations of our implementation and the future work that can be undertaken with respect to the limitations. Finally Section 6.3 concludes the entire thesis.

6.1 Summary Of Experiments

The primary goal of this research was to take an XML document and a given set of desired tag elements and transform these to SQL statements using source transformation. The SQL statements when executed on a database server will realize a relational view over the input XML document. We conducted an evaluation of our software system. This is explained and shown in Chapter 5. The software system we used in these experiments is implemented using TXL, a source transformation tool. The transformation process was explained in Chapter 4. Thus, from our experiments we can conclude that it is reasonable to use source transformation techniques to parse the text in XML documents and obtain a relational view over them. The relational view obtained is constrained by the search tags or desired tags specified in the input.

The secondary goal of our research was to observe the different kinds of queries that can be posed on the relational schema formulated from the input XML document. For each input shown in Chapter 5, we have also shown examples of queries that can be posed on the extracted relational data. The relational data is queried efficiently using SQL. The querying is focussed because the

relational view obtained over the XML data is limited by the XML tag elements specified by the external user or application.

Therefore our experiments, although limited in nature, have shown that our approach to obtaining a relational view on XML documents and querying the XML data using SQL is definitely feasible.

6.2 Limitations And Future Work

We restricted our goals in this research project to showing that an automated source transformation can be performed to acquire a relational view over an XML document. We were able to demonstrate the manner in which this can be done with some elementary examples and queries shown in Chapter 5. However, our implementation is not foolproof and so in this section we list and explain some of the limitations of our work. In some cases, we also provide possible solutions, which can be further explored in the future.

The relational schema that is derived using our approach is fairly basic and fixed. Every extracted tag is transformed into a relation in the schema with each attribute of the tag being transformed becomes a column in the relation. This has two main consequences. Firstly, the number of relations created is directly proportional to the number of extracted tags. This number could become very large and this leads to the second consequence. As the number of relations in a schema increases, the number of joins in the SQL queries over that schema tends to increase too. This results in a decrease in the efficiency of the queries, that is, the queries can take a long time

to execute to completion. A possible solution to this is to extend one of the inlining techniques suggested by J. Shanmugasundaram et al [34] to our approach. Another possible solution would be to use the Document Type Definition (DTD) for the XML documents that have a DTD specified. This DTD could be used to enhance and optimize the extracted relational schema. Thus, with modification of the relational schema, it is possible to improve the efficiency of the SQL query.

The second important limitation of our work is related to the search tags specified. If the tag being searched is a keyword in SQL then this causes an SQL error when the SQL statements are executed on the database server. A possible solution to this is to create a list of SQL keywords and process the SQL statements generated to check if any searched tag name is a keyword. If such a tag name is found, then that tag name can be replaced by a modified version of itself. For example, if the search tag is 'Right' which is a SQL keyword then once the SQL statements are generated we can replace all 'Right' strings with the string '_Right'.

The third important limitation is also related to the search tags. This deals with the case in which the searched tag is not present in the input XML document. If any of the desired tags are not found in the input XML document, then no table is created for that tag in the extracted relational schema. The table cannot be created because the tag being searched for is non-existent. If you would like to have a table created for this tag, which is empty, then you will have to create an empty intermediate file for that missing tag in Stage 3 of the transformation process. Stages 4 and 5 will have to be modified to include an empty file as input.

Our implementation is currently limited by the size of the XML document. Larger documents require longer time to be parsed and transformed, and therefore take a longer time to produce the result using our approach. To overcome this limitation, the input document could be split into chunks and then be processed one chunk at a time using our approach. To study the scalability of our approach, there is a need for experiments with larger documents. This remains as future work.

With respect to the Semantic Web, a possible future extension to our work would involve automating the specification of search tags from a search query in a search engine. The search query can be used to formulate concepts and these concepts can be used to identify the desired tags.

The last important aspect we would like to touch upon is a constructive addition that can be made to the current implementation. It would be very useful to have a way to associate document name and line number in the document to the extracted tag elements in the relational schema. A possible way this can be done is to include the filename and line number as attributes in every XML tag element in the document in Stage 1 of our transformation process. Another way would be to include these two values in the relation that stores all the table elements, that is, the “table_elems” relation explained in Section 4.3.4 of Chapter 4.

6.3 Thesis Conclusion

In this thesis, using TXL as our source transformation tool, we have shown that it is feasible to use source transformation techniques to obtain a relational view on XML documents. This

technique allows us to parse the input document and the desired tags and allows us to obtain an intermediate representation of the database schema and data. The technique further allows us to parse this intermediate representation and transform it into SQL statements. We have used our transformation technique on numerous example documents. This approach can be useful for querying large XML documents, for example legal documents, which are usually queried for the same information numerous times.

Our approach differs from the XML query language - XQuery. The most important advantage of our approach is that it leverages the querying functionality of the well-established relational database systems. Therefore a separate processor like an XQuery processor is not required. Our approach facilitates restricting querying to only those parts of the input document that are interesting to the problem at hand. XQuery does not directly provide this facility. XQuery is an advanced query language and has many operators and functions to support querying. However, one of its drawbacks is that the results returned from the queries are always in XML format. In our approach, results returned are in relational data format and so they can be used in any manner suitable for the application. An important aspect of both approaches is that it is beneficial to have some idea of the XML document's schema. Since XQuery uses path expressions in querying, it is required to know the exact schema of the entire XML document to be able to formulate the queries. In our approach, just the knowledge of the element tags present in the XML document is enough for querying the data in the documents. Since we extract the implicit relationships among the tag elements as a relation, it is possible to infer the schema of the XML document from the derived relational view.

In conclusion we do believe that source transformation techniques can be used to analyse XML documents and to obtain a relational view over them. The XML input documents we used were varied in nature and we were able to obtain corresponding relational views. We admit that our research work was undertaken with limited resources, and does have the limitations mentioned in Section 6.2. However, with the success achieved in this implementation, it is clear that there is scope for more research in this area in order to realize the full potential of using source transformation techniques to query XML data.

References

- [1] W3C: *About The World Wide Web*, 1992, <http://www.w3.org/WWW/> (Fetched on June 7, 2007)
- [2] T. Berners-Lee, J. Hendler, O. Lassila: *The Semantic Web*. Scientific American, pages 34–43, May 2001
- [3] N. Kiyavitskaya, N. Zeni, J. Cordy, L. Mich, J. Mylopoulos: *Applying Software Analysis Technology to Lightweight Semantic Markup of Document Text*. Proceedings of the 3rd International Conference on Advances in Pattern Recognition (ICAPR), pages 590-600, Bath, U.K., August 2005
- [4] World Wide Web Consortium (W3C) Recommendation: *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. Aug 2006, <http://www.w3.org/TR/REC-xml/>
- [5] D. Chamberlin, R. Boyce: *SEQUEL: A structured English query language*. International Conference on Management of Data, Proceedings Of the 1974 ACM SIGFIDET (now SIGMOD) Workshop on Data Description, Access and Control, pages 249–264. Ann Arbor, Michigan, USA, 1974
- [6] E. Codd: *A Relational Model of Data for Large Shared Data Banks*. Communications of the ACM, Volume 13, No. 6, pages 377-387, June 1970
- [7] T. Dean, J. Cordy, K. Schneider, A. Malton: *Experience using design recovery techniques to transform legacy systems*. Proceedings Of 17th International Conference on Software Maintenance (ICSM), pages 622–631, 2001
- [8] J. Cordy, T. Dean, A. Malton, K. Schneider: *Source transformation in software engineering using the TXL transformation system*. Journal of Information and Software Technology 44 (13), pages 827–837, 2002
- [9] World Wide Web Consortium (W3C) Recommendation: *XML Path Language (XPath) Version 1.0*. Nov 1999, <http://www.w3.org/TR/xpath> (Fetched on June 7, 2007)
- [10] World Wide Web Consortium (W3C) Candidate Recommendation: *XQuery 1.0: An XML Query Language Version 1.0*. Nov 2005, <http://www.w3.org/TR/xquery/> (Fetched on June 7, 2007)
- [11] J. Cordy: *TXL – a language for programming language tools and applications*. Proceedings Of 4th International. Workshop on Language Descriptions, Tools and

- Applications (LDTA), *Electronic Notes in Theoretical Computer Science* 110, pages 3–31, 2004
- [12] World Wide Web Consortium (W3C): *Tim Berners-Lee*.
<http://www.w3.org/People/Berners-Lee/> (Fetched on June 7, 2007)
- [13] E. Dumbill: *Building the Semantic Web*. XML.com Article (March 07, 2001), O'Reilly Media Inc, <http://www.xml.com/pub/a/2001/03/07/buildingsw.html> (Fetched on June 7, 2007)
- [14] E. Dumbill: *Berners-Lee and the Semantic Web Vision*. XML.com Article (December 06, 2000), O'Reilly Media Inc, <http://www.xml.com/pub/a/2000/12/xml2000/timbl.html> (Fetched on June 7, 2007)
- [15] T. Berners-Lee: *WWW past & future - Berners Lee - Royal Society*. World Wide Consortium Talks September, 22, 2003, <http://www.w3.org/2003/Talks/0922-rsoc-tbl/slide30-0.html> (Fetched on Jul 9th 2007)
- [16] World Wide Web Consortium (W3C) Recommendation: *RDF Primer*. Feb 2004, <http://www.w3.org/TR/rdf-primer/> (Fetched on June 7, 2007)
- [17] V. R. Benjamins, J. Contreras, O. Corcho and A. Gómez-Pérez: *Six Challenges for the Semantic Web*.
http://www.cs.man.ac.uk/~ocorcho/documents/KRR2002WS_BenjaminsEtAl.pdf (Fetched on June 7, 2007)
- [18] A. Chebotko: *Challenges for Information Systems Towards The Semantic Web*.
<http://www.sigsemis.org/?p=9> (Fetched on June 7, 2007)
- [19] E. Prud'hommeaux: *Contrasting Logic Over XPath and RDF*. May 2003, <http://www.w3.org/2002/02/11-XPath-vs-RDF> (Fetched on June 7, 2007)
- [20] L. Argerich, C. Lea, K. Egervari, M. Anton, C. Killian, C. Hubbard, J. Fuller: *Professional PHP4XML*. ISBN: 1861007213, Wrox Press Ltd.
- [21] D. Stodder, B. Dupont: *XQuery Sets Stage for Changes in Database Management*. Network Computing Reports (2007),
http://www.networkcomputingreports.com/issues_archive.jhtml (Fetched on June 7, 2007)
<http://www.networkcomputing.com/showArticle.jhtml?articleID=196900616> (Fetched on June 7, 2007)
- [22] P. Weinberg and J. Groff: *SQL – The Complete Reference*. ISBN: 9780072225594 McGraw-Hill Osborne Media, Aug 2002.

- [23] A. Malton, K. Schneider, J. Cordy, T. Dean, D. Cousineau, J. Reynolds: *Processing Software Source Text in Automated Design Recovery and Transformation*. Proceedings of the 9th IEEE International Workshop on Program Comprehension (IWPC), pages 127-134, Toronto, Canada, May 2001
- [24] S. Zhang, T. Dean, S. Knight: *A Lightweight Approach To State Based Security Testing*. Proceedings of CASCON, The 16th IBM Centre for Advanced Studies International Conference on Computer Science and Software Engineering, Article No. 28, pages 341-344, Toronto, Canada, October 2006
- [25] C. Dahn, S. Mancoridis: *Using Program Transformation to Secure C Programs Against Buffer Overflows*. Proceedings Of the 10th IEEE International Working Conference on Reverse Engineering (WCRE), pages 323-333, Victoria, Canada, November 2003
- [26] S. Xu,, T. Dean: *Modernizing JavaServer Pages by Transformation*. Proceedings of the 7th International Symposium on Web Site Evolution, Volume 00, pages 111-118, Budapest, Hungary, September 2005
- [27] A. Hassan, R. Holt: *A Lightweight Approach for Migrating Web Frameworks*. Journal of Information and Software Technology, Volume 47, No. 8, pages 521-532, June 2005
- [28] X. Guo, J. Cordy, T. Dean: *Unique Renaming of Java Using Source Transformation*. Proceedings of the 3rd IEEE International Workshop on Source Code Analysis and Manipulation (SCAM), pages 151-160, Amsterdam, September 2003
- [29] A. Halverson, V. Josifovski, G. Lohman, H. Pirahesh, M. Morschel: *ROX: Relational Over XML*. Proceedings of the 30th Very Large Data Base (VLDB) Conference, pages 264-275, Toronto, Canada, 2004.
- [30] J. Shanmugasundaram, J. Kiernan, E. Shekita, C. Fan, J. Funderburk: *Querying XML Views of Relational Data*. Proceedings of the 27th Very Large Data Base (VLDB) Conference, pages 261-270, Roma, Italy, 2001.
- [31] M. Fernandez, A. Morishima, D. Suciu: *Efficient Evaluation of XML Middle-ware Queries*. Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, pages 103-114, Santa Barbara, CA, USA, 2001.
- [32] I. Manolescu, D. Florescu, D. Kossmann: *Answering XML Queries over Heterogeneous Data Sources*. Proceedings of the 27th Very Large Data Base (VLDB) Conference, pages 241-250, Roma, Italy, 2001.

- [33] A. Deutsch, V. Tannen: *MARS: A System for Publishing XML from Mixed and Redundant Storage*. Proceedings of the 29th Very Large Data Base (VLDB) Conference, pages 201-212, Berlin, Germany, 2003.
- [34] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. DeWitt, J. Naughton: *Relational Databases for Querying XML Documents: Limitations and Opportunities*. In Proceedings Of 25th Very Large Data Base (VLDB) Conference, pages 302-314, Edinburgh, Scotland, 1999.
- [35] S. Amer-Yahia, F. Du, J. Freire: *A Comprehensive Solution to the XML-to-Relational Mapping Problem*. Proceedings of the 6th Annual ACM International Workshop on Web Information and Data Management (WIDM), pages 31-38 (Session: XML Processing), Washington DC, USA, 2004.
- [36] D. Florescu, D. Kossman: *Storing and Querying XML Data using an RDBMS*. IEEE Data Engineering Bulletin, Volume 22, No. 3, pages 27-34, 1999.
- [37] A. Deutsch, M. Fernandez, D. Suci: *Storing Semistructured Data with STORED*. Proceedings Of ACM SIGMOD International Conference on Management of Data, pages 431-442, 1999.
- [38] I. Tatarinov, S. Vigiass, E. Kevin S. Beyer, J. Shanmugasundaram, C. Zhang: *Storing and Querying Ordered XML Using a Relational Database System*. In Proceedings Of SIGMOD Conference, pages 204-215, 2002.
- [39] M. Yoshikawa, T. Amagasa, T. Shimura, S. Uemura: *XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases*. ACM Transactions on Internet Technology (TOIT), Volume 1, pages 110-141, 2001.
- [40] D. DeHaan, D. Toman, M. Consens, M. Ozsu: *A Comprehensive XQuery to SQL Translation using Dynamic Interval Encoding*. Proceedings Of the ACM SIGMOD International Conference on Management of Data, pages 623-634, San Diego, CA, USA, 2003.
- [41] K. Beyer, R. Cochrane, M. Hvizdos, V. Josifovski, J. Kleweein, G. Lapis, G. Lohman, R. Lyle, M. Nicola, F. Özcan, H. Pirahesh, N. Seemann, A. Singh, T. Truong, R.C. Van der Linden, B. Vickery, C. Zhang, G. Zhang: *DB2 goes hybrid: Integrating native XML and XQuery with relational data and SQL*. IBM Systems Journal, Volume 45, No 2, 2006.
- [42] *Oracle Database 10g Release 2 XML DB Technical Overview*. An Oracle White Paper, May 2005.

- [43] S. Pal, M. Fussell, I. Dolobowsky: *XML Support in Microsoft SQL Server 2005*. Microsoft Corporation, Microsoft SQL Server 9.0 Technical Articles, December 2005.
- [44] T. Dean, J. Cordy, A. Malton, K. Schneider: *Agile parsing in TXL*. Journal of Automated Software Engineering 10(4), pages 311-336, 2003
- [45] World Wide Web Consortium (W3C) Working Group: *XML Query Use Cases*. March 2007, <http://www.w3.org/TR/xquery-use-cases/> (Fetched on Sep 25th 2007)
- [46] J. Bosak: *Shakespeare 2.00*, <http://www.xml.com/pub/r/396> (Fetched on Aug 2nd 2007)

Appendix A

TXL Grammars

XML Grammar

```
% TXL Grammar for XML Documents
% Shruti Atre
% Queen's University, August 2007
% This file contains the grammar for elements and attributes of XML element tags

% The default in TXL is to allow tokens to extend over multiple lines. The following 'pragma'
limits tokens to a single line.
#pragma -nomultiline

tokens
    id      + "[\a_\:][\a\d.-_\:]*"      % From Definition [5] in XML 1.0 Recommendation
    charlit ""
end tokens

comments      % From Definition [15] in XML 1.0 Recommendation
    <!-- -->
    <? ?>
    <! >
end comments

define program      % The non-terminal that describes the whole input to the programs
    [xmldoc]
end define

define xmldoc
    [opt element]      % Element is optional to accommodate blank,
empty documents
end define

% An XML element could be tag with some content an empty tag
define element      % From Definition [39] in XML 1.0
    Recommendation
    [empty_elem_tag]
    | [tag_content]
end define

define tag_content      % XML tag with a start tag, some content and an
end tag
    [NL]
    [stag]      [IN]
    [content]      [NL][EX]
    [etag]
end define
```

```

define empty_elem_tag          % From Definition [44] in XML 1.0 Recommendation
    '< [SPOFF] [id] [repeat attribute] !/> [SPON]
end define

define stag                    % From Definition [40] in XML 1.0
Recommendation
    '< [SPOFF] [id] [repeat attribute] !> [SPON]
end define

% The content for the XML tag element
define content                 % From Definition [43] XML 1.0 Recommendation
    [repeat sub_content]
end define

define sub_content            % Considering ONLY other elements and
comments
    [repeat chardata]
    | [element]
    | [comment]
end define

define chardata               % Definition [14] XML 1.0 Recommendation
    [not '< ] [token]
end define

define attribute              % From Definition [41] XML 1.0 Recommendation
    [SP] [id] '=' [attvalue]
end define

define attvalue              % From Definition [10] XML 1.0 Recommendation
    [stringlit]
    | [charlit]
end define

define etag                   % From Definition [42] XML 1.0 Recommendation
    '< [SPOFF] !/[id] !> [SPON]
end define

```

SQL Grammar

% TXL Grammar For SQL Statements Needed For Our System

% Shruti Atre

% Queen's University, August 2007

```
define sqlStatement                % Grammar is limited to few types of SQL statements
    [tableDefinition] |
    [insertStatement] |
    [alterStatement]
end define
```

% SQL-Data Definition Language (DDL)

```
define tableDefinition
    CREATE TABLE [tableName] (    [IN]
    [list tableAttrDefinition]
    [opt primaryKeyDefinition]
    [opt foreignKeyDefinition] ) ; [EX][NL]
end define
```

```
define tableName
    [id]
end define
```

```
define tableAttrDefinition
    [NL] [tableAttribute] [dataType]
end define
```

```
define dataType                    % Limited set of allowed data types for our purpose
    VARCHAR [opt int]
    | INTEGER
    | TEXT                          % Equivalent of CLOB (Character Large Objects) in MySQL
end define
```

```
define tableAttribute
    [id]
end define
```

```
define int
    '( [integernumber] )'
end define
```

```
define primaryKeyDefinition
    , [NL] PRIMARY KEY ( [tableAttributelist] )
end define
```

```
define foreignKeyDefinition
    , [NL] [list foreignKey] [NL]
```

```

end define

define foreignKey
    FOREIGN KEY ( [tableAttributelist] ) REFERENCES [tableName] ( [tableAttributelist] )
end define

define tableAttributelist
    [list tableAttribute]
end define

% SQL-Data Manipulation Language (DML). Currently grammar for only INSERT is needed.
define insertStatement
    INSERT INTO [tableName] ( [list tableAttribute] ) VALUES ( [list stringlit] ) ;    [NL]
end define

define alterStatement
    ALTER TABLE [tableName] [list alterSpecification] ; [NL]
end define

define alterSpecification
    MODIFY [tableAttrDefinition] [opt colPosition]
end define

define colPosition
    FIRST | AFTER [tableAttribute]
end define

```

Appendix B

TXL Programs

MarkTagIds.txt

```
% TXL Program To Explicitly Mark Unique Tag Ids
% Shruti Atre
% Queen's University, August 2007

#pragma -w 32768

include "XML_Element_v8.Grammar"

% Grammar override to be able to store all the tag elements as table elements
redefine program
    ... |
    [repeat element]
end redefine

% This rule adds unique id attribute to all elements in the input document
rule main
    % Import command line arguments to extract the filename for storing all "table_elems"
    import TXLargs [repeat stringlit]
    deconstruct * TXLargs
        "-file" FileName [stringlit]

    % Find every element node
    replace $ [element]
        <Tag [id] TagAttrs [repeat attribute]>
            Contents [content]
        </Tag>
    deconstruct not * [attribute] TagAttrs % Make sure it doesn't already
    have a tagId attribute
        'tagId= Value [attvalue]

    construct UniqueTagId [id] % Create a new tag id for it that is unique based
    on its tagname
        Tag [ ! ]
    construct UniqueTagIdStr [stringlit]
        _ [quote UniqueTagId]

    % Output a list of all table elements to an external file. This is needed to maintain
    referential integrity in the relational schema.
    construct TagNameStr [stringlit]
        _ [quote Tag]
    construct TableElemTag [element]
```

```
        <table_elems 'tableName=TagNameStr 'tagId=UniqueTagIdStr ></table_elems>
construct FileOutput [element]
    TableElemTag [fput FileName]

% Replace the node with the tag id as a new attribute
by
    <Tag 'tagId=UniqueTagIdStr TagAttrs>
        Contents
    </Tag>
end rule
```

ExplicateContains.txt

```
% TXL Program To Explicitly Mark The Containment Relationship In The Input XML
Document
% Shruti Atre
% Queen's University, August 2007

#pragma -w 32768

include "XML_Element_v8.Grammar"

function main
  replace [program]
    P [program]
  by
    P [markContainsRelation]
end function

% This rule adds a new tag element to indicate the containment relationship
rule markContainsRelation
  % Find every element node
  replace $ [element]
    <ParentTag [id] ParentAttrs [repeat attribute]>
      Children [repeat sub_content]
    </ParentTag>
  deconstruct not ParentTag
    'contained_elem          % Make sure we are not dealing with a contained_elem
node itself

  % Check to make sure that this parent does have child elements and not just chardata
  deconstruct * [tag_content] Children
    SomeTagContent [tag_content]

  % Extract the tag Id of the parent
  deconstruct * [attribute] ParentAttrs
    'tagId= ParentId [attvalue]

  % For each child that is an element add the new tag and the attributes to indicate its
  parent's id
  construct NewChildren [repeat sub_content]
    _ [formNewChildren ParentId each Children]
  by
    <ParentTag ParentAttrs>
      NewChildren
    </ParentTag>
end rule
```

```

% This function forms the new content with appropriate sub_content enclosed in the
"contained_elem" tags
function formNewChildren ParentId [attvalue] Child [sub_content]
    replace [repeat sub_content]
        PrevOnes [repeat sub_content]
    construct NewChild [sub_content]
        Child [addContainedByTag ParentId]
    by
        PrevOnes [. NewChild]
end function

% This function encloses each child tag within a tag that indicates its id as well as the parent's id
using attributes
function addContainedByTag ParentId [attvalue]
    replace [sub_content]
        <ChildTag [id] ChildAttrs [repeat attribute]>
            ChildContent [content]
        </ChildTag>
    % Extract the tag Id of the child tag
    deconstruct * [attribute] ChildAttrs
        'tagId= ChildId [attvalue]
    % Make a new child element with 2 new attributes: 'childId' = child's tag id and 'parentId'
= parent's tag id
    construct NewChild [sub_content]
        <contained_elem 'childId=ChildId 'parentId=ParentId >
            <ChildTag ChildAttrs >
                ChildContent
            </ChildTag>
        </contained_elem>
    by
        NewChild
end function

```

ExplicateSequences.txt

```
% TXL Program To Explicitly Mark The Sequencing Relationship In The Input XML Document
% Shruti Atre
% Queen's University, August 2007
```

```
#pragma -w 32768
```

```
include "XML_Element_v8.Grammar"
```

```
function main
```

```
    % Export the starting element number so that they can be accessed by other functions
```

```
    export NumInSeq [number]
```

```
        1
```

```
    replace [program]
```

```
        P [program]
```

```
    by
```

```
        P [markSequence]
```

```
end function
```

```
% This rule adds a new tag element to indicate the sequencing relationship
```

```
rule markSequence
```

```
    skipping [element]
```

```
    % Find every element node
```

```
    replace $ [element]
```

```
        <Tag [id] TagAttrs [repeat attribute]>
```

```
            Contents [content]
```

```
        </Tag>
```

```
    deconstruct not Tag
```

```
        'sequence_elem
```

```
        % Make sure it is not the sequence_elem tag
```

```
itself
```

```
    deconstruct * [attribute] TagAttrs
```

```
    % Get the tag Id for the node under
```

```
consideration
```

```
        'tagId= TagId [attvalue]
```

```
    import NumInSeq [number]
```

```
    % Get the sequence number that indicates the
```

```
element's position in the document order
```

```
    construct NumInSeqStr [stringlit]
```

```
        _ [quote NumInSeq]
```

```
    export NumInSeq
```

```
    % Export a new sequence number that indicates
```

```
the next element's position in the document order
```

```
        NumInSeq [+ 1]
```

```
    % Replace the original tag with the sequence element for the tag considered
```

```
by
```

```
    <sequence_elem 'numInSeq=NumInSeqStr 'elemId= TagId >
```

```
                <Tag TagAttrs>
                    Contents [markSequence]
                </Tag>
            </sequence_elem>
end rule
```

GetCleanTags.txtl

```
% TXL Program To Search Through XML-like Input For A Specific Set Of Tags
% Shruti Atre
% Queen's University, August 2007

#pragma -w 32768

include "XML_Element_v8.Grammar"
include "includes/GetSpecificTags.txtl"

% Grammar overrides for marking interesting elements as elements that match the search tags
redefine element
    ... | [not token] [interesting_element]
end redefine

define interesting_element
    [element]
end define

% Input and output is an XML doc or a sequence of interesting elements
redefine program
    ... |
    [repeat interesting_element]
end redefine

% Main function
function main
    % Import command line arguments
    import TXLargs [repeat stringlit]
    deconstruct * TXLargs
        "-tag" TagArgs [repeat stringlit]
    replace [program]
        P [program]
    by
        % Get the tags that are specified and clean up their contents to remove all nested
        XML tags
        P [getSpecificTags TagArgs][cleanUpFoundTags]
end function

% Rule to clean the content of each tag found by removing nested XML tags
rule cleanUpFoundTags
    replace $ [interesting_element]
        EachElement [element]
    by
        EachElement [removeTags]
end rule
```

```
% This rule removes nested tags in its scope
rule removeTags
  replace [repeat sub_content]
    <TagName [id] Attr [repeat attribute]>
      Contents [repeat sub_content]
    </TagName>
  RemainingContent [repeat sub_content]
  by
    Contents [. RemainingContent]
end rule
```

GetSpecificTags.txtl

```
% TXL Program To Search Through An XML Document For The Specific Tags
% Shruti Atre
% Queen's University, August 2007

% Main function
function getSpecificTags TagArgs [repeat stringlit]
  construct SearchTags [repeat id]
    _ [addSearchTag each TagArgs]

  replace [program]
  Input [program]

  construct InputAfterSearch [program]
    Input [lookForTag each SearchTags]

  % Now extract ONLY the interesting elements
  construct InterestingElems [repeat interesting_element]
    _ [ ^ InputAfterSearch]

  by
    InterestingElems
end function

% Function to add the passed tagname to the list of tag names to be searched for
function addSearchTag TagArg [stringlit]
  replace [repeat id]
    PrevOnes [repeat id]
  construct OptTagName [opt id]
    _ [parse TagArg]
  deconstruct OptTagName
    TagName [id]
  by
    PrevOnes [. TagName]
end function

% Rule to search for elements that match the passed tag name
rule lookForTag TagName [id]
  replace $ [sub_content]
    <TagName Attrs [repeat attribute]>
      Contents [content]
    </TagName>
  % If the tag matches your search tag name then mark it as an "interesting_element"
  construct InterestingElem [interesting_element]
    <TagName Attrs>
      Contents
    </TagName>
```

```
        % Replace the tag by its interesting version and appending contents to respective
outputfile
        by
        InterestingElem
end rule
```

GetDBInXMLFormat.txtl

```
% TXL Program To Generate The Intermediate Representation For The Relational Database
% Shruti Atre
% Queen's University, August 2007
```

```
#pragma -w 32768
```

```
include "XML_Element_v8.Grammar"
include "includes/GetDBInXMLForm.txtl"
```

```
redefine program
    [repeat element]
    | ...
end redefine
```

```
function main
    % Import command line arguments
    import TXLargs [repeat stringlit]
    deconstruct * TXLargs
        "-tag" TagArgs [repeat stringlit]

    replace [program]
        P [repeat element]

    construct TableSchemas [repeat sub_content]
        _ [getTableFor each TagArgs]
    by
        <database name="db1">
            TableSchemas
        </database>
end function
```

```
% Function to generate the intermediate representation of the table schema and table data for
passed tag name argument
```

```
function getTableFor TagName [stringlit]
    % Read the input filename whose name is derived from the tag name argument
    construct InputFileName [stringlit]
        TagName [+ ".xml"]
    construct DirName [stringlit]
        "CleanTags/"
    construct InputFile [stringlit]
        DirName [+ InputFileName]
    construct Input [program]
        _ [read InputFile]

    % Get schema and data for the table corresponding to passed TagName
    construct SchemaDataOutput [program]
```

```
        Input [getDBInXMLForm]
deconstruct SchemaDataOutput
    <table>
        SchemaAndData [repeat sub_content]
    </table>

% Append the new sub_content to the ones created previously for other tag names
replace [repeat sub_content]
    PrevOnes [repeat sub_content]
by
    PrevOnes [. SchemaAndData]
end function
```

GetDBInXMLForm.txl

```
% TXL Program To Create The Intermediate Representation Of The Database
% Shruti Atre
% Queen's University, August 2007
```

```
#pragma -w 327688
```

```
% Grammar overrides
redefine program
    [repeat element]
    | ...
end redefine
```

```
redefine sub_content
    ...
    | [not token] [repeat element]
end redefine
```

```
% Function to create the table schema and table data
function getDBInXMLForm
```

```
    replace [program]
        P [program]
    % Construct the table schema
    construct TableSchema [program]
        P [makeSchema]
    deconstruct TableSchema
        TableSchemaContent [tag_content]
    % Construct the table data
    construct TableData [program]
        P [makeTable]
    deconstruct TableData
        TableDataContent [tag_content]
    by
        <table>
            TableSchemaContent
            TableDataContent
        </table>
```

```
end function
```

```
% Function to create the intermediate representation for the database schema
function makeSchema
```

```
    replace [program]
        P [repeat element]
    % Extracting the name for the table from the first interesting element
    deconstruct * [element] P
        <TagName [id] Attrs [repeat attribute]>
        Contents [repeat sub_content]
```

```

    </TagName>
    construct TableName [stringlit]
        _ [quote TagName]

    % Obtaining all possible attribute names from all elements without any duplicate entries
    construct AttrsFromAllElems [repeat attribute]
        _ [getAllAttributes each P]
    construct AllAttrs [repeat attribute]
        AttrsFromAllElems [removeDuplicateAttrs]

    % Constructing the field representation for attribute which becomes a field in table
    construct TableSchema [repeat sub_content]
        _ [makeSchemaField 'Field each AllAttrs ]

    % Constructing the field representation for the "tag_content" field of table
    % This field should be added ONLY for tags other than contained_elem and
sequence_elem
    construct LastFieldSchema [sub_content]
        _ [getLastFieldSchema TagName]

    % Constructing the whole schema for the table
    construct EntireTableSchema [repeat sub_content]
        TableSchema [. LastFieldSchema]
    by
        <table_structure 'name=TableName >
            EntireTableSchema
        </table_structure>
end function

% Function to obtain all possible attribute names from all elements
function getAllAttributes Element [element]
    replace [repeat attribute]
        OldAttrs [repeat attribute]
    deconstruct Element
        <TagName [id] NewAttrs [repeat attribute]>
            Contents [repeat sub_content]
        </TagName>
    by
        OldAttrs [. NewAttrs]
end function

% Function to remove duplicate attribute names
function removeDuplicateAttrs
    replace [repeat attribute]
        AllAttrs [repeat attribute]
    construct NoDupsAttrs [repeat attribute]
        _ [checkForDup each AllAttrs]
    by

```

```

        % Attributes With No Duplicates
        NoDupsAttrs
    end function

% Function to check if the passed attribute is already present in the attributes obtained so far
function checkForDup AttrArg [attribute]
    replace [repeat attribute]
        Attrs [repeat attribute]
    deconstruct AttrArg
        AttrName [id] = AttrValue [attvalue]
    deconstruct not * [id] Attrs
        AttrName
    by
        Attrs [. AttrArg]
end function

% Function to create the intermediate representation for the different fields in the table schema
function makeSchemaField FieldName [id] AttrArg [attribute]
    deconstruct AttrArg
        AttrName [id] '= Value [stringlit]
    % Convert AttrName to stringlit
    construct AttrNameStr [stringlit]
        _ [quote AttrName]
    % Create field
    replace [repeat sub_content]
        PrevOnes [repeat sub_content]
    construct NewField [sub_content]
        <field FieldName '= AttrNameStr> </field>
    by
        PrevOnes [. NewField]
end function

% Function to add tag content field ONLY for tags other than table_elem, contained_elem and
sequence_elem
function getLastFieldSchema TagName [id]
    replace [sub_content]
        NullContent [sub_content]
    deconstruct not TagName
        'contained_elem
    deconstruct not TagName
        'sequence_elem
    deconstruct not TagName
        'table_elems
    by
        <field Field="tag_content"></field>
end function

% Function to create the intermediate representation for the database table

```

```

function makeTable
  replace [program]
    P [repeat element]

  % Extracting the name for the table from the first interesting element
  deconstruct * [element] P
    <TagName [id] Attrs [repeat attribute]>
      Contents [repeat sub_content]
    </TagName>
  construct TableName [stringlit]
    _ [quote TagName]

  % Converting the interesting elements to intermediate representation's format
  construct RowContent [program]
    P [convertToRowFormat]

  deconstruct RowContent
    AllRows [repeat element]
  by
    <table_data 'name=TableName >
      AllRows
    </table_data>
end function

% Rule to form the rows of the table from the interesting elements
rule convertToRowFormat
  skipping [element]
  replace $ [element]
    <TagName [id] Attrs [repeat attribute]>
      Content [repeat sub_content]
    </TagName>
  % Convert each attribute of the tag to a table field
  construct AttrFields [repeat sub_content]
    _ [makeTableField 'name each Attrs ]

  % This field should be added ONLY for tags other than contained_elem and
sequence_elem
  construct ContentField [sub_content]
    _ [getLastFieldContent TagName Content]

  construct AllFields [repeat sub_content]
    AttrFields [. ContentField]
  by
    <row> AllFields      </row>
end rule

% Function to create a field from attribute passed as an argument
function makeTableField FieldName [id] AttrArg [attribute]

```

```

    deconstruct AttrArg
      AttrName [id] '= Value [stringlit]
    % Convert AttrName to stringlit
    construct AttrNameStr [stringlit]
      _ [quote AttrName]
    % Convert Value from type stringlit to type sub_content
    construct OptValue [opt sub_content]
      _ [parse Value]
    deconstruct OptValue
      AttrVal [sub_content]

    replace [repeat sub_content]
      PrevOnes [repeat sub_content]
    % Create field
    construct NewField [sub_content]
      <field fieldName '=' AttrNameStr > AttrVal </field>
    by
      PrevOnes [. NewField]
  end function

% Function to determine if the last field (tag_content) should be included or not
function getLastFieldContent TagName [id] Content [repeat sub_content]
  replace [sub_content]
    NoContent [sub_content]
  deconstruct not TagName
    'contained_elem
  deconstruct not TagName
    'sequence_elem
  deconstruct not TagName
    'table_elems
  by
    <field name="tag_content"> Content </field>
end function

```

GenerateSQL.txl

```
% TXL Program To Convert DB In XML format To SQL Statements
% Shruti Atre
% Queen's University, August 2007
```

```
#pragma -w 32768
#pragma -esc ""
```

```
include "XML_Element_v8.Grammar"
include "includes/SQL.Grammar"
include "includes/getCreateStatements.txl"
include "includes/getInsertStatements.txl"
include "includes/GetSpecificTags.txl"
```

```
% Grammar overrides for marking interesting elements as elements that match the search tags
redefine sub_content
```

```
    ...
    | [interesting_element]
end redefine
```

```
define interesting_element
    [element]
end define
```

```
redefine program
    ... |
    [repeat interesting_element]
    | [repeat sqlStatement]
end redefine
```

```
function main
    replace [program]
        P [program]

    % Form the SQL "CREATE" statements from input
    construct SQLCreateStmts [program]
        P [getCreateStatements]
    deconstruct SQLCreateStmts
        CreateStmts [repeat sqlStatement]

    % Form the SQL "INSERT" statements from input
    construct SQLInsertStmts [program]
        P [getInsertStatements]
    deconstruct SQLInsertStmts
        InsertStmts [repeat sqlStatement]
```

```

relation
    % This statement modifies the datatype of the 'numInSeq' attribute of the 'sequence_elem'
    construct AlterTableStmt [sqlStatement]
        'ALTER TABLE sequence_elem MODIFY numInSeq INTEGER ;

    % Combine the SQL "CREATE", "INSERT" and other statements to create final output
    construct AllStatements [repeat sqlStatement]
        CreateStmts [. InsertStmts][. AlterTableStmt]
    construct SQLStatements [program]
        AllStatements
    by
        SQLStatements
end function

```

GetCreateStatements.txl

% TXL Program To Convert Intermediate Representation Of The Relational Table Schema In XML Format To SQL CREATE Statements

% Shruti Atre

% Queen's University, August 2007

% Main function

```
function getCreateStatements
  replace [program]
    P [program]
  by
    P [createTables]
end function
```

% Function to make a SQL table definition statement for every "table_structure" tag found in the input

```
function createTables
  replace [program]
    P [program]
  % Construct the list of tag names to search for to include "table_structure"
  construct TagArgs [repeat stringlit]
    "table_structure"
  % Get all the "table_structure" tags from the intermediate representation of the database
  construct TableStructures [program]
    P [getSpecificTags TagArgs]
  by
    TableStructures [makeTableDefinitions]
end function
```

% Function to formulate the SQL "CREATE" statements

```
function makeTableDefinitions
  replace [program]
    TableStructElems [program]

  % Get the interesting elements i.e. the "table_structure" tags
  deconstruct TableStructElems
    AllTableStructures [repeat interesting_element]

  % Create a table definition for each "table_structure" tag found
  construct NewProgram [repeat sqlStatement]
    _ [createTableDefinitions each AllTableStructures]
  by
    NewProgram
end function
```

% Function to transform

```
% <table_structure name="TableName">
```

```

%          <field Field="nameOfField1"></field>
%          <field Field="nameOfField2"></field>
%      </table_structure>
% into a SQL CREATE statement
%      CREATE TABLE TableName (
%          nameOfField1 VARCHAR(50), nameOfField2 VARCHAR(50)
%      )
function createTableDefinitions TableStructure [interesting_element]
% Deconstruct the argument to get the table name and its fields
deconstruct TableStructure
    <table_structure name= TabName[stringlit] >
        TableFields [repeat sub_content]
    </table_structure>

% Make a list of the fields for the SQL "CREATE" statement
construct TableAttrList [list tableAttrDefinition]
    _ [makeTableAttribute each TableFields]

construct PrimaryKey [opt primaryKeyDefinition]
    _ [getPrimaryKey TabName][getPrimaryKeyForContain
TabName][getPrimaryKeyForSequence TabName]

construct ForeignKey [opt foreignKeyDefinition]
    _ [getForeignKeyForContain TabName][getForeignKeyForSequence TabName]

% Convert TableName from type 'stringlit' to type 'id'
construct OptTabName [opt id]
    _ [parse TabName]
deconstruct OptTabName
    TableName [id]

replace * [repeat sqlStatement]
    % Nothing

% Form the "CREATE" statement using all the pieces created above
by
    CREATE TABLE TableName (
        TableAttrList
        PrimaryKey
        ForeignKey
    );
end function

% Function to transform
%      <field Field="fieldName"></field>
% into field part of the SQL "CREATE" statment
%      fieldName VARCHAR(n)
function makeTableAttribute TableField [sub_content]

```

```

deconstruct TableField
    <field Field= Name [stringlit] ></field>

    % Convert FieldName from type 'stringlit' to type 'id'
construct OptFieldName [opt id]
    _ [parse Name]
deconstruct OptFieldName
    FieldName [id]

    % Get the appropriate data type for the field
construct defaultDataType [dataType]
    VARCHAR(50)
construct checkedDataType [dataType]
    defaultDataType [getDataType FieldName]

    % Construct the new table field
construct NewTableField [tableAttrDefinition]
    FieldName checkedDataType

replace [list tableAttrDefinition]
    PrevOnes [list tableAttrDefinition]
by
    PrevOnes [, NewTableField]
end function

% Function to determine the appropriate data type for tag content fields i.e. CLOB (TEXT in
MySQL)
function getDataType FieldName [id]
    replace [dataType]
        DataType [dataType]
    deconstruct FieldName
        'tag_content'
    by
        TEXT
end function

% Function to determine the appropriate primary key for the passed table name
function getPrimaryKey TabName [stringlit]
    replace [opt primaryKeyDefinition]
        PrimaryKey [opt primaryKeyDefinition]

    deconstruct not * [stringlit] TabName
        "contained_elem"
    deconstruct not * [stringlit] TabName
        "sequence_elem"

    construct PrimKeyAttrs [list tableAttribute]
        'tagId

```

```

        by
            , PRIMARY KEY ( PrimKeyAttrs )
end function

```

```

% Function to determine the appropriate primary key for the passed table name i.e.
contained_elem table

```

```

function getPrimaryKeyForContain TabName [stringlit]
    replace [opt primaryKeyDefinition]
        PrimaryKey [opt primaryKeyDefinition]
    deconstruct * [stringlit] TabName
        "contained_elem"

    construct PrimKeyAttrs [list tableAttribute]
        'parentId,childId
    by
        , PRIMARY KEY ( PrimKeyAttrs )
end function

```

```

% Function to determine the appropriate primary key for the passed table name i.e.
sequence_elem table

```

```

function getPrimaryKeyForSequence TabName [stringlit]
    replace [opt primaryKeyDefinition]
        PrimaryKey [opt primaryKeyDefinition]
    deconstruct * [stringlit] TabName
        "sequence_elem"

    construct PrimKeyAttrs [list tableAttribute]
        'elemId
    by
        , PRIMARY KEY ( PrimKeyAttrs )
end function

```

```

% Function to determine the appropriate foreign key for the passed table name i.e.
contained_elem table

```

```

function getForeignKeyForContain TabName [stringlit]
    replace [opt foreignKeyDefinition]
        ForeignKey [opt foreignKeyDefinition]
    deconstruct * [stringlit] TabName
        "contained_elem"
    construct ForeignKeyForContain [opt foreignKeyDefinition]
        , FOREIGN KEY ( 'childId ) REFERENCES 'table_elems ( 'tagId )
        , FOREIGN KEY ( 'parentId ) REFERENCES 'table_elems ( 'tagId )
    by
        ForeignKeyForContain
end function

```

```

% Function to determine the appropriate foreign key for the passed table name i.e. sequence_elem
table

```

```
function getForeignKeyForSequence TabName [stringlit]
  replace [opt foreignKeyDefinition]
    ForeignKey [opt foreignKeyDefinition]
  deconstruct * [stringlit] TabName
    "sequence_elem"
  construct ForeignKeyForSequence [opt foreignKeyDefinition]
    , FOREIGN KEY ( 'elemId ) REFERENCES 'table_elems ( 'tagId )
  by
    ForeignKeyForSequence
end function
```

GetInsertStatements.txl

% TXL Program To Convert Intermediate Representation Of The Relational Table Data In XML
Format To SQL INSERT Statements

% Shruti Atre

% Queen's University, August 2007

% Main function

```
function getInsertStatements
  replace [program]
    P [program]
  by
    P [createInsertStmts]
end function
```

% Function to make a SQL table "INSERT" statement for every "table_data" tag found in the
input

```
function createInsertStmts
  replace [program]
    P [program]
  % Construct the list of tag names to search for to include "table_data"
  construct TagArgs [repeat stringlit]
    "table_data"
  % Get all the "table_data" tags from the intermediate representation of the database
  construct TableInserts [program]
    P [getSpecificTags TagArgs]
  by
    TableInserts [makeTableInsertStmts]
end function
```

% Function to formulate the SQL "INSERT" statements

```
function makeTableInsertStmts
  replace [program]
    TableDataElems [program]

  % Get the interesting elements i.e. the "table_data" tags
  deconstruct TableDataElems
    AllTableInserts [repeat interesting_element]

  % Create a table insert for each "table_data" tag found
  construct NewProgram [repeat sqlStatement]
    _ [createTableInserts each AllTableInserts]
  by
    NewProgram
end function
```

% Function to transform

```
% <table_data name="TableName">
```

```

%           <row>
%           <field name="nameOfField1">content1</field>
%           <field name="nameOfField2">content2</field>
%           </row>
% </table_data>
% into a SQL INSERT statement
%   INSERT INTO TableName (nameOfField1, nameOfField2) VALUES ("content1",
"content2")
function createTableInserts TableData [interesting_element]
    % Deconstruct the argument to get the table name and its fields
    deconstruct TableData
        <table_data name= TabName[stringlit] >
            TableRows [repeat sub_content]
        </table_data>
    % Convert TableName from type 'stringlit' to type 'id'
    construct OptTableName [opt id]
        _ [parse TabName]
    deconstruct OptTableName
        TableName [id]

        construct InsertStatements [repeat sqlStatement]
            _ [createInsertFor TableName each TableRows]

        replace * [repeat sqlStatement]
            % Nothing
        by
            InsertStatements
end function

function createInsertFor TableName [id] Row [sub_content]
    deconstruct Row
        <row>
            RowFields [repeat sub_content]
        </row>

    % Form the ( "attribute1" , "attribute2") part of the INSERT statement
    construct ListOfAttrs [list tableAttribute]
        _ [makeAttrList each RowFields]

    % Form the VALUES( "stringlit1" , "stringlit2") part of the INSERT statement
    construct ListOfValues [list stringlit]
        _ [makeValuesList each RowFields]

    replace * [repeat sqlStatement]
        % Nothing
    by
        INSERT INTO TableName ( ListOfAttrs ) VALUES ( ListOfValues ) ;
end function

```

```

% Function to transform
%   <field Field="fieldName"> FieldContent </field>
% into field part of the SQL "INSERT" statment
%   fieldName,
function makeAttrList TableField [sub_content]
    replace [list tableAttribute]
        PrevOnes [list tableAttribute]
    deconstruct TableField
        <field name= Name [stringlit] > FieldContent [content] </field>

        % Convert FieldName from type 'stringlit' to type 'id'
    construct OptFieldName [opt id]
        _ [parse Name]
    deconstruct OptFieldName
        FieldName [id]

        construct NewTableField [tableAttribute]
            FieldName
        by
            PrevOnes [, NewTableField]
end function

% Function to transform
%   <field Field="fieldName"> FieldContent </field>
% into value part of the SQL "INSERT" statment
%   "FieldContent",
function makeValuesList TableField [sub_content]
    replace [list stringlit]
        PrevOnes [list stringlit]
    deconstruct TableField
        <field name= Name [stringlit] > Content [content] </field>

        % Convert Content from type 'content' to type 'stringlit'
    construct FieldContent [stringlit]
        _ [quote Content]
    by
        PrevOnes [, FieldContent]
end function

```