Querying over Heterogeneous XML Schemas in a Content Management System

by

Fabian F. Morgan

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 24, 2002

Copyright 2002 Fabian F. Morgan. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and
distribute publicly paper and electronic copies of this thesis
and to grant others the right to do so.

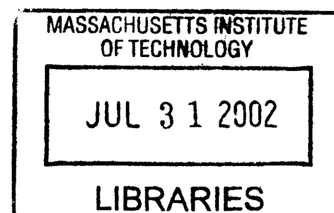
Author______________________________
Department of Electrical Engineering and Computer Science
May 24, 2002

Certified by______________________________
Peter Donaldson
Thesis Supervisor

Certified by______________________________
Kurt Fendt
Thesis Supervisor

Accepted by______________________________
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

BARKER


MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUL 3 1 2002
LIBRARIES

Querying over Heterogeneous XML Schemas in a Content Management System

by
Fabian F. Morgan
Submitted to the
Department of Electrical Engineering and Computer Science

May 24, 2002

In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

This thesis focuses on designing and implementing an efficient method for querying over and updating multiple schemas in a single XML repository. The repository is an integral part of a multimedia content management framework, known as Metamedia. The documents entered into the repository are various pieces of literary and multimedia works along with their corresponding bibliographic information and annotation metadata in different XML representations. These representations include the MPEG-7, TEI, DocBook and Dublin Core schemas. I solve the querying problem by generating replicated copies of documents in the different schemas through the use of XSLT stylesheets; the update and merge problem is solved by generating a set of node matches in the form of XPointers, then using the matches in an algorithm to update the original document. I conclude that while the more general problem is not completely solvable until advanced techniques allow computers to understand semantics in the text, my solution is adequate for the specific Metamedia domain.

Thesis Committee:

Peter Donaldson, Professor of Literature
Kurt Fendt, Research Associate, Comparative Media Studies
Christopher York, Technical Director, Metamedia Project

ACKNOWLEDGEMENTS

I would like to thank the members of my thesis committee, Professor Peter Donaldson, Kurt Fendt, and Christopher York for their financial support and guidance in making this thesis a reality. I would also like to thank my parents, sister, and church family for the love, encouragement, and enduring support they have shown towards me during my years at MIT.

TABLE OF CONTENTS

1 INTRODUCTION 8

1.1 PROJECT BACKGROUND 8
1.2 MOTIVATION 9
1.3 GOALS 10
1.4 THESIS OUTLINE 11

2 TECHNOLOGIES IN THE FRAMEWORK 12

2.1 XML 12
2.1.1 TEI and DocBook 13
2.1.2 Dublin Core and RDF 14
2.1.3 MPEG-7 14
2.2 XSLT 15
2.2.1 Xalan 15
2.3 XPOINTER SPECIFICATION 15
2.4 XUPDATE SPECIFICATION 16
2.5 SCALABLE VECTOR GRAPHICS (SVG) 16
2.6 APACHE COCOON 17
2.7 SYSTEM ARCHITECTURE 17

3 RESEARCH PROBLEM 19

3.1 QUERYING OVER MULTIPLE SCHEMAS 19
3.2 UPDATING/MERGING DISPARATE SCHEMAS 20

4 RELATED WORK 23

4.1 ONTOLOGIES 23
4.2 DETECTING CHANGES IN STRUCTURED DOCUMENTS 24

5 DESIGN AND IMPLEMENTATION 26

5.1 QUERYING OVER MULTIPLE SCHEMAS 26
5.2 UPDATE AND MERGE 27
5.2.1 Matching Problem: Introduction 28
5.2.2 Matching Problem: Original to Translated 29
5.2.3 Matching Problem: Translated to Updated 34
5.2.4 Matching Problem: Updated to Derived 43
5.2.5 Update and Merge Algorithm 44

6 ANALYSIS 47

6.1 COMPARISON WITH OTHER METHODS 47
6.2 LIMITATIONS OF ALGORITHMS 48

7 CONCLUSION AND FUTURE WORK 50

8 REFERENCES 51

9 APPENDIX 53

9.1 TRANSLATESCHEMA.JAVA 53
9.2 MATCHINGMANAGER.JAVA 53
9.3 XUPDATETOOL.JAVA 56
9.4 XUPDATESAXHANDLER.JAVA 58
9.5 UPDATEMERGETOOL.JAVA 65
9.6 TEI_TO_DC.XSL 70

LIST OF FIGURES

FIGURE 2-1: EXAMPLE TEI SNIPPET. 13

FIGURE 2-2: HAMLET SNIPPET WITHOUT TEI ENCODING 13

FIGURE 2-3: EXAMPLE DUBLIN CORE FILE 14

FIGURE 2-4: SYSTEM ARCHITECTURE 18

FIGURE 3-1: PICTORIAL DESCRIPTION OF UPDATE AND MERGE 22

FIGURE 5-1: ONE-TO-ONE MAPPINGS 26

FIGURE 5-2: COMPOUND-TO-ONE MAPPINGS 27

FIGURE 5-3: COMPOUND-TO-COMPOUND MAPPINGS 27

FIGURE 5-4: PICTORIAL OUTLINE OF UPDATE AND MERGE ALGORITHM 28

FIGURE 5-5: RUNNING EXAMPLE - ORIGINAL DOCUMENT 29

FIGURE 5-6: RUNNING EXAMPLE - TRANSLATED DOCUMENT 30

FIGURE 5-7: XSLT STYLESHEET WITH EXTENSION FUNCTION CALLS 31

FIGURE 5-8: ALGORITHM CREATEMATCH 33

FIGURE 5-9: XUPDATE QUERY FOR THE RUNNING EXAMPLE 35

FIGURE 5-10: RUNNING EXAMPLE - UPDATED DOCUMENT 35

FIGURE 5-11: ALGORITHM GENERATEUPDATEMATCHINGS 42

FIGURE 5-12: ALGORITHM UPDATEANDMERGE 46

FIGURE 6-1: EXAMPLE OF NON-EXISTENT INVERSE MAPPING 48

LIST OF TABLES

TABLE 5-1: MATCHINGS TABLE AFTER THE ORIGINAL DOCUMENT HAS BEEN TRANSFORMED TO THE TRANSLATED ONE 34

TABLE 5-2: MATCHES FOR XUPDATE:REMOVE - CASE 1 36

TABLE 5-3: MATCHES FOR XUPDATE:REMOVE - CASE 2 36

TABLE 5-4: MATCHES FOR XUPDATE:REMOVE - CASE 3 37

TABLE 5-5: MATCHES FOR XUPDATE:REMOVE - CASE 4 37

TABLE 5-6: MATCHES FOR XUPDATE:INSERT-AFTER – CASE 1 38

TABLE 5-7: MATCHES FOR XUPDATE:INSERT-AFTER – CASE 2 39

TABLE 5-8: MATCHES FOR XUPDATE:INSERT-AFTER – CASE 3 40

TABLE 5-9: MATCHES FOR XUPDATE:INSERT-AFTER – CASE 4 41

TABLE 5-10: MATCHINGS TABLE AFTER THE ALGORITHM GENERATEUPDATEMATCHINGS HAS RUN 43

TABLE 5-11: COMPLETE MATCHINGS TABLE 44

1 INTRODUCTION

This thesis addresses a fundamental issue in the design of a flexible and scalable multimedia-based content management framework, known as Metamedia. In this chapter, I present an overview of the project, including the motivations and some of its goals.

1.1 Project Background

Metamedia is a two-year research project undertaken in the Comparative Media Studies (CMS) program within the School of Humanities, Arts, and Social Sciences at MIT. This project is part of an institute-wide commitment[1] to using technology to provide emerging approaches to distance learning and training. It intends to bring about an educational reform that encourages students to think deeper and harder about interpretations of multimedia documents in their proper context. Through the use of multimedia-based teaching materials, the project will help students to develop the skills and ethics necessary for communicating in a global context [1].

The Metamedia framework contains a digital repository of recorded sounds, movies, photographs, and text documents. Teachers are able to refer to these resources in their classroom instructions, while students can use them as reference points in presentations or essays. In addition, the framework serves as a base through which professors and students at different institutions can seamlessly exchange data, thus encouraging new modes of interdisciplinary scholarship and collaboration [2].

There are a number of other programs at external institutions with similar research initiatives, including the Perseus Project at Tufts University [3] that contains a large repository of classical texts. Projects in a number of existing humanities sections at MIT, including Literature, Foreign Languages, Anthropology, and CMS, will be ported to the new framework. These include the Berliner sehen [4] project and the Shakespeare

[1] Other such initiatives at MIT include the OpenCourseWare Project, and the Open Knowledge Initiative.

Electronic Archive [5]. The following sections discuss the motivations and goals for the Metamedia project itself.

1.2 Motivation

The motivation for the Metamedia project stemmed from a realization of the limitations in the way some of the current projects are implemented. Most consist of hordes of static HTML pages that mix the multimedia content, its presentation, and the logic that controls content flow into one file. This mixing results in a number of significant drawbacks. First, it greatly reduces the flexibility of the project. For example, if one has the text of a play mixed with HTML tags that dictate how that play should be presented in a browser, then it becomes difficult to change the presentation of the text in the future. In order to change this presentation, a developer would have to first write a script that parses out the content of the play from all the HTML files, and then write a script that adds new HTML tags, corresponding to the new presentation, around the content. Changing the site presentation thus becomes a time-intensive chore, and no one would want to do it after several times. Of course, this is not desirable behavior; we would ideally like to be able to change the presentation many times with the least amount of work necessary.

A second drawback, which is closely related to the first, is that the mixing of presentation with content reduces the ability to share multimedia content with others. Continuing from the above scenario, if one wanted to share the text of a play with a colleague at another university, for example, either he would have to parse the content into his representation and send it to the colleague, or the colleague would have to take the HTML file and parse it into his own representation. One could imagine what would happen if everyone did this. A person would end up having multimedia content in many different representations, and thus would have to write a script corresponding to each representation that was specific to the way in which he wanted to present it in a browser. This would be a nightmare for code maintainability, and would prove a very inefficient way of developing web-based hypertext projects.

A third drawback of having logic mixed with content and presentation is that it results in a lack of code reusability. There are a number of functions that are common to many

projects. For example, in projects consisting of plays, it is likely that a common search query would be to return the content of a line number, or a set of line numbers. Having this logic mixed into static pages results in lots of unnecessary code duplication and rework, because there is no base set of components that could be used from project to project.

A final drawback is that the mixing restricts efficient development of the project. Ideally, one would want the ability for a graphic designer creating the presentational aspects of the project to work in parallel with a developer writing the logic code for the project, with the least amount of overlap as possible. However, clearly if the presentation, logic, and content are all in the same file, there is a lot of interaction among the developer types and hence it is likely for bugs to occur resulting from one person overwriting the work of another, etc.

The Metamedia project was started to solve these problems and more. Its specific goals are listed in the following section.

1.3 Goals

The Metamedia project was started to develop a robust framework that has the following properties:

- Flexible - relative ease of code maintainability and extensibility
- Contains reusable components
- Scalable to thousands of users
- Has good performance characteristics
- Has collaborative features, including the ability for students to add/view annotations for different documents
- Has built-in access control features

In addition to these technical goals, the Metamedia project intends to have these outcomes as well [2]:

- Creation of multi-media modules and curricular material for use within and outside MIT, as well as in distance education settings

- Creation and expansion of multi-media modules and their integration in communication intensive subjects and other HASS-D courses
- Development of a team of graduate and undergraduate students to consult with faculty in the development of media modules

1.4 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 describes the technologies we use in the framework in order to satisfy the goals above, the reasons for choosing them, and the resources needed in order to implement them. Chapter 3 gives a detailed explanation of an issue in designing such a system, which serves as my research focus. Chapter 4 gives descriptions of related work that has already been done in the field. Chapter 5 is the core of the thesis; it presents my solution to the problem described in Chapter 3. Chapter 6 compares my solution with earlier work. Finally, Chapter 7 concludes the thesis and lists ideas for future work.

2 TECHNOLOGIES IN THE FRAMEWORK

In this chapter, I outline some of the core technologies we used in developing the framework. The technologies are all open-source technologies and/or industry standards. We decided to use these open standards for the following reasons:

- First, they are either free or very inexpensive to obtain an academic license.
- Second, many developers have the ability to look at the source code and hence have the ability to fix or report bugs and soon as they find them. This generally tends to result in higher quality code.
- Finally, and most important, the use of open standards facilitates the exchange of materials across institutions. The use of internal, private representations makes it harder to share data more readily.

The following sections list the technologies we decided to use. Section 2.1 describes the schemas that are queried over in the system. Sections 2.2 through 2.4 list the technologies used in the implementation of the algorithms presented in Chapter 5. Sections 2.5 and 2.6 list technologies used for presentational aspects in the system. Section 2.7 concludes with an overview of the system architecture.

2.1 XML

XML is an acronym for Extensible Markup Language, and has become the de facto standard of representing structured information via the web. XML was designed to be human-legible and reasonably easy to create and process. Its predecessor, the Standard Generalized Markup Language (SGML), also provides arbitrary structure for documents; XML is, in essence, a restricted form of SGML [6].

In our system, we use XML to represent our textual content, as well as our annotations. Specifically, we chose TEI and DocBook for the textual content, a combination of the Dublin Core and RDF schemas for the representation of annotations, and MPEG-7 for audio and visual metadata. Note that the TEI and DocBook specifications are used for textual data, while the Dublin Core, RDF, and MPEG-7 specifications are "metadata" standards, that is, they are used to store data about data.

2.1.1 TEI and DocBook

TEI is the Text Encoding Initiative, which was initially founded by a consortium in 1987 to represent literary and linguistic texts in an encoding scheme that is both "maximally expressive and minimally obsolescent" [7]. Because the encoding is standardized, we can readily share our textual content in this representation with some confidence that other academic institutions would have their content in the same representation. A snippet of a TEI document, taken from Shakespeare's Hamlet is shown in Figure 2-1:

```
<div1 type ='Act' n='I'><head>ACT I</head>
<div2 type ='Scene' n='1'><head>SCENE I</head>
<stage rend="italic">
Enter Barnardo and Francisco, two Sentinels, at several doors</stage>
<sp><speaker>Barn<l part="Y">Who's there?</l></speaker></sp>
<sp><speaker>Fran<l>Nay, answer me. Stand and unfold
  yourself.</l></speaker></sp>
<sp><speaker>Barn<l part="i">Long live the King!</l></speaker></sp>
<sp><speaker>Fran<l part="m">Barnardo?</l></speaker></sp>
<sp><speaker>Barn<l part="f">He.</l></speaker></sp>
<sp><speaker>Fran<l>You come most carefully upon your
hour.</l></speaker></sp>
</div2>
</div1>
```

Figure 2-1: Example TEI Snippet.

The source for the play, without the TEI encoding, is shown in Figure 2-2:

```
1 ACT I. SCENE I.
2 Enter Barnardo and Francisco, two Sentinels, at several doors.
3 Barn.
4 Who's there?
5 Fran. Nay, answer me. Stand and unfold
6 yourself.
7 Barn. Long live the King!
8 Fran. Barnardo?
9 Barn. He.
10 Fran. You come most carefully upon your
11 hour.
```

Figure 2-2: Hamlet Snippet without TEI Encoding

DocBook [8] is another schema used to represent books, chapters, articles, or any other type of literary text in the repository.

2.1.2 Dublin Core and RDF

The Dublin Core Metadata Initiative is an organization "dedicated to promoting the widespread adoption of interoperable metadata standards and developing specialized metadata vocabularies for describing resources that enable more intelligent information discovery systems" [9]. RDF, the Resource Description Framework, was founded with similar goals [10]. We use a combination of the Dublin Core Element Set along with RDF fields to describe bibliographic information in our repository (see Figure 2-3 for an example). In addition, RDF fields are used to represent annotations to documents in our system. Dublin Core and RDF are both XML-based schemas.

```
<?xml version="1.0"?>
 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:dc="http://purl.org/dc/elements/1.1/">
   <rdf:Description>
      <dc:title>The Cruise of the Pequod</dc:title>
      <dc:creator>Melville, Herman</dc:creator>
      <dc:subject>map</dc:subject>
      <dc:description>From Moby-Dick, ed. Andrew Delbanco and Tom Quirk
(New York: Penguin Books, 1992).</dc:description>
      <dc:publisher>Penguin Books</dc:publisher>
      <dc:contributor>Delbanco, Andrew; Quirk, Tom</dc:contributor>
      <dc:date>1992</dc:date>
      <dc:format>image</dc:format>
      <dc:identifier>urn:metamedia:Melville:img:mel000001.tif
      </dc:identifier>
   </rdf:Description>
 </rdf:RDF>
```

Figure 2-3: Example Dublin Core File

2.1.3 MPEG-7

The Moving Picture Experts Group (MPEG) started a project in 1996 to create a standard way of representing descriptions of audio-visual content, in order to facilitate ways of finding information in digital form [11]. The standard contains functionality that allows one to annotate a particular time segment of a video clip, for example, or to specify pixel regions of an image and associate textual descriptions with them. In our system, metadata about digital files are stored in this XML-based schema.

2.2 XSLT

XSLT (Extensible Stylesheet Language Transformation) allows us to convert XML content into presentational forms, such as HTML, or WML, or even other XML documents. It is a well-formed XML document itself, using template-matching code to carry out the desired transformation. XSLT serves as the basis for our presentation layer, at least for the web-based projects. (The presentation layer for other projects with a richer set of Graphical User Interface components will be implemented as a fat client application, most likely based in Java.)

The powerful combination of XML and XSLT allows us to exploit the advantage of having the multimedia content separate from its presentation. Since the XSLT transformations are defined in a separate file, the presentation can be changed multiple times, simply by writing a new stylesheet that corresponds to the new presentation, and replacing the old one with the new. Furthermore, this isolation allows a graphic designer to be able to work in parallel with a content developer working on marking up the content, effectively speeding up development time.

With respect to my thesis solution, XSLT stylesheets are used to transform documents in one schema to another schema. More details about these transformations will be given in Chapters 3 and 5.

2.2.1 Xalan

The specific implementation of the XSLT standard we are using is Xalan [12], from Apache. This particular XSL processor has functionality known as the Extension Mechanism that will prove extremely useful in the implementation of my thesis solution.

2.3 XPointer Specification

XPointer is shorthand for the XML Pointer Language. It provides a standard way of addressing into individual parts of an XML document [13]. The need for this specification arose from limitations in current URL's. For example, if a person wanted to reference a specific sentence in a particular paragraph of a document on a website, he

would have to insert anchors at those positions in the document. If he did not have write access to the file, which is almost always the case, then he would have to rely on the author to insert the anchors, which is inconvenient in most cases.

XPointers do not show up much in the base framework itself; however, it does comprise a significant portion of my thesis solution.

2.4 XUpdate Specification

The XUpdate Working Group of the XML:DB initiative developed XUpdate, or the XML Update Language. The group's mission is to provide standard and flexible facilities for modifying XML documents [14]. The current implementation of the standard, Lexus version 0.2.2, provides for the following modifications to an XML document:

- Insert – Inserting a new node (element, text, comment, etc.) before or after an existing node specified by an XPointer
- Append – Creating a new node and appending it as a child of a node specified by an XPointer
- Update – Updating the textual value of an element specified by an XPointer
- Remove – Removal of a node specified by an XPointer
- Rename – Changing the name of an element or attribute node specified by an XPointer

2.5 Scalable Vector Graphics (SVG)

SVG is a language for describing two-dimensional graphics in XML. It allows for three types of graphic objects: vector graphic shapes, such as lines and curves, images, and text [15]. Most of the images and text elements in the user interfaces are dynamically generated with SVG. As an example, thumbnailed versions of archival documents are generated with SVG, so that the smaller versions can be placed next to annotations that refer to them.

2.6 Apache Cocoon

We decided to use an XML/web publishing framework to reduce the amount of code we needed to write ourselves, and well as to take advantage of any performance enhancements it may provide. McLaughlin [16] suggests that a good framework has the following properties:

- Stable and portable - Ideally it would be a second-generation product that can serve clients on any platform.
- Integrates with other XML Tools and API's - Should at least support SAX and DOM API's and should not be tied to any particular implementation.
- Production Presence - Has to be used in some production-quality applications.

Besides satisfying the above criteria, we decided to use Apache's Cocoon as our web application framework for the following reasons:

- First, it caches the output of XSLT transformations, greatly enhancing performance. For example, Cocoon is able to cache generated thumbnailed images of archival documents in the file system of the server, so that subsequent requests for the image would be retrieved faster.
- It contains built-in browser detection code, so that we can specify different stylesheets to use with different browsers, e.g. Microsoft Internet Explorer and Netscape Navigator.
- Supports the notion of tag libraries to encapsulate logic into reusable tags.
- Supports the chaining of multiple stylesheets, so the output of one XSLT transformation can be passed through another stylesheet, and so on.

2.7 System Architecture

The framework contains a store for the different XML schemas. This database was originally a native XML database, but a colleague of mine has been working on a relational table-based approach for optimizing query times and index generation. At the time of this writing, the system does not have any authentication/access control mechanism; this will be added in future versions. The Cocoon framework comprises the

presentation layer, at least for projects that are web-based. The content is generated as Cocoon XSP's (Extensible Server Pages), and then translated to HTML using XSLT stylesheets. Figure 2-4 below pictorially outlines the architecture for the framework.

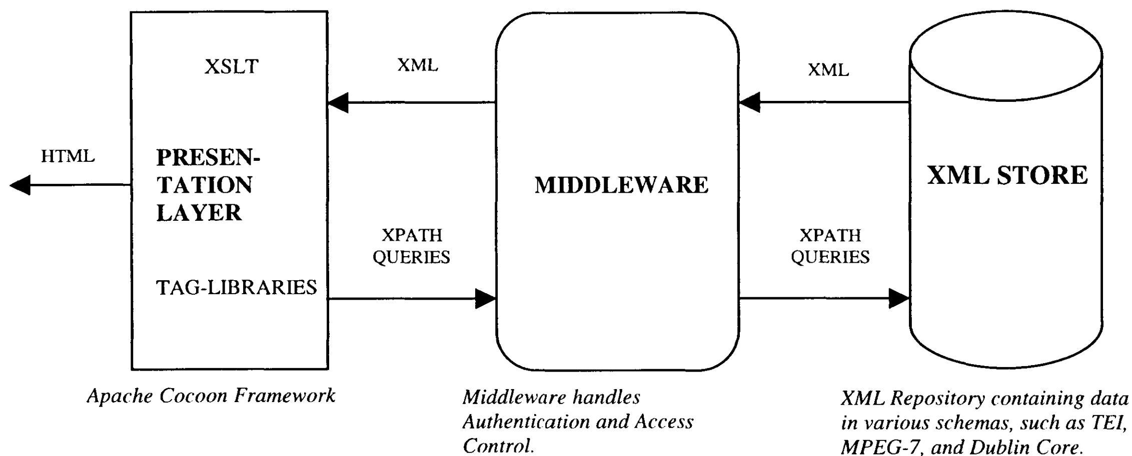


Figure 2-4: System Architecture

3 RESEARCH PROBLEM

In this chapter, I outline the problem with which this thesis is concerned. The problem can be broken down into two parts - that of querying over the different schemas in the backend store, and of updating and merging the data in the different schemas. I propose a preliminary solution to the first part in this section, which will be expanded upon in Chapter 5, and I defer the solution for the second part to that chapter.

3.1 Querying Over Multiple Schemas

From my discussion of the technologies used in the framework in the previous chapter, this part of the problem almost jumps out immediately. We have many different schema standards in the XML repository. Suppose a client of the system wanted to obtain all of the bibliographic information for the materials created by a particular author, whether they be images, videos, or text. How would he or she pose such a query to the system? Ideally, the client should not have to know about each of the schemas in the backend; he or she should only be concerned with one standard for representing bibliographic information, for example Dublin Core, and then pose queries in XPath over that schema, and receive documents back in that schema.

In order to bring about such functionality, other projects, such as the examples I give in the next chapter, construct an ontology for the most common elements of all the schemas, and then pose queries over that ontology. However, this project has the constraint that we want clients to only query over standard schemas, and not a representation that is Metamedia specific.

Thus, as the most direct solution, I proposed that we have multiple copies of the material in all the different schemas, grouped under their respective folders in the XML store. This strategy works because many of the schemas have elements that have corresponding mappings to elements in other schemas. For example, the Title element (the title of the bibliographic resource) in the Dublin Core specification maps to the CreationInformation.Creation.Title element in MPEG-7. Therefore, whenever a new

document is added to the system, a copy would need to be made in each of the schemas, with the corresponding fields mapped. In addition, clients would need to "register" that they are interested in sending queries and receiving results in a particular schema. Furthermore, the logic in the tag libraries would have to look up this registration in the database, and issue the clients' queries over the corresponding folder in the XML store.

While having these multiple copies in the database has the drawback of taking up extra storage space, this disadvantage is not significant – storage space today is very cheap and there are very good proprietary and open-source database implementations that handle large amounts of data. It is basically a tradeoff between time and space – whether we would want to take the time to dynamically generate these copies with every query request, or to simply generate these copies once when the document is added and use the extra space. In addition, we already know that clients of the framework would be interested in receiving XML from a query in only a limited number of schema representations. For example, anyone who is interested in bibliographic data about image, audio, or video files would probably be interested in only the Dublin Core schema[2]. Thus, each new document that is added to the system would not necessarily have to be mapped to all of the schemas. It would only need to be mapped to a few pertinent ones.

3.2 Updating/Merging Disparate Schemas

Now, the more interesting part of the problem. We have copies of documents in the repository in multiple schemas. Suppose there is a document 'A' in the repository that was originally created in the TEI schema, and there is a document 'B' in the DocBook schema, that is a mapped version of document A. A client who has registered to receive information in the DocBook schema comes along, and views document B (perhaps as the result of a query). He or she notices that some information in document B is wrong, say that a word in a chapter title is misspelled. The client (who has the access control rights

---

[2] A person interested in bibliographic data about textual files may be interested in the TEI and DocBook schemas, as well as the Dublin Core schema.

to do so) updates document B and makes the changes known in a document 'C'. How then do we incorporate the changes in document C into document A?

Clearly, we need to map document C (which is in the DocBook schema) back to document A; we will call this mapped version document D, and in the remainder of this paper, this document will be referred to as the 'derived' document. The problem then is given the original document A and the derived document D, find an algorithm that incorporates the changes in D into A (the update part), while keeping the elements in A that did not have mappings in B unchanged (the merge part). Figure 3-1 on the next page gives a pictorial description of the problem.

Note that the documents in Figure 3-1 do not show an example of the merge part of the problem, due to space limitations. However, if the original document in the TEI schema had bibliographic information in its `<teiHeader>` element, and if it was translated to Dublin Core, then most of the bibliographic information could have been mapped to Dublin Core, while the actual text in the TEI document (the article, book, etc. contained within the `<text>` element) would not have had mappings.

The update and merge problem is significantly complicated because the nodes (XML elements) in the original and derived documents are generally "keyless" – there is no unique identifier for the nodes that will aid in matching nodes in the original document to nodes in the derived document so we can update them. The next chapter looks at ways others in the field have approached this problem.

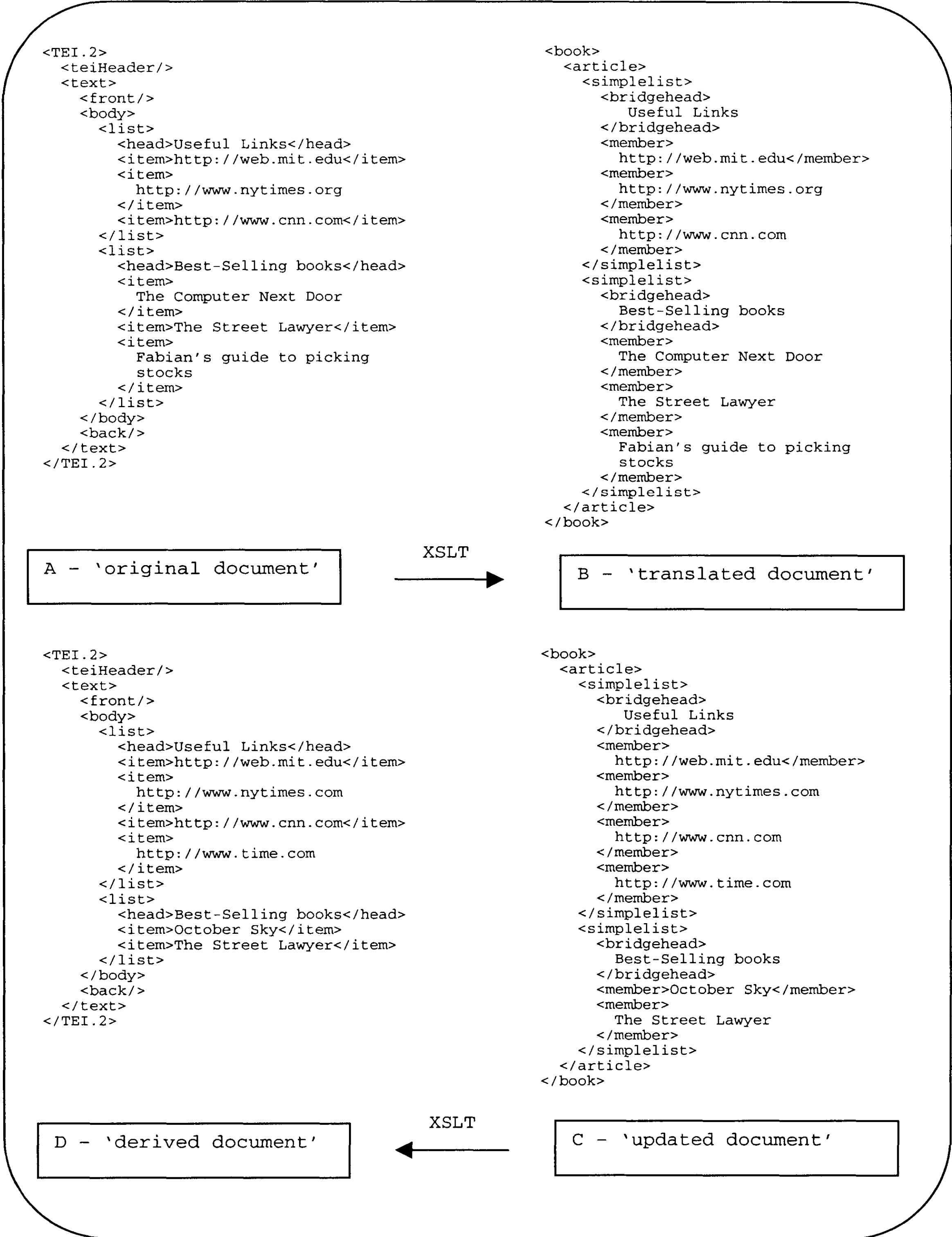


Figure 3-1: Pictorial Description of Update and Merge

4 RELATED WORK

In this chapter, I discuss some of the previous work done in the field that is related to this thesis. Although the works may not address exactly the same problem, I was able to use and expand upon some of the ideas in the papers.

4.1 Ontologies

As I briefly mentioned in the last chapter, many external projects design private ontologies for use in querying over heterogeneous XML data. Ahmedi, et. al. [17] designed an ontology for their data based on LDAP, the Lightweight Directory Access Protocol. They chose LDAP for two main reasons: 1) the LDAP model is very similar to the XML Document Object Model (DOM) and hence it is relatively easy to translate the data and 2) the LDAP model performs extremely well for the type of operations carried out on the Internet, that is, simple and fast read operations that occur more often than updates. In their architecture, the user poses queries over the ontology in LDAP, and an LDAP integration engine reformulates these queries in terms of the different XML schemas. Results are returned in XML from the document repository back to the integration engine, which then converts it back to LDAP for the user.

Hunter [18] developed the ABC ontology for use in integrating heterogeneous metadata descriptions. Their need for such interoperability followed from these scenarios: 1) to have one single search interface over the metadata descriptions, 2) to enable the integration and merging of different standards that may have overlapping fields, and 3) to enable views of the metadata based on user's interest or requirements. The team noticed that many entities and relationships, such as people, places, and events, occur across all of the domains, and the ABC ontology intends to model these intersections. They showed that this ontology provided a more scalable and cost-effective approach to integrating metadata descriptions over the one-to-one manually generated mappings that were done in the past.

Other approaches designed a language for querying over the ontology, and then issued this query to one or more "wrappers" that could convert the general query into the query language of the store it managed [19]. While these examples proved viable alternative ways of querying over heterogeneous data, they could not be implemented in the Metamedia framework because of the restriction that the query language and the schema queried over must be open standards.

4.2 Detecting Changes in Structured Documents

Chawathe, et. al [20] presents two algorithms for detecting changes in structured documents (such as XML data). The first addresses the "Good Matching Problem", that is, finding the nodes that correspond to each other from an original document to an updated document, when the document nodes are keyless. The second algorithm computes a "Minimum Cost Edit Script", given a set of matchings M, that generates a list of actions, such as insert node, delete node, and update node that would transform the original document into the updated document. Note the difference here: whereas their aim is to find the changes that would transform the original document into the updated one, mine is to incorporate the changes while leaving the remaining information unchanged. Nevertheless, much of their algorithm can be applied to this problem, with minor modifications that will be explained in greater detail in Chapter 5.

Chawathe, et. al [20] developed two criteria to determine if two nodes match. The first concerns leaf nodes or nodes that have no other nodes as children (they may have text as children). The criterion is that the label of the nodes must match, and that the value of the nodes (its textual content) must be "similar" enough by some predefined function "compare". The second criterion concerns interior nodes (nodes that are not leaf nodes) – the nodes must have the same label, and the number of children that match must be greater than or equal to half the maximum number of leaf children in either node.

The edit script takes in a set of matches and lists the sequence of actions that would convert the original document into the updated one. (The script also performs these actions as they are appended.) It is minimum cost because the algorithm finds the sequence that is computationally least expensive (there may be many ways of converting

one document to another). The algorithm comprises five different phases of generating the edit script. The first phase, the "Update Phase", adds an action to the script that updates the value of the original document node, if its matching node has a different value. The second phase, the "Align Phase", adds an action that rearranges the children of the original node if their matches in the updated document are in a different order. The "Insert Phase" then adds an action to insert a copy of node from the updated document into the original document if the node's parent has a match in the original document, but it itself does not have a match. Next, the "Move Phase" adds an action that moves nodes in the original document with a match in the updated document, but whose parents do not have matches. The nodes in the original document are moved such that its new parent has the correct match with the updated document. Finally, the "Delete Phase" adds an action to delete nodes in the original document that do not have a match in the updated document. At the end of this algorithm, the original document is said to be "isomorphic" to the updated document, meaning that they are identical except for node identifiers.

This algorithm will be modified appropriately to perform the required functionality for this thesis.

5 DESIGN AND IMPLEMENTATION

In this chapter, I present my solutions for the problems described in Chapter 3. I begin with the problem of querying over the heterogeneous schemas, and then address the update and merge problem.

5.1 Querying over Multiple Schemas

As stated in Chapter 3, I proposed to have multiple copies of the documents in the various schemas in the backend so that clients can receive documents in the representation that they want. In order to implement this, we write XSLT stylesheets that map from one schema to another. We also write stylesheets that perform the reverse mapping, as these stylesheets are needed in the update and merge part of the problem.

XSLT performs its transformations through the use of templates that match on a specific element or group of elements. It is extremely flexible, and handles all types of mappings that we need:

- *One-to-One Mappings*: For example, the TEI header `<author>` element maps directly to the Dublin Core `<creator>` element

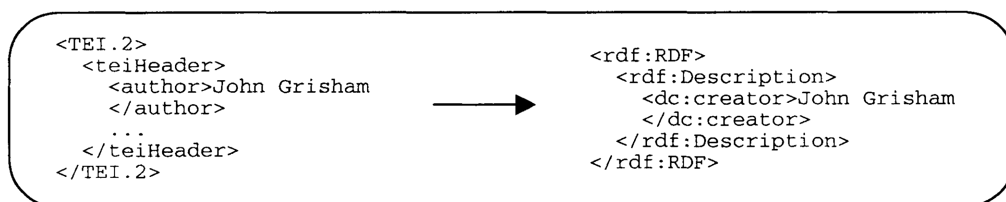


Figure 5-1: One-to-One Mappings

- *Compound-to-One Mappings*: For example, the DocBook `<publisher>/<publishername>` combination maps to the TEI header `<publisher>` element

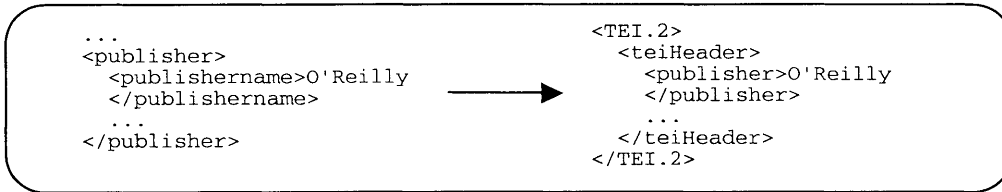


Figure 5-2: Compound-to-One Mappings

- *Compound-to-Compound Mappings*: For example, the combination of the TEI <list> element and its type attribute denotes the corresponding element matches in the DocBook schema

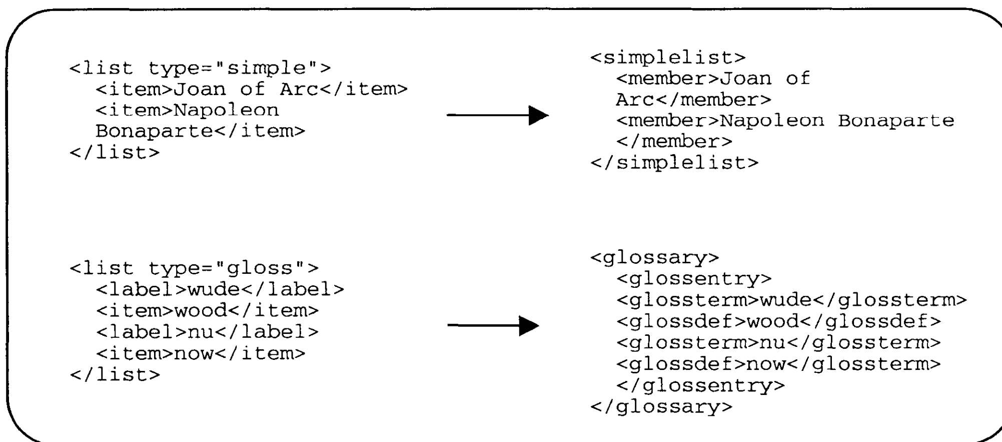


Figure 5-3: Compound-to-Compound Mappings

In order to query over the system in the different schemas, we must execute the following two steps whenever a document needs to be added to the repository:

1. Generate translated versions of the document using XSLT
2. Insert the original document and the copies into the database under their respective schema folders

5.2 Update and Merge

This part of the problem can be broken down further into two more parts: 1) a Matching Problem and 2) the actual merge/update given the set of matchings. As an overview of the algorithm, we want to generate XPointer matches between the input and output documents as they are updated or translated to a different schema. The matchings for documents that have been translated are generated via XSLT Extension Function calls,

while the matchings for updated documents are generated through a separate algorithm that will be discussed in Section 5.2.3. Finally, the algorithm takes these XPointer matches and merges in any changes with the original document. Figure 5-4 gives a pictorial outline of the algorithm:

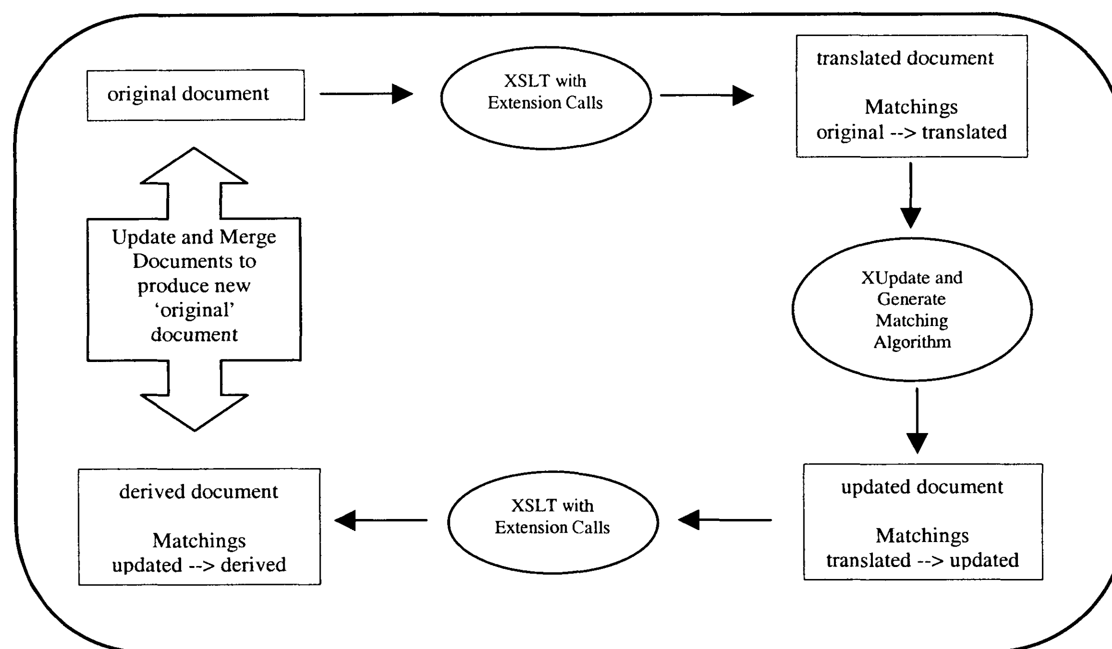


Figure 5-4: Pictorial Outline of Update and Merge Algorithm

5.2.1 Matching Problem: Introduction

As stated in Chapter 4, this problem concerns determining which nodes correspond to each other from an original source document to an updated document when the nodes are keyless (no unique identifiers). Trying to find the node matches blindly having just the original and updated documents is quite a complex problem, if not insoluble[3]. The matching criteria that Chawathe, et. al [20] developed seemed arbitrary and inapplicable to our more general domain. Thus, I proposed that we explicitly generate these matches as we are transforming the original document to another schema. These matches will be in the form of key/value pairs, where both the keys and values are XPointers. We will

[3] The reason it is so difficult is because one can never be completely sure that two nodes match, given only the documents to which they belong. For example, two nodes may have the same label, the same number of children, and the same values for the text elements of their children, but may not correspond to each other.

use the documents from Figure 3-1 in Chapter 3 as a running example of how the process works. Parts of the figure will be repeated in these sections as necessary.

At the high level, in order to generate node matches from the original document to the derived, we first want to generate matches from the original to the translated document, then generate matches from the translated document to the updated document, and finally generate matches from the updated document to the derived. After this process is finished, we would then have mappings from the original document to the derived document. The following subsections describe the algorithms used in each part of the process.

5.2.2 Matching Problem: Original to Translated

The original document that we will use for our running example is shown in Figure 5-5 below. It conforms to the TEI schema and contains two groups of lists:

```
<TEI.2>
  <teiHeader/>
  <text>
    <front/>
    <body>
      <list>
        <head>Useful Links</head>
        <item>http://web.mit.edu</item>
        <item>http://www.nytimes.org</item>
        <item>http://www.cnn.com</item>
      </list>
      <list>
        <head>Best-Selling books</head>
        <item>The Computer Next Door</item>
        <item>The Street Lawyer</item>
        <item>Fabian's guide to picking stocks</item>
      </list>
    </body>
    <back/>
  </text>
</TEI.2>
```

Figure 5-5: Running Example - Original Document

Its translated version, conforming to the DocBook schema, is shown in Figure 5-6.

```
<book>
  <article>
    <simplelist>
      <bridgehead>Useful Links</bridgehead>
      <member>http://web.mit.edu</member>
      <member>http://www.nytimes.org</member>
      <member>http://www.cnn.com</member>
    </simplelist>
    <simplelist>
      <bridgehead>Best-Selling books</bridgehead>
      <member>The Computer Next Door</member>
      <member>The Street Lawyer</member>
      <member>Fabian's guide to picking stocks</member>
    </simplelist>
  </article>
</book>
```

Figure 5-6: Running Example - Translated Document

How do we generate the node matches from the original document to the translated? Xalan, our XSLT processor, has an extension mechanism that allows a developer to instantiate a Java class and invoke methods on the instantiated object during the translation of a document. I have created such a class, called the MatchingManager to be used in the stylesheets. Its most important methods are the following:

1. `void addPathElement( String node )` - This method is used to keep track of the ancestral hierarchy of the translated nodes as the transformation process is taking place; it appends the currently mapped node to a variable that saves this state
2. `void createMatch( NodeList originalDocumentMatchList, String derivedDocumentElement )` - This method is the crux of the algorithm; it generates the XPointer matches for the original document and the translated document, modifying existing matches as necessary
3. `void removeCurrentPathElement()` - This method is used in conjunction with `addPathElement` to keep track of the ancestral hierarchy; it removes the element that was most recently added to the hierarchy list
4. `void serialize()` - This method serializes the generated matchings into persistent store, i.e. the database

The solution works as follows. The XSLT stylesheet that maps one schema to another first instantiates a MatchingManager object. Whenever a node in the original document is being mapped to a new node in the translated document, the `createMatch` method is

passed the ancestors of the element in the original document and the name of the mapped element:

```
<xsl:template match="list">
  <xsl:variable name="matching" select="matcher:createMatch( ancestor-or-
self::*, 'simplelist' )" />
  <simplelist>
    <xsl:variable name="path" select="matcher:addPathElement( 'simplelist'
)" />
    <xsl:apply-templates select="head | item"/>
  </simplelist>
  <xsl:variable name="path" select="matcher:removeCurrentPathElement()" />
</xsl:template>
```

Figure 5-7: XSLT Stylesheet with Extension Function Calls

The `createMatch` method concatenates the list of ancestors into a string that represents an XPointer. It then figures out what match to append to the current list based on an algorithm that I will describe below. The algorithm also modifies any existing matches if necessary. Finally, at the end of transformation templates, the stylesheet calls the MatchingManager's `serialize` method to store the generated matchings in the database.

The `createMatch` algorithm looks at the XPointer representation of the ancestors passed in as the argument and the current list of matches that it has stored (this list will be empty on the first invocation of the method). It then decides what to do based on four different cases:

- **Case 1:** We are adding a match for the first time, or there is no current match that is similar to this one.
  Example: `/TEI.2/body/list`

  We add the match to our list of matchings for the original document.

- **Case 2:** We are adding a match exactly like one in our current list of matchings.
  Example: Our list of matchings contains `/TEI.2/body/list` and `/TEI.2/body/list/item`. We want to add `/TEI.2/body/list/item`.

  We update the second match to be `/TEI.2/body/list/item[1]` and add the new match as `/TEI.2/body/list/item[2]`. We also add this match to a list of duplicates we have

already seen. Each element in the list of duplicates is a key/value pair, with the key being the match and the value being the number of times we have seen the duplicate. In this case, we would add a member to the duplicate list with key `/TEI.2/body/list/item` and value 2.

- **Case 3:** We are adding a match that is similar to one in our matchings list, and is also a member of our list of duplicates.
  Example: Our list of matchings contains `/TEI.2/body/list`, `/TEI.2/body/list/item[1]`, and `/TEI.2/body/list/item[2]`. We want to add `/TEI.2/body/list/item`.

  We leave the previous three matchings unchanged and add the new match as `/TEI.2/body/list/item[3]`. We also find the member of the duplicate list that has `/TEI.2/body/list/item` as its key, and increment its value by 1.

- **Case 4:** We are adding a match that is similar to one in our matchings list, and the parent (or some ancestor) of the element is a member of our duplicate list.
  Example: Our list of matchings contains `/TEI.2/body/list[1]`, `/TEI.2/body/list[1]/item[1]`, `/TEI.2/body/list[1]/item[2]`, and `/TEI.2/body/list[2]`. We want to add `/TEI.2/body/list/head`.

  In this case, before we add the new match, a member with the key `/TEI.body/list` would be in our list of duplicates with the value 2 (as per case 2). Since the new element we want to add has this key as its parent (or more generally as an ancestor), the value 2 must be incorporated into the match we add in our matchings list. Therefore, we leave the previous matchings unchanged and add the new match as `/TEI.2/body/list[2]/head`.

The complete algorithm is shown in the Figure 5-8 on the next page.

originalDocumentDuplicates <-- Ø; translatedDocumentDuplicates <-- Ø;
originalDocumentMatchings <-- Ø; translatedDocumentMatchings <-- Ø;

Algorithm createMatch(originalDocumentMatch, translatedDocumentMatch) {

// Handle Case 4

1. For each duplicateKey in originalDocumentDuplicates
   - a) If originalDocumentMatch **startsWith** duplicateKey and duplicateKey is not equal to originalDocumentMatch
     - i) pos <-- get value of duplicateKey entry in originalDocumentDuplicates
     - ii) originalDocumentMatch <-- originalDocumentMatch with "[" + pos + "]" inserted at position **length**(duplicateKey)

// Handle Case 3

2. If originalDocumentDuplicates **containsKey** originalDocumentMatch
   - a) i <-- get value of originalDocumentMatch entry in originalDocumentDuplicates
   - b) i = i + 1
   - c) update value of originalDocumentMatch entry to be i
   - d) append originalDocumentMatch + "[" + i + "]" to originalDocumentMatchings

// Handle Case 2

3. Else
   - a) If originalDocumentMatchings **contains** originalDocumentMatch
     - i) For each matching in originalDocumentMatchings
       - A) If matching **startsWith** originalDocumentMatch
         - 1) matching <-- insert "[1]" at **length**(matching)
         - 2) update matching in originalDocumentMatchings to be new value of matching
         - 3) remove all entries in originalDocumentDuplicates that have keys that **startWith** originalDocumentMatch
         - 4) add new entry to originalDocumentDuplicates with key originalDocumentMatch and value 2.
     - ii) append originalDocumentMatch + "[2]" to originalDocumentMatchings
   - b) Else
     - *// Handle Case 1*
     - i) append originalDocumentMatch to originalDocumentMatchings

4. Repeat steps 1-3 replacing originalDocument* with translatedDocument*.

}

Figure 5-8: Algorithm createMatch

Note that the words in bold are functions that are presumed to be present. The **startsWith** function checks to see if a string starts with a certain sequence of characters. The **length** method returns the length of a string. The **containsKey** method checks to see if a list has a key/value pair with a particular key, while the **contains** method checks to see if a list has a particular element in it.

In the body of the algorithm, originalDocumentMatchings refers to the list that contains the node matches for the original document, while translatedDocumentMatchings refers

to the list of matches for the translated document. By virtue of the algorithm, both of these lists will be the same size, and an XPointer at position i in originalDocumentMatchings matches an XPointer at the same position in translatedDocumentMatchings. See the Appendix for an implementation of this algorithm in Java.

Before the XSLT transformation is complete, the stylesheet calls the `serialize()` method, which inserts the matches in originalDocumentMatchings and translatedDocumentMatchings into the database. At the end of the transformation, the Matchings table for our running example will look like the following:

ORIGINAL	*TRANSLATED*	*UPDATED*	*DERIVED*
/TEI.2/text/body/list[1]	/book/article/simplelist[1]		
/TEI.2/text/body/list[1]/head	/book/article/simplelist[1]/bridgehead		
/TEI.2/text/body/list[1]/item[1]	/book/article/simplelist[1]/member[1]		
/TEI.2/text/body/list[1]/item[2]	/book/article/simplelist[1]/member[2]		
/TEI.2/text/body/list[1]/item[3]	/book/article/simplelist[1]/member[3]		
/TEI.2/text/body/list[2]	/book/article/simplelist[2]		
/TEI.2/text/body/list[2]/head	/book/article/simplelist[2]/bridgehead		
/TEI.2/text/body/list[2]/item[1]	/book/article/simplelist[2]/member[1]		
/TEI.2/text/body/list[2]/item[2]	/book/article/simplelist[2]/member[2]		
/TEI.2/text/body/list[2]/item[3]	/book/article/simplelist[2]/member[3]		

Table 5-1: Matchings table after the original document has been transformed to the translated one.

5.2.3 Matching Problem: Translated to Updated

Now we wish to update the translated version of the document. Again, such a case may arise when a client of the system obtains the translated document as the result of a query. He or she may want to change the document in some way, and if the person has the requisite access control rights, the system will allow him or her to make them.

In our running example, we would like to make the following changes to the translated document:

- Correct the New York times link to be http://www.nytimes.com
- Add another useful link - http://www.time.com

- Remove the bogus best-selling books, "The Computer Next Door" and "Fabian's Guide to Picking Stocks".
- Add another best-selling book, "October Sky" before the book "The Street Lawyer".

The XUpdate query that makes these changes is the following:

```
<?xml version="1.0"?>
<xupdate:modifications version="1.0"
xmlns:xupdate="http://www.xmldb.org/xupdate">
  <xupdate:update select="/book/article/simplelist[1]/member[2]">
    http://www.nytimes.com
  </xupdate:update>

  <xupdate:insert-after select="/book/article/simplelist[1]/member[3]">
    <xupdate:element name="member">
      http://www.time.com
    </xupdate:element>
  </xupdate:insert-after>

  <xupdate:remove select="/book/article/simplelist[2]/member[1]" />

  <xupdate:remove select="/book/article/simplelist[2]/member[3]" />

  <xupdate:insert-before select="/book/article/simplelist[2[/member">
    <xupdate:element name="member">
      October Sky
    </xupdate:element>
  </xupdate:insert-before>
</xupdate:modifications>
```

Figure 5-9: XUpdate Query for the Running Example

The updated document that results is shown in Figure 5-10:

```
<book>
  <article>
    <simplelist>
      <bridgehead>Useful Links</bridgehead>
      <member>http://web.mit.edu</member>
      <member>http://www.nytimes.com</member>
      <member>http://www.cnn.com</member>
      <member>http://www.time.com</member>
    </simplelist>
    <simplelist>
      <bridgehead>Best-Selling books</bridgehead>
      <member>October Sky</member>
      <member>The Street Lawyer</member>
    </simplelist>
  </article>
</book>
```

Figure 5-10: Running Example - Updated Document

The algorithm that generates the matches from the translated to the updated document, call it `generateUpdateMatchings`, must parse through the XUpdate query file. Note that the current implementation of my algorithm does not support the `xupdate:append` nor the

`xupdate:rename` elements. I now discuss what the algorithm should do when it encounters each of the different elements:

- `xupdate:remove` - There are four cases to handle.

  Case 1: The element we want to remove does not have any position information. Then, we set the match for this XPointer in the updated element to the empty string. *Example*: The element we want to remove is `/book/article`. Its match in the updated document should be the empty string (denoting that it has been deleted).

TRANSLATED	*UPDATED*
/book/article	

Table 5-2: Matches for XUpdate:Remove - Case 1

  Case 2: The element we want to remove is one of two sibling elements with the same label and parent. Then, the algorithm should update the match in the translated document to be the empty string, and update the match of the other element to not have any position information.
  Example: Elements `/book/article/simplelist[1]` and `/book/article/simplelist[2]` are in the translated document. Suppose we want to remove the element `/book/article/simplelist[1]`. The algorithm should match `/book/article/simplelist[1]` to the empty string in the updated document, and it should match `/book/article/simplelist[2]` in the translated document to `/book/article/simplelist` (no position information) in the updated document.

TRANSLATED	*UPDATED*
/book/article/simplelist[1]	
/book/article/simplelist[2]	/book/article/simplelist

Table 5-3: Matches for XUpdate:Remove - Case 2

  Case 3: The element we want to remove has sibling elements with higher position numbers. In this case, the algorithm should match all sibling elements with position numbers lower than the one we wish to delete to themselves, and match the other sibling elements to elements with the position number decremented by 1.
  Example: Elements `/book/article/simplelist/member[1]`, `/book/article/simplelist/member[2]`, `/book/article/simplelist/member[3]`, and

`/book/article/simplelist/member[4]` are in the translated document. Suppose we want to remove the element `/book/article/simplelist/member[2]`. The algorithm should match `/book/article/simplelist/member[1]` to `/book/article/simplelist/member[1]`, `/book/article/simplelist/member[3]` to `/book/article/simplelist/member[2]` (decremented the position number), and `/book/article/simplelist/member[4]` to `/book/article/simplelist/member[3]` (again decremented the position number).

TRANSLATED	*UPDATED*
/book/article/simplelist/member[1]	/book/article/simplelist/member[1]
/book/article/simplelist/member[2]	
/book/article/simplelist/member[3]	/book/article/simplelist/member[2]
/book/article/simplelist/member[4]	/book/article/simplelist/member[3]

Table 5-4: Matches for XUpdate:Remove - Case 3

Case 4: The element we want to remove is a sibling element with the highest position number. Then, we match this element in the translated document to the empty string in the updated document. All the other elements with the same label will have matches to themselves.

Example: Elements `/book/article/simplelist/member[1]`, `/book/article/simplelist/member[2]`, `/book/article/simplelist/member[3]` are in the translated document. We want to remove `/book/article/simplelist/member[3]`. The algorithm should match `/book/article/simplelist/member[3]` to the empty string, and match the other two to themselves.

TRANSLATED	*UPDATED*
/book/article/simplelist/member[1]	/book/article/simplelist/member[1]
/book/article/simplelist/member[2]	/book/article/simplelist/member[2]
/book/article/simplelist/member[3]	

Table 5-5: Matches for XUpdate:Remove - Case 4

- `xupdate:insert-after` - In both the insert-after and insert-before cases, we match the empty string in the translated document to an XPointer for the inserted node in the updated document. In order to generate the matchings, we must iterate over the siblings of the element we want to insert, looking for the element we want to insert the new one after. We call the element we want to insert the new one after the hinge

element. The iteration is done in order, from first child element to last. There are four cases to consider.

Case 1: As we search for the hinge element, we do not find any sibling elements with the same label as the one we want to insert (before the hinge element). We then count the number of elements after the hinge element with the same label as the one we want to insert. In this case, the count is 0, so the XPointer match for the inserted element will not have any position information.

Example: (These examples are clearer with snippets from the Dublin Core schema). Elements `/rdf:RDF/rdf:Description/dc:title`, `/rdf:RDF/rdf:Description/dc:publisher`, and `/rdf:RDF/rdf:Description/dc:subject` were originally in the translated document (in that order). The hinge element is `/rdf:RDF/rdf:Description/dc:publisher`, and we want to insert a `dc:identifier` element after it. The algorithm should add an XPointer representation for this element, namely `/rdf:RDF/rdf:Description/dc:identifier`, as the match in the updated document, and its corresponding match in the translated document should be the empty string (elements inserted in the updated document would not have a match in the translated document). All other elements should have matches to themselves.

TRANSLATED	*UPDATED*
/rdf:RDF/rdf:Description/dc:title	/rdf:RDF/rdf:Description/dc:title
/rdf:RDF/rdf:Description/dc:publisher	/rdf:RDF/rdf:Description/dc:publisher
/rdf:RDF/rdf:Description/dc:subject	/rdf:RDF/rdf:Description/dc:subject
	/rdf:RDF/rdf:Description/dc:identifier

Table 5-6: Matches for XUpdate:Insert-After – Case 1

Case 2: Case 2 is identical to case 1, except that the number of elements after the hinge element with the same label as the one we want to insert is greater than 0. In this case, the XPointer match for the inserted element will have position equal to 1, and we increment the position information of the other elements by 1 to account for the newly inserted element.

Example: Elements `/rdf:RDF/rdf:Description/dc:title`,

/rdf:RDF/rdf:Description/dc:publisher, /rdf:RDF/rdf:Description/dc:identifier, /rdf:RDF/rdf:Description/dc:subject, and /rdf:RDF/rdf:Description/dc:identifier were originally in the translated document (in that order). The hinge element is again /rdf:RDF/rdf:Description/dc:publisher, and we would like to add a dc:identifier element after it. The algorithm should add /rdf:RDF/rdf:Description/dc:identifier[1] as the XPointer in the updated document that matches the empty string in the translated document. It should also match /rdf:RDF/rdf:Description/dc:identifier[1] in the translated document to /rdf:RDF/rdf:Description/dc:identifier[2] in the updated document, and /rdf:RDF/rdf:Description/dc:identifier[2] in the translated document to /rdf:RDF/rdf:Description/dc:identifier[3] in the updated document; the incremented position values account for the inserted node. All other elements should have matches to themselves.

TRANSLATED	***UPDATED***
/rdf:RDF/rdf:Description/dc:title	/rdf:RDF/rdf:Description/dc:title
/rdf:RDF/rdf:Description/dc:publisher	/rdf:RDF/rdf:Description/dc:publisher
/rdf:RDF/rdf:Description/dc:identifier[1]	/rdf:RDF/rdf:Description/dc:identifier[2]
/rdf:RDF/rdf:Description/dc:subject	/rdf:RDF/rdf:Description/dc:subject
/rdf:RDF/rdf:Description/dc:identifier[2]	/rdf:RDF/rdf:Description/dc:identifier[3]
	/rdf:RDF/rdf:Description/dc:identifier[1]

Table 5-7: Matches for XUpdate:Insert-After – Case 2

Case 3: As we search for the hinge element in this case, we find one or more elements with the same label as the one we want to insert (before the hinge element). We count the number of elements with the same label after the hinge element, and the count is 0. In this case, the XPointer match for the inserted element should have position number that is one greater than the number of similar elements we saw before the hinge element.

Example: Elements /rdf:RDF/rdf:Description/dc:title, /rdf:RDF/rdf:Description/dc:identifier, and /rdf:RDF/rdf:Description/dc:publisher were originally in the translated document (in that order). The hinge element is /rdf:RDF/rdf:Description/dc:publisher, and we would like to add another dc:identifier element after it. The algorithm should

add /rdf:RDF/rdf:Description/dc:identifier[2] as the XPointer in the updated document that matches the empty string in the translated document. In addition, it should match /rdf:RDF/rdf:Description/dc:identifier in the translated document to /rdf:RDF/rdf:Description/dc:identifier[1] in the updated document. All other elements should have matches to themselves.

TRANSLATED	*UPDATED*
/rdf:RDF/rdf:Description/dc:title	/rdf:RDF/rdf:Description/dc:title
/rdf:RDF/rdf:Description/dc:identifier	/rdf:RDF/rdf:Description/dc:identifier[1]
/rdf:RDF/rdf:Description/dc:publisher	/rdf:RDF/rdf:Description/dc:publisher
	/rdf:RDF/rdf:Description/dc:identifier[2]

Table 5-8: Matches for XUpdate:Insert-After – Case 3

Case 4: Case 4 is identical to case 3, except that the number of elements after the hinge element with the same label as the element we want to insert is greater than 0. In this case, the XPointer match for the inserted element should have position number that is one greater than the number of similar elements we saw before the hinge element. In addition, we increment the position information of the other similar elements by 1 to account for the newly inserted element.

Example: Elements /rdf:RDF/rdf:Description/dc:title, /rdf:RDF/rdf:Description/dc:identifier, /rdf:RDF/rdf:Description/dc:publisher, and /rdf:RDF/rdf:Description/dc:identifier were originally in the translated document (in that order). The hinge element is /rdf:RDF/rdf:Description/dc:publisher, and we would like to add another dc:identifier element after it. The algorithm should add /rdf:RDF/rdf:Description/dc:identifier[2] as the XPointer in the updated document that matches the empty string in the translated document. It should also match /rdf:RDF/rdf:Description/dc:identifier[1] in the translated document to /rdf:RDF/rdf:Description/dc:identifier[1] in the updated document, and /rdf:RDF/rdf:Description/dc:identifier[2] to /rdf:RDF/rdf:Description/dc:identifier[3]. All other elements should have matches to themselves.

TRANSLATED	*UPDATED*
/rdf:RDF/rdf:Description/dc:title	/rdf:RDF/rdf:Description/dc:title
/rdf:RDF/rdf:Description/dc:identifier[1]	/rdf:RDF/rdf:Description/dc:identifier[1]
/rdf:RDF/rdf:Description/dc:publisher	/rdf:RDF/rdf:Description/dc:publisher
/rdf:RDF/rdf:Description/dc:identifier[2]	/rdf:RDF/rdf:Description/dc:identifier[3]
	/rdf:RDF/rdf:Description/dc:identifier[2]

Table 5-9: Matches for XUpdate:Insert-After – Case 4

- `xupdate:insert-before` - We handle the insert-before element in a similar manner as the insert-after element. However, there are important differences. First, we iterate over the siblings of the node we want to insert in reverse order, from the last node to the first node. There are also slight differences in the way we handle the different cases of the insert-after element. These differences are described below.

  Case 1: This case is handled exactly the same way for insert-before (with the exception of iterating over the sibling nodes in reverse order).

  Case 2: This case is handled in a similar manner as the insert-after case, except that the match for the inserted element has position information that is one greater than the number of similar elements before the hinge element, and the position numbers of the elements before the hinge element are unchanged (as opposed to being incremented by one in the insert-after case).

  Case 3: The difference here is that the XPointer match for the inserted element has position equal to 1 (as opposed to being one greater than the number of similar elements we saw after the hinge element).

  Case 4: We want to have the same effect as we did for Case 4 in the insert-after case. But since we are iterating over the child nodes in reverse order, the actual implementation that produces the result will be slightly different.

The complete algorithm is shown in Figure 5-11 on the following page.

```
matchings <-- retrieve list of matchings from database for translated document;
updatedMatchings <-- Ø;

Algorithm generateUpdatedMatchings( ) {
1. Parse through XUpdate query file and check name of elements.
2. If element name is 'xupdate:remove'
        a) i <-- indexOf element you want to remove in matchings
        // Handle Case 1
        b) updatedMatchings( i ) = ""
        // Handle Case 3
        c) For each match in matchings
                i) If match has similar XPointer to element you want to remove and its position is
                greater than the position of element you want to remove
                        A) updatedMatch <-- decrement position of match by 1
                        B) i <-- indexOf match in matchings
                        C) updatedMatchings( i ) = updatedMatch
        // Handle Case 2
        d) If number of similar elements left after deletion is 1
                i) i <-- indexOf similar element left in matchings
                ii) updatedMatchings( i ) = XPointer of element you want to remove with no
                position information
3. Else if element name is 'xupdate:element' and previous element name encountered was
'xupdate:insert-after'
        a) elementXPointer <-- XPointer of element you want to insert
        b) Iterate over nodes of translated document that are siblings of the node we want to insert.
                i) countBefore <-- number of elements before hinge element with the same label
                as node you want to insert.
                ii) countAfter <-- number of elements after hinge element with the same label as
                node you want to insert.
        // Handle Cases 1 and 3
        c) append elementXPointer + "[" + (countBefore + 1) + "]" to updatedMatchings
        // Handle Cases 2 and 4
        d) For each of the elements in the translated document with the same label as the element
        we want to insert and that have position numbers greater than (countBefore + 1)
                i) i <-- indexOf element in matchings
                ii) updatedMatch <-- increment position of element by 1
                Iii) updatedMatchings( i ) = updatedMatch
        e) For all other elements
                i) i <-- indexOf element in matchings
                ii) updatedMatch( i ) = element
4. Else if element name is 'xupdate:element' and previous element name encountered was
'xupdate:insert-before'
        a) Handle this case the same way as Step 3, except iterate over the nodes of the translated
        document in reverse order.
5. For the elements in updatedMatchings at indices < size( matchings ), add the XPointers in
matchings and updatedMatchings as matches in the database for the translated and updated
documents.
6. For all other elements in updatedMatchings, match "" to the XPointer in
updatedMatchings as a match in the database for the translated and updated documents.
}
```

Figure 5-11: Algorithm generateUpdateMatchings

The algorithm presumes a function **indexOf** is present. The function returns the index of an array that has a certain element at that position. The **size** method returns the number of elements in the array. See the appendix for an implementation of this algorithm in Java.

After the algorithm runs, the Matchings table will now look like the following:

ORIGINAL	*TRANSLATED*	*UPDATED*	*DERIVED*
/TEI.2/text/body/list[1]	/book/article/simplelist[1]	/book/article/simplelist[1]	
/TEI.2/text/body/list[1]/head	/book/article/simplelist[1]/bridgehead	/book/article/simplelist[1]/bridgehead	
/TEI.2/text/body/list[1]/item[1]	/book/article/simplelist[1]/member[1]	/book/article/simplelist[1]/member[1]	
/TEI.2/text/body/list[1]/item[2]	/book/article/simplelist[1]/member[2]	/book/article/simplelist[1]/member[2]	
/TEI.2/text/body/list[1]/item[3]	/book/article/simplelist[1]/member[3]	/book/article/simplelist[1]/member[3]	
/TEI.2/text/body/list[2]	/book/article/simplelist[2]	/book/article/simplelist[2]	
/TEI.2/text/body/list[2]/head	/book/article/simplelist[2]/bridgehead	/book/article/simplelist[2]/bridgehead	
/TEI.2/text/body/list[2]/item[1]	/book/article/simplelist[2]/member[1]		
/TEI.2/text/body/list[2]/item[2]	/book/article/simplelist[2]/member[2]	/book/article/simplelist[2]/member[2]	
/TEI.2/text/body/list[2]/item[3]	/book/article/simplelist[2]/member[3]		
		/book/article/simplelist[1]/member[4]	
		/book/article/simplelist[2]/member[1]	

Table 5-10: Matchings Table after the algorithm generateUpdateMatchings has run

We have inserted the matchings for the translated and updated documents. The rows with empty values in the Translated column and non-empty values in the Updated column correspond to elements that have been inserted, while the rows with empty values in the Updated column and non-empty values in the Translated column correspond to elements that have been deleted.

5.2.4 Matching Problem: Updated to Derived

Fortunately, given the tools we have so far, this case is very easy. All we need to do is transform the updated document back to the derived document, using the transformational process that we used previously when converting the original document to the translated document. The stylesheet itself will of course be different – it will now perform the inverse mapping for the updated to the derived document. In addition, the serialize method in this case would not simply insert the matches into the database, but update the already existing rows with the matches for the derived document.

After this algorithm runs, the complete Matchings table will look like the following:

ORIGINAL	TRANSLATED	UPDATED	DERIVED
/TEI.2/text/body/list[1]	/book/article/simplelist[1]	/book/article/simplelist[1]	/TEI.2/text/body/list[1]
/TEI.2/text/body/list[1]/head	/book/article/simplelist[1]/bridgehead	/book/article/simplelist[1]/bridgehead	/TEI.2/text/body/list[1]/head
/TEI.2/text/body/list[1]/item[1]	/book/article/simplelist[1]/member[1]	/book/article/simplelist[1]/member[1]	/TEI.2/text/body/list[1]/item[1]
/TEI.2/text/body/list[1]/item[2]	/book/article/simplelist[1]/member[2]	/book/article/simplelist[1]/member[2]	/TEI.2/text/body/list[1]/item[2]
/TEI.2/text/body/list[1]/item[3]	/book/article/simplelist[1]/member[3]	/book/article/simplelist[1]/member[3]	/TEI.2/text/body/list[1]/item[3]
/TEI.2/text/body/list[2]	/book/article/simplelist[2]	/book/article/simplelist[2]	/TEI.2/text/body/list[2]
/TEI.2/text/body/list[2]/head	/book/article/simplelist[2]/bridgehead	/book/article/simplelist[2]/bridgehead	/TEI.2/text/body/list[2]/head
/TEI.2/text/body/list[2]/item[1]	/book/article/simplelist[2]/member[1]		
/TEI.2/text/body/list[2]/item[2]	/book/article/simplelist[2]/member[2]	/book/article/simplelist[2]/member[2]	/TEI.2/text/body/list[2]/item[2]
/TEI.2/text/body/list[2]/item[3]	/book/article/simplelist[2]/member[3]		
		/book/article/simplelist[1]/member[4]	/TEI.2/text/body/list[1]/item[4]
		/book/article/simplelist[2]/member[1]	/TEI.2/text/body/list[2]/item[1]

Table 5-11: Complete Matchings Table

Note that in this example, the derived document happens to look like the document we want to replace the original one with, because no merges are necessary.

We therefore reached our goal of having explicit matchings from the original document to the derived one. We now look at an algorithm that takes the matches in the Original and Derived columns and performs an update and merge operation.

5.2.5 Update and Merge Algorithm

Given the matchings that were generated by the previous algorithms, the Update and Merge Algorithm is straightforward. We do not want to touch any of the nodes in the original document that do not have XPointer representations in the Matchings table. This is essentially the merge part, because the original document most likely contains information that is absent from the derived document. Next, for elements in the original document that have matches to themselves in the derived document (for instance `/TEI.2/text/body/list[1]/item[1]` in our running example), we update the values of the attributes in the original document to be equal to the attribute values of the corresponding node in the derived document, and we also update the content of the textual child nodes. This makes sense, since the derived document would always have the most up-to-date attribute and textual values. Continuing, we want to delete the nodes in the original

document whose XPointer representations in the Matchings table match the empty string in the Derived column; one such example is `/TEI.2/text/body/list[2]/item[1]`. Finally, for the entries in the Derived column of the Matchings table that match the empty string in the original, we want to get the node out of the derived document that corresponds to that XPointer representation, and insert it at the specified position in the original document; one such example is `/TEI.2/text/body/list[1]/item[4]`. It does not matter that we modify the derived document, because it will be discarded anyway. After the algorithms in this chapter have been run, the original document would have been changed so that it incorporates the updates. The old original, translated, updated, and derived documents and their matchings in the table will be discarded. The "new" original document will take the place of the old one, a translated copy of this document will be made (along with the matchings), and the process begins again when a user wants to make another update.

There is one crucial point that should be mentioned with regards to the order in which the algorithm handles the different cases – all updates must be handled before any inserts or deletes. Otherwise, we may insert a node at the correct position, but its textual content may be incorrectly replaced by the handling of a subsequent update. For our database implementation, this restriction means that we have to add an "ORDER BY Original" clause to the SQL query that retrieves the matchings, so that the matches with an empty string in the Original column show up at the bottom of the result set. This clause suffices since the algorithm handles all deletes after all inserts and updates have been completed.

The algorithm, called `updateAndMerge`, is shown in Figure 5-12 on the next page.

```
matchingPairs <-- retrieve matchings from Original and Derived columns of Matchings table,
sorted by the Original column;

Algorithm updateAndMerge( originalDoc, derivedDoc, matchingPairs ) {
1. For each pair in matchingPairs
        a) If key( pair ) is not empty and value( pair ) is not empty
                i) updateNode( getNode( originalDoc, key( pair ) ),
                               getNode( derivedDoc, value( pair ) ) )
        b) Else if key( pair ) is not empty and value( pair ) is empty
                i) markNodeToDelete( getNode( originalDoc, key( pair ) ) )
        c) Else if key( pair ) is empty and value( pair ) is not empty
                i) insertNode( originalDoc, getNode( derivedDoc, value( pair ) ) )
2. Do a post-order traversal of originalDoc and remove nodes that were marked to be deleted.
}
```

Figure 5-12: Algorithm updateAndMerge

The **key** and **value** methods return the key and value of the pairs returned from the database, respectively; the key would be the XPointer for a node in the original document, while the **value** would be the XPointer for a node in the derived document. The **getNode** method takes in an XML document and an XPointer as arguments, and returns the actual node object from the document that the XPointer represents. The **updateNode** method takes in the matching nodes from the original and derived documents, and updates the value of the node in the original document as explained above. The **markNodeToDelete** method modifies the node in the original document in some manner so that during the post-order traversal, the code knows that this node should be deleted. (For example, in the implementation of this algorithm found in the appendix, the code creates an attribute called "metamediaDeleteElement" and sets its value to "true"; thus, during the post-order traversal, any node with this particular attribute gets deleted). Finally, the **insertNode** method gets the node from the derived document (passed in as the second argument) and inserts it at the position in the original document, based on the XPointer.

6 ANALYSIS

We now have a solution for the problems posed in chapter 3. This chapter analyzes that solution and compares it to other ways people have approached similar problems. It ends with a discussion of the limitations of the algorithms.

6.1 Comparison with Other Methods

As mentioned in chapter 3, other groups of researchers defined their own ontologies for a particular domain, which was as generic and flexible as possible, then mapped specific schemas to that ontology. Whereas that worked well in their cases, we could not solve the heterogeneous integration problem in this manner because of the restriction that we wanted users of the system to query over fields that were part of some standard.

If we had used this method to solve the problem, it would have provided a number of advantages:

1. *Space Allocation*: Because this solution would have entailed performing dynamic translations of the schemas to the ontology, there would not have been a need to keep replicas of the files in different schemas.
2. *Transactionality*: With the ontological approach, any transactional updates to the ontology would have given us transactionality for the translated copies for free. The reason is that there would only be one copy of each document represented in the ontology we defined, so if this document was updated in a transactional context, any subsequent read via a translation to a different schema would have returned the document with the updates included. In my scheme, the developer must ensure that the updates to all the translated copies happen in the same transactional context as an update to any document. Otherwise, a subsequent query to a translated copy of an updated document may result in a "dirty read."

However, the ontological approach would have also had one main disadvantage:

1. *Dynamic Translation Time*: Because there would not be replicated copies of the files in different schemas, the users would have to wait for the time it takes the XSLT stylesheets to map the schemas dynamically, in response to their queries.

6.2 Limitations of Algorithms

This subsection describes known limitations with the algorithms as presented in chapter 5:

- *Non-existent Inverse Mapping*: There may be cases where it is not possible to write an XSLT stylesheet that translates the updated document back to the derived document. In these cases, my solution to the problem would be inapplicable. One such example is if the concatenation of three elements in the original document map to one element in the translated document (see Figure 6-1):

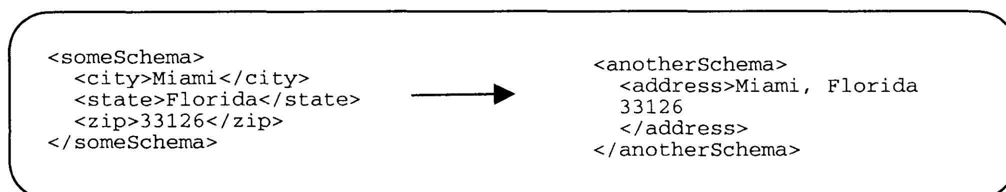


Figure 6-1: Example of Non-existent Inverse Mapping

From the snippets in Figure 6-1, it would be impossible to write a stylesheet that translates the document in "anotherSchema" back to "someSchema" – any such stylesheet would not know how to break the string in the address element back to three separate city, state, and zip elements. For example, if the string "Florida 33126" was present in the address element, then the inverse mapping stylesheet would not know whether "Florida" referred to a city, or a state. This issue is common to most data integration applications.

- *Schemas Dependent on Ordering of Nodes*: The solution handles only certain cases where the ordering of nodes in a schema is important. In the TEI schema, for example, the `idno` element must come directly after the `publisher` element (as children of the `publicationStmt` element). In my current implementation, if at least one `idno` element existed in the original TEI document, and the user added one or more identifiers in the updated document, then the update and merge algorithm would

correctly add the newly inserted elements after the original one. However, if no `idno` element originally existed and the user added one or more identifiers in the updated document, then the algorithm would by default add the inserted elements to the end of the current list as it was updating the document. The resultant merged document may or not comply with the ordering restrictions in the TEI schema.

- *Unchecked Schema Compliance*: The solution, as presented, does not check whether the updated document that the user created conforms to any schema. This means that the user can update the translated document in any way that results in a valid XML document and the changes would be made. However, adding such checks should not be too difficult; they were not added here mainly due to time restrictions.

7 CONCLUSION AND FUTURE WORK

In conclusion, this document presented a solution for querying over and updating heterogeneous schemas in a content management system. The solution to the heterogeneous schema integration problem involved the use of XSLT stylesheets to map one schema to another; the database stored the replicas under their respective folders. The update and merge algorithms consisted of first generating explicit XPointer matches from the original document to the derived document, then using these matches to carry out the updating.

Future development of these algorithms need to handle the limitations outlined in the previous chapter, namely, taking the ordering of nodes into consideration when updating the derived document and making sure any updates to the translated document are validated against the respective DTD. The Inverse Mapping limitation is typical of most data integration problems, and cannot be solved until advanced Artificial Intelligence techniques allow programs to understand semantics in text.

8 REFERENCES

1. Transforming Humanities Education [WWW Document] URL http://web.mit.edu/cms/Research/transform.html (22 May 2002)

2. Metamedia [WWW Document] URL http://metamedia.mit.edu (22 May 2002)

3. The Perseus Digital Library [WWW Document] URL http://www.perseus.tufts.edu/ (22 May 2002)

4. Berliner sehen [WWW Document] URL http://web.mit.edu/afs/athena.mit.edu/org/f/fll/www/projects/BerlinerSehen.html (22 May 2002)

5. Shakespeare Electronic Archive [WWW Document] URL http://htf-puppy.mit.edu/research/shakespeare/index.html (22 May 2002)

6. Walsh, Norman. What is XML? [WWW Document] URL http://www.xml.com/pub/a/98/10/guide1.html#AEN58 (22 May 2002)

7. Text Encoding Initiative [WWW Document] URL http://www.tei-c.org/ (22 May 2002)

8. Walsh, Norman and Leonard Muellner. DocBook: The Definitive Guide [WWW Document] URL http://www.oreilly.com/catalog/docbook/chapter/book/docbook.html (22 May 2002)

9. Dublin Core Metadata Initiative [WWW Document] URL http://www.dublincore.org (22 May 2002)

10. What is RDF? [WWW Document] URL http://www.xml.com/pub/a/2001/01/24/rdf.html (22 May 2002)

11. Martínez, José M. Overview of the MPEG-7 Standard [WWW Document] URL http://mpeg.telecomitalialab.com/standards/mpeg-7/mpeg-7.htm (22 May 2002)

12. Xalan-Java version 2.3.1 [WWW Document] URL http://xml.apache.org/xalan-j/index.html (22 May 2002)

13. Chapter 20 of the *XML Bible, Second Edition: XPointers* [WWW Document] URL http://www.ibiblio.org/xml/books/bible2/chapters/ch20.html (22 May 2002)

14. XUpdate – XML Update Language [WWW Document] URL http://www.xmldb.org/xupdate/ (22 May 2002)

15. Eisenberg, J. David. An Introduction to Scalable Vector Graphics [WWW Document] URL http://www.xml.com/pub/a/2001/03/21/svg.html (22 May 2002)

16. McLaughlin, Brett. Web Publishing Frameworks [WWW Document] URL http://www.oreilly.com/catalog/javaxml/chapter/ch09.html (22 May 2002)

17. Ahmedi, Lule, Pedro José Marrón, and Georg Lausen. Ontology-based Access to Heterogeneous XML Data *Int. Workshop on Web Databases*, Sept. 26-28, 2001, Vienna, Austria.

18. Hunter, Jane. MetaNet – A Metadata Term Thesaurus to Enable Semantic Interoperability Between Metadata Domains *Journal of Digital Information, Special Issue on Networked Knowledge Organization Systems, Volume 1, Issue 8*, April 2001.

19. Ullman, Jeffrey D. Information Integration Using Logical Views *6th International Conference on Database Theory, LNCS 1186*, 1997.

20. Chawathe, Sudarshan S, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change Detection in Hierarchically Structured Information *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 493-504, Montreal, Quebec, June 1996.

9 APPENDIX

9.1 TranslateSchema.java

This file translates an XML document from one schema to another.

```
package edu.mit.metamedia.util;

import javax.xml.transform.TransformerFactory;
import javax.xml.transform.Transformer;
import javax.xml.transform.Source;
import javax.xml.transform.stream.StreamSource;
import javax.xml.transform.stream.StreamResult;
import java.io.*;

public class TranslateSchema {

   public static void main( String[] args ) {
                        try {
                           TransformerFactory tFactory = TransformerFactory.newInstance();
                           // Get the XML input document and the stylesheet.
                           Source xmlSource = new StreamSource(
                                                   new File(
"/nfshome/fabian/Backup/MEng_Thesis/implementation/xml/MEng_tei_original.xml") );
                           Source xslSource = new StreamSource(
                                                   new File( "/nfshome/fabian/Backup/MEng_Thesis/xsl-translations/tei_to_DC.xsl" )
);

                           Transformer transformer = tFactory.newTransformer( xslSource );
                           // Perform the transformation, sending the output to a file
                           transformer.transform( xmlSource, new StreamResult(
                                                   new File(
"/nfshome/fabian/Backup/MEng_Thesis/implementation/xml/MEng_dc_translated.xml")) );
                        }
                        catch ( Exception ex ) {
                           ex.printStackTrace();
                        }
   }
}
```

9.2 MatchingManager.java

This file contains a Java implementation of the createMatch algorithm of Figure 5-8 (pg. 33).

```
package edu.mit.metamedia.util;

import java.util.*;
import org.w3c.dom.*;
import java.io.*;
import java.sql.*;

public class MatchingManager {
   Hashtable originalDocumentDuplicates;
   Hashtable derivedDocumentDuplicates;
   Vector originalDocumentMatchings;
   Vector derivedDocumentMatchings;
   StringBuffer currentDerivedDocumentPath;

   public MatchingManager() {
          // Note: It is implicit that each element of originalDocumentMatchings maps
          // to the corresponding element of derivedDocumentMatchings
          originalDocumentDuplicates = new Hashtable();
          derivedDocumentDuplicates = new Hashtable();
```

```
        originalDocumentMatchings = new Vector();
        derivedDocumentMatchings = new Vector();

        currentDerivedDocumentPath = new StringBuffer( "" );
}

public void addPathElement( String str ) {
        currentDerivedDocumentPath = currentDerivedDocumentPath.append( "/" + str );
}

public void createMatch( NodeList originalDocumentMatchList,
                                String derivedDocumentElement ) {
        // Make a String object representing the XPath out of the NodeList
        String originalDocumentMatch = "";
        Node node = null;
        for ( int i = 0; i<originalDocumentMatchList.getLength(); i++ ) {
           node = originalDocumentMatchList.item( i );
           originalDocumentMatch = originalDocumentMatch + "/" + node.getNodeName();
        }

        createMatchHelper( originalDocumentMatch, originalDocumentDuplicates,
                              originalDocumentMatchings );

        String derivedDocumentMatch = null;
        if ( derivedDocumentElement.startsWith ( "../" ) ) {
           String tmpStr = currentDerivedDocumentPath.toString();
           int i = tmpStr.lastIndexOf( "/" );
           // Important: Must make copy of currentDerivedDocumentPath first
           StringBuffer buf = new StringBuffer( currentDerivedDocumentPath.toString() );
           buf.replace( i, buf.length(), "" );
           buf.append( "/" + derivedDocumentElement.substring( 3, derivedDocumentElement.length() ) );
           derivedDocumentMatch = buf.toString();
        }
        else {
           derivedDocumentMatch = currentDerivedDocumentPath.toString().concat(
                                                  "/" + derivedDocumentElement );
        }

        createMatchHelper( derivedDocumentMatch, derivedDocumentDuplicates,
                              derivedDocumentMatchings );
}

public void createMatchHelper( String match, Hashtable duplicates,
                                       Vector matchings ) {

        // Check first to see if there is a duplicate that starts with this
        // match
        String tmpDuplicate = null;
        for ( Enumeration e = duplicates.keys(); e.hasMoreElements(); ) {
           tmpDuplicate = (String)e.nextElement();
           StringBuffer buf = null;
           if ( match.startsWith( tmpDuplicate ) && !(match.equals( tmpDuplicate ) )) {
                // Update the matching to include position information.  Implement
                // this via mutable StringBuffer object
                buf = new StringBuffer( match );
                int pos = ((Integer)duplicates.get( tmpDuplicate )).intValue();
                buf.insert( tmpDuplicate.length(), "[" +
                                String.valueOf( pos ) + "]" );
                // Update the old value of the match variable to be the new one
                // with the position information
                match = buf.toString();
           }
        }
        if ( duplicates.containsKey( match ) ) {
           Integer integer = (Integer)duplicates.get( match );
           int i = integer.intValue();
           i++;
           duplicates.put( match, new Integer( i ) );
           matchings.addElement( match + "[" + String.valueOf( i ) +
                                        "]" );
```

```
        }
        else {
          boolean alreadyAddedMatch = matchings.contains( match );
          if ( alreadyAddedMatch ) {
                String tmpMatch = null;
                StringBuffer buf = null;
                for ( int i = 0; i<matchings.size(); i++ ) {
                  tmpMatch = (String)matchings.elementAt( i );
                  if ( tmpMatch.startsWith( match ) ) {
                        // Update the matching to include position information.  Implement
                        // this via mutable StringBuffer object
                        buf = new StringBuffer( tmpMatch );
                        buf.insert( match.length(), "[1]" );
                        matchings.setElementAt( buf.toString(), i );

                        // Remove original duplicates that start with the match
                        String tmpString = null;
                        for ( Enumeration e = duplicates.keys(); e.hasMoreElements(); ) {
                          tmpString = (String)e.nextElement();
                          if ( tmpString.startsWith( match ) ) {
                                duplicates.remove( tmpString );
                          }
                        }
                        // Add new duplicate to list
                        duplicates.put( match, new Integer( 2 ) );
                  }
                }
                matchings.addElement( match + "[2]" );
          }
          else { // We haven't added this match yet
                matchings.addElement( match );
          }
        }
}

public void removeCurrentPathElement() {
        String tmpStr = currentDerivedDocumentPath.toString();
        int i = tmpStr.lastIndexOf( "/" );
        currentDerivedDocumentPath = currentDerivedDocumentPath.replace( i,
                currentDerivedDocumentPath.length(), "" );
}

public void serialize( org.apache.xalan.extensions.XSLProcessorContext context,
                                org.apache.xalan.templates.ElemExtensionCall extElem ) {
        try {
          // Connect to database and store Matchings data
          Class.forName( "org.postgresql.Driver" );
          String user = "xxxxxx";
          String pass = "xxxxxx";
          String url = "xxxxxxxxxx";
          Connection con = DriverManager.getConnection( url, user, pass );
          String sqlstr = "INSERT INTO Matchings ( original, translated ) VALUES (?, ?)";
          PreparedStatement stat = con.prepareStatement( sqlstr );
          // Iterate over matchings Vector and perform insertions
          for ( int i = 0; i<originalDocumentMatchings.size(); i++ ) {
                stat.setString( 1, (String)originalDocumentMatchings.elementAt( i ) );
                stat.setString( 2, (String)derivedDocumentMatchings.elementAt( i ) );
                stat.executeUpdate();
          }
          stat.close();
          con.close();
        }
        catch ( Exception ex ) {
          ex.printStackTrace();
        }
}


public void serializeDerived( org.apache.xalan.extensions.XSLProcessorContext context,
                                        org.apache.xalan.templates.ElemExtensionCall extElem ) {
```

```
            try {
               // Connect to database and store Matchings data
               Class.forName( "org.postgresql.Driver" );
               String user = "xxxxxx";
               String pass = "xxxxxx";
               String url = "xxxxxxxxxx";
               Connection con = DriverManager.getConnection( url, user, pass );
               String sqlstr = "UPDATE Matchings SET derived = ? WHERE updated = ?";
               PreparedStatement stat = con.prepareStatement( sqlstr );
               // Iterate over matchings Vector and perform updates
               for ( int i = 0; i<originalDocumentMatchings.size(); i++ ) {
                         stat.setString( 2, (String)originalDocumentMatchings.elementAt( i ) );
                         stat.setString( 1, (String)derivedDocumentMatchings.elementAt( i ) );
                         stat.executeUpdate();
               }
               stat.close();
               con.close();
            }
            catch ( Exception ex ) {
               ex.printStackTrace();
            }
   }
}
```

9.3 XUpdateTool.java

This file is used in conjunction with XUpdateSAXHandler.java to provide an implementation of the generateUpdateMatchings algorithm of Figure 5-11 (pg. 42).

```
/**
 * Fabian F. Morgan
 * April 24-26, 2002
 **/
package edu.mit.metamedia.util;

import java.util.*;
import java.io.*;
import org.jdom.input.SAXBuilder;
import org.jdom.*;
import org.apache.oro.text.regex.*;
import org.jdom.adapters.*;

import org.w3c.dom.*;

import org.infozone.lexus.*;

import org.jdom.output.XMLOutputter;

import org.infozone.tools.xml.queries.XUpdateQuery;
import org.infozone.tools.xml.queries.XUpdateQueryFactory;

import org.apache.xerces.parsers.DOMParser;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.xml.sax.*;
import org.xml.sax.helpers.*;

import org.apache.xalan.serialize.*;
import org.apache.xalan.templates.OutputProperties;

import java.sql.*;

public class XUpdateTool {
   private static String vendorParserClass = "org.apache.xerces.parsers.SAXParser";  // The SAX parser to use

   public static void main( String[] args ) {
```

```
// Parse original and XUpdate documents into their respective DOM's
org.w3c.dom.Document originalDoc = null;
org.jdom.Document xupdateDoc = null;
String xupdateStr = null;
try {
  XercesDOMAdapter adapter = new XercesDOMAdapter();
  originalDoc = adapter.getDocument( new FileInputStream (
            new File ( "/nfshome/fabian/Backup/MEng_Thesis/implementation/xml/MEng_dc_translated.xml" ) ), false );

  SAXBuilder builder = new SAXBuilder( vendorParserClass );
  File xupdateDocFile = new File ( "/nfshome/fabian/Backup/MEng_Thesis/implementation/xupdate-xml/xupdate.xml" );
  xupdateDoc = builder.build( xupdateDocFile );

  XMLOutputter outputter = new XMLOutputter();
  outputter.setTextNormalize( true );
  outputter.setIndent( true );
  outputter.setNewlines( true );
  xupdateStr = outputter.outputString( xupdateDoc );
}
catch ( Exception ex ) {
  ex.printStackTrace();
}

org.w3c.dom.Document result = null;
try {
  // Perform XUpdate query ... We clone() originalDoc because it gets modified by XUpdate
  // and we wnat to keep the old version around
  result = updateFile( xupdateStr, (org.w3c.dom.Document)originalDoc.cloneNode( true ) );

  // Now write this document to a file
  String resultFile = "/nfshome/fabian/Backup/MEng_Thesis/implementation/xml/MEng_dc_updated.xml";
  Serializer serializer = SerializerFactory.getSerializer
          ( OutputProperties.getDefaultMethodProperties( "xml" ) );
  serializer.setOutputStream( new FileOutputStream( resultFile ) );
  serializer.asDOMSerializer().serialize( result );
}
catch ( Exception ex ) {
  ex.printStackTrace();
}

// Now go through XUpdate query file looking for inserts and deletes
// so that the XPointer matches in the database can be updated
try {
  // Go to database and retrieve "translated" document matchings as a Vector
  Vector matchings = new Vector();

  Class.forName( "org.postgresql.Driver" );
  String user = "xxxxxx";
  String pass = "xxxxxx";
  String url = "xxxxxxxxxx";
  Connection con = DriverManager.getConnection( url, user, pass );
  Statement stat = con.createStatement();
  String sqlstr = "SELECT translated FROM Matchings";
  ResultSet rs = stat.executeQuery( sqlstr );
  while ( rs.next() ) {
          matchings.addElement( rs.getString( 1 ) );
  }
  rs.close();
  stat.close();
  con.close();

  // Generate matchings for "updated" document based on XUpdate Query and original document
  XMLReader xmlReader = XMLReaderFactory.createXMLReader( vendorParserClass );
  XUpdateSAXHandler handler = new XUpdateSAXHandler( (Vector)matchings.clone(), originalDoc );
  xmlReader.setContentHandler( handler );
  xmlReader.parse( new InputSource( new StringReader( xupdateStr ) ) );

  Vector updatedMatchings = handler.getUpdatedMatchings();

  // Connect to database and store matchings for 'updated' document
```

```
            con = DriverManager.getConnection( url, user, pass );
            // Update values of old matchings
            sqlstr = "UPDATE Matchings SET updated = ? WHERE translated = ?";
            PreparedStatement preparedStat = con.prepareStatement( sqlstr );
            for ( int i = 0; i<matchings.size(); i++ ) {
                preparedStat.setString( 1, (String)updatedMatchings.elementAt( i ) );
                preparedStat.setString( 2, (String)matchings.elementAt( i ) );
                preparedStat.executeUpdate();
            }

            // Now add matchings for new nodes into database. These are basically
            // extra elements in the updatedMatchings vector that are at positions >= size
            // of matchings vector
            sqlstr = "INSERT INTO Matchings ( updated ) VALUES (?)";
            preparedStat = con.prepareStatement( sqlstr );
            // Iterate over matchings Vector and perform insertions
            for ( int i = matchings.size(); i<updatedMatchings.size(); i++ ) {
                preparedStat.setString( 1, (String)updatedMatchings.elementAt( i ) );
                preparedStat.executeUpdate();
            }
            preparedStat.close();
            con.close();
        }
        catch ( Exception ex ) {
            ex.printStackTrace();
        }

    }

    /**
     * Performs XUpdate-query and returns the result
     */
    public static org.w3c.dom.Document updateFile( String query, org.w3c.dom.Document inputDoc )
            throws Exception {
        //update
        XUpdateQuery xupdate = XUpdateQueryFactory.newInstance( ).newXUpdateQuery( );
        xupdate.setQString( query );
        xupdate.execute( inputDoc );
        //return
        return inputDoc;
    }

    /**
     * Parses input file and generates DOM.
     */
    public static org.w3c.dom.Document parseInputFile ( String filename ) throws Exception {
        if (filename == null) {
            throw new IllegalArgumentException( "name of input file must not be null !" );
        }
            XercesDOMAdapter adapter = new XercesDOMAdapter();
            org.w3c.dom.Document doc = adapter.getDocument( new FileInputStream ( new File ( filename ) ), false );
            return doc;
    }
}
```

9.4 XUpdateSAXHandler.java

This file is used in conjunction with XUpdateTool.java to provide an implementation of the generateUpdateMatchings algorithm of Figure 5-11 (pg. 42).

```
/**
 * Fabian F. Morgan
 * April 26, 2002
 **/
package edu.mit.metamedia.util;
```

```
import org.apache.oro.text.regex.*;
import java.io.*;
import javax.xml.parsers.*;
import org.w3c.dom.Document;

import org.xml.sax.XMLReader;
import org.xml.sax.Attributes;
import org.xml.sax.InputSource;
import org.xml.sax.helpers.XMLReaderFactory;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.SAXException;
import org.xml.sax.ext.LexicalHandler;

import java.util.*;

import org.jdom.input.*;
import org.jdom.output.XMLOutputter;
import org.jdom.*;

public class XUpdateSAXHandler extends DefaultHandler {
   private Vector matchings = null;
   private org.w3c.dom.Document originalXercesDocument = null;
   private org.jdom.Document originalDoc = null;

   // Flag Definitions
   private boolean insertBeforeOpened = false;
   private boolean insertAfterOpened = false;
   private boolean removeOpened = false;

   // Temporary Variables
   private String tmpOriginalPath = null;
   private String tmpUpdatedPath = null;

   public XUpdateSAXHandler( Vector matchings, org.w3c.dom.Document originalXercesDocument ) {
         this.matchings = matchings;
         this.originalXercesDocument = originalXercesDocument;
         DOMBuilder domBuilder = new DOMBuilder();
         originalDoc = domBuilder.build( originalXercesDocument );
   }

   public void startDocument() { }

   public void startElement (String uri, String name,
                                    String qName, Attributes atts) {
         if ( qName.equals( "xupdate:insert-before" ) ) {
            insertBeforeOpened = true;
            tmpOriginalPath = atts.getValue( "select" );
         }
         else if ( qName.equals( "xupdate:insert-after" ) ) {
            insertAfterOpened = true;
            tmpOriginalPath = atts.getValue( "select" );
         }
         else if ( qName.equals( "xupdate:remove" ) ) {
            removeOpened = true;
            tmpOriginalPath = atts.getValue( "select" );

            matchings.setElementAt( "", matchings.indexOf( tmpOriginalPath ) ); // Deleted Node - maps to nothing in updated
                  // document

            // Now search all other nodes of this type and decrement their position() information
            // or leave unchanged as needed
            // For example, if we originally had /dc:creator[1], /dc:creator[2], and /dc:creator[3] and we removed
            // /dc:creator[2], then /dc:creator[3] would need to be changed to /dc:creator[2]. /dc:creator[1] would
            // remain unchanged

            String match = null;
            int similarElementCount = 0; // Counter tallying how many nodes are similar to the one we deleted
            String similarPath = tmpOriginalPath.substring( 0, tmpOriginalPath.lastIndexOf( '[' ) ); // The node without position info
            int similarPathPosition = getPositionInformation( tmpOriginalPath ); // The position info of the node we deleted
            for ( int i = 0; i<matchings.size(); i++ ) {
```

```
                match = (String)matchings.elementAt( i );
                if ( match.startsWith( similarPath ) ) {
                    similarElementCount++;
                    int matchPosition = getPositionInformation( match );
                    if ( hasPositionInformation( match ) &&
                            ( matchPosition > similarPathPosition ) ) {
                            // Update this value in matchings vector
                            matchings.setElementAt( updatePositionInformation( match, (matchPosition - 1)), i );
                    }
                }
        }

        // Now handle special case when the node we deleted was one of only two such nodes.
        // For example, if we originally had /dc:creator[1] and /dc:creator[2], and we deleted /dc:creator[2],
        // then we would want /dc:creator to remain, NOT /dc:creator[1].
        if ( similarElementCount == 1 ) {
                // Since we know there were only two such elements originally, the one that was left must
                // have it's position = 1
                matchings.setElementAt( similarPath, matchings.indexOf( similarPath + "[1]" ) );
        }
    }
    else if ( qName.equals( "xupdate:element" ) ) {
        if ( insertBeforeOpened ) {
                tmpUpdatedPath = tmpOriginalPath + "/precedingSibling::" + atts.getValue( "name" );
                insertBeforeOpened = false;

                // Now search all other nodes of this type and increment/decrement their position() information
                // as necessary
                try {
                    /** Ex. suppose updatePath = "/rdf:RDF/rdf:Description/dc:publisher/precedingSibling::dc:identifier"
                            That is, we want to insert a dc:identifier element after the dc:publisher element
                            Here are values of following variables:
                            parentStr = "/rdf:RDF/rdf:Description"
                            similarPath = "dc:indentifier"
                            precedingSibling = "dc:publisher"
                            similarPathFull = "/rdf:RDF/rdf:Description/dc:identifier";
                    **/
                    String parentStr = tmpOriginalPath.substring( 0, tmpOriginalPath.lastIndexOf('/') );
                    Element parent = UpdateMergeTool.getElement( originalDoc, parentStr );
                    List childrenList = parent.getChildren();
                    Object content = null;
                    String similarPath = atts.getValue( "name" );
                    String precedingSibling = tmpOriginalPath.substring( tmpOriginalPath.lastIndexOf( '/' ) + 1,
                            tmpOriginalPath.length() ); // The element we want to insert after
                                                        // the precedingSibling ( the name only )
                    String similarPathFull = parentStr + "/" + similarPath;  // The full path of the element we want to insert
                    int similarElementCount = 0;
                    boolean foundPrecedingSibling = false;
                    boolean precedingSiblingHasPositionInformation = hasPositionInformation( precedingSibling );
                    int desiredNumberPrecedingSiblings = getPositionInformation( precedingSibling );
                    if ( precedingSiblingHasPositionInformation ) {
                            precedingSibling = precedingSibling.substring( 0, precedingSibling.lastIndexOf( '[' ) );
                    }
                    int currentNumberPrecedingSiblings = 0;  // The current number of preceding siblings we've encountered
                    Vector newChildrenContent = new Vector();
                    boolean propagatedChanges = false;
                    int numberElementsBeforePrecedingSibling = 0;
                    for ( int i = childrenList.size() - 1; i>=0 ; i-- ) {
                            content = childrenList.get( i );
                            if ( UpdateMergeTool.isTextElement( content ) ) {
                                newChildrenContent.addElement( ((Text)content).clone() );
                                continue;
                            }
                            else {
                                if ( propagatedChanges ) {
                                        newChildrenContent.addElement( ((Element)content).clone() );
                                        continue;
                                }
                                Element elem = (Element)content;
                                String elemName = elem.getQualifiedName();
```

```
if ( elemName.equals( similarPath ) ) {
        similarElementCount++;
        numberElementsBeforePrecedingSibling++;
        newChildrenContent.addElement( ((Element)content).clone() );
}

if ( elemName.equals( precedingSibling ) ) {
        currentNumberPrecedingSiblings++;
        if ( precedingSiblingHasPositionInformation ) {
           if ( currentNumberPrecedingSiblings == desiredNumberPrecedingSiblings ) {
                   foundPrecedingSibling = true;
                   newChildrenContent.addElement( ((Element)content).clone() );
                   similarElementCount++;

                   if ( numberElementsBeforePrecedingSibling == 0 ) {
                      int indexToUpdate = matchings.indexOf( similarPathFull );
                      if ( indexToUpdate != -1 ) {
                          matchings.setElementAt( similarPathFull + "[1]" , indexToUpdate );
                      }
                      propagatedChanges = true;
                   }
                   else {
                      // Add 1 to the position numbers of all similar elements after this
                      // one
                      propagateChanges( matchings, similarElementCount - 1, similarPathFull );
                      propagatedChanges = true;
                   }

                   // Add inserted element to newChildren list
                   if ( hasNameSpace( similarPath ) ) {
                      Namespace ns = originalDoc.getRootElement().getNamespace(
                              getNameSpaceFromString( similarPath ) );
                      newChildrenContent.addElement(
                        new org.jdom.Element(  removeNameSpaceFromString( similarPath )
                                                                              , ns ) );
                   }
                   else {
                      newChildrenContent.addElement( new org.jdom.Element( similarPath ) );
                   }
           }
        }
        else {
           foundPrecedingSibling = true;
           newChildrenContent.addElement( ((Element)content).clone() );
           similarElementCount++;

           if ( numberElementsBeforePrecedingSibling == 0 ) {
                   int indexToUpdate = matchings.indexOf( similarPathFull );
                   if ( indexToUpdate != -1 ) {
                      matchings.setElementAt( similarPathFull + "[1]" , indexToUpdate );
                   }
                   propagatedChanges = true;
           }
           else {
                   // Add 1 to the position numbers of all similar elements after this
                   // one
                   propagateChanges( matchings, similarElementCount - 1, similarPathFull );
                   propagatedChanges = true;
           }

           // Add inserted element to newChildren list
           if ( hasNameSpace( similarPath ) ) {
                   Namespace ns = originalDoc.getRootElement().getNamespace(
                       getNameSpaceFromString( similarPath ) );
                   newChildrenContent.addElement( new org.jdom.Element(
                     removeNameSpaceFromString( similarPath )  , ns ) );
           }
           else {
                   newChildrenContent.addElement( new org.jdom.Element( similarPath ) );
           }
```

```
                    }
                }
                else {  // Add all other elements to newChildren list
                    newChildrenContent.addElement( ((Element)content).clone() );
                }
            }
        }

        // Now insert node
        if ( similarElementCount == 1 ) {
            matchings.addElement( similarPathFull );
        }
        else if ( numberElementsBeforePrecedingSibling == 0 ) {
            matchings.addElement( similarPathFull + "[" + similarElementCount + "]" );
        }
        else {
            matchings.addElement( similarPathFull + "[" +
                ( similarElementCount - numberElementsBeforePrecedingSibling ) + "]" );
        }

        // Modify originalDoc to include added nodes
        parent.setContent( newChildrenContent );
    }
    catch ( Exception ex ) {
        ex.printStackTrace();
    }
}
else if ( insertAfterOpened ) {
    tmpUpdatedPath = tmpOriginalPath + "/followingSibling::" + atts.getValue( "name" );
    insertAfterOpened = false;

    // Now search all other nodes of this type and increment/decrement their position() information
    // as necessary.  Note: we are modifying the originalDoc instance as we do this
    try {
        /** Ex. suppose updatePath = "/rdf:RDF/rdf:Description/dc:publisher/followingSibling::dc:identifier"
            That is, we want to insert a dc:identifier element after the dc:publisher element
            Here are values of following variables:
            parentStr = "/rdf:RDF/rdf:Description"
            similarPath = "dc:indentifier"
            followingSibling = "dc:publisher"
            similarPathFull = "/rdf:RDF/rdf:Description/dc:identifier";
        **/
        String parentStr = tmpOriginalPath.substring( 0, tmpOriginalPath.lastIndexOf('/') );
        Element parent = UpdateMergeTool.getElement( originalDoc, parentStr );
        List childrenList = parent.getChildren();
        Object content = null;
        String similarPath = atts.getValue( "name" );
        String followingSibling = tmpOriginalPath.substring( tmpOriginalPath.lastIndexOf( '/' ) + 1,
                    tmpOriginalPath.length() ); // The element we want to insert after
                                                // the followingSibling ( the name only )
        String similarPathFull = parentStr + "/" + similarPath;  // The full path of the element we want to insert
        int similarElementCount = 0;
        boolean foundFollowingSibling = false;
        boolean followingSiblingHasPositionInformation = hasPositionInformation( followingSibling );
        int desiredNumberFollowingSiblings = getPositionInformation( followingSibling );
        if ( followingSiblingHasPositionInformation ) {
            followingSibling = followingSibling.substring( 0, followingSibling.lastIndexOf( '[' ) );
        }
        int currentNumberFollowingSiblings = 0;  // The current number of following siblings we've encountered
        int insertPositionNum = 0;  // The position where we'll insert the new node
        Vector newChildrenContent = new Vector();
        boolean propagatedChanges = false;
        for ( int i = 0; i<childrenList.size(); i++ ) {
            content = childrenList.get( i );
            if ( UpdateMergeTool.isTextElement( content ) ) {
                newChildrenContent.addElement( ((Text)content).clone() );
                continue;
            }
            else {
                if ( propagatedChanges ) {
```

```
                newChildrenContent.addElement( ((Element)content).clone() );
                continue;
            }

            Element elem = (Element)content;
            String elemName = elem.getQualifiedName();
            if ( elemName.equals( similarPath ) ) {
                similarElementCount++;

                if ( foundFollowingSibling ) {
                   // Add 1 to the position numbers of all similar elements after this
                   // one
                   propagateChanges( matchings, similarElementCount - 1, similarPathFull );
                   propagatedChanges = true;
                }
                else {
                   int indexToUpdate = matchings.indexOf( similarPathFull );
                   if ( indexToUpdate != -1 ) {
                         matchings.setElementAt( similarPathFull + "[1]" , indexToUpdate );
                   }
                }
                newChildrenContent.addElement( ((Element)content).clone() );
            }

            if ( elemName.equals( followingSibling ) ) {
                currentNumberFollowingSiblings++;
                if ( followingSiblingHasPositionInformation ) {
                   if ( currentNumberFollowingSiblings == desiredNumberFollowingSiblings ) {
                         foundFollowingSibling = true;
                         newChildrenContent.addElement( ((Element)content).clone() );
                         similarElementCount++;
                         insertPositionNum = similarElementCount;

                         if ( hasNameSpace( similarPath ) ) {
                            Namespace ns = originalDoc.getRootElement().getNamespace(
                                  getNameSpaceFromString( similarPath ) );
                            newChildrenContent.addElement( new org.jdom.Element(
                                  removeNameSpaceFromString( similarPath ), ns ) );
                         }
                         else {
                            newChildrenContent.addElement( new org.jdom.Element( similarPath ) );
                         }
                   }
                }
                else {
                   foundFollowingSibling = true;
                   newChildrenContent.addElement( ((Element)content).clone() );
                   similarElementCount++;
                   insertPositionNum = similarElementCount;

                   if ( hasNameSpace( similarPath ) ) {
                         Namespace ns = originalDoc.getRootElement().getNamespace(
                                  getNameSpaceFromString( similarPath ) );
                         newChildrenContent.addElement( new org.jdom.Element(
                               removeNameSpaceFromString( similarPath ) , ns ) );
                   }
                   else {
                         newChildrenContent.addElement( new org.jdom.Element( similarPath ) );
                   }
                }
            }
            else {  // Add all other elements to newChildren list
                newChildrenContent.addElement( ((Element)content).clone() );
            }
         }
}
// Now insert node
if ( similarElementCount == 1 ) {
         matchings.addElement( similarPathFull );
}
```

```
                    else {
                        matchings.addElement( similarPathFull + "[" + insertPositionNum + "]" );
                    }
                    // Modify originalDoc to include added nodes
                    parent.setContent( newChildrenContent );
                }
                catch ( Exception ex ) {
                    ex.printStackTrace();
                }
            }
        }
}

public void endElement (String uri, String name, String qName) { }
public void characters (char ch[], int start, int length) { }

public Vector getUpdatedMatchings() {
        return matchings;
}

/** Utility Methods **/
public int getPositionInformation( String input ) {
        PatternCompiler compiler = null;
        PatternMatcher matcher = null;
        PatternMatcherInput matcherInput = null;
        Pattern pattern = null;
        try {
            // Use Jakarta ORO to parse for the position() information
            compiler = new Perl5Compiler();
            matcher = new Perl5Matcher();
            matcherInput = new PatternMatcherInput( input );
            MatchResult matchResult = null;
            pattern = null;
            String regexp = "(.*?)\\[(.)\\]";
            pattern = compiler.compile( regexp, Perl5Compiler.CASE_INSENSITIVE_MASK );

            if ( matcher.matches( matcherInput, pattern ) ) {
                    matchResult = matcher.getMatch();
                    return Integer.parseInt( matchResult.group( 2 ) );
            }
        }
        catch ( Exception ex ) {
            ex.printStackTrace();
        }
        return 0;
}

public boolean hasPositionInformation( String input ) {
        PatternCompiler compiler = null;
        PatternMatcher matcher = null;
        PatternMatcherInput matcherInput = null;
        Pattern pattern = null;
        try {
            // Use Jakarta ORO to parse for the position() information
            compiler = new Perl5Compiler();
            matcher = new Perl5Matcher();
            matcherInput = new PatternMatcherInput( input );
            MatchResult matchResult = null;
            pattern = null;
            String regexp = "(.*?)\\[(.)\\]";
            pattern = compiler.compile( regexp, Perl5Compiler.CASE_INSENSITIVE_MASK );
        }
        catch ( Exception ex ) {
            ex.printStackTrace();
        }

        return matcher.matches( matcherInput, pattern );
}

public String updatePositionInformation( String input, int newPosition ) {
```

```
        String pathToPosition = input.substring( 0, input.lastIndexOf( '[' ) );
        String updatedStr = pathToPosition + "[" + String.valueOf( newPosition ) + "]";
        return updatedStr;
  }

  public boolean hasNameSpace( String input ) {
        int pos = input.indexOf( ':' );
        return (pos == -1) ? false : true;
  }

  public String getNameSpaceFromString( String input ) {
        return input.substring( 0, input.indexOf( ':' ) );
  }

  public String removeNameSpaceFromString( String input ) {
        return input.substring( input.indexOf( ':' ) + 1, input.length() );
  }

  public void propagateChanges( Vector matchings, int start, String similarPathFull ) {
        int indexToUpdate = 0;
        if ( start == 1 ) {
          indexToUpdate = matchings.indexOf( similarPathFull );
          if ( indexToUpdate == -1 ) {
                indexToUpdate = matchings.indexOf( similarPathFull + "[1]" );
          }
        }
        else {
          indexToUpdate = matchings.indexOf( similarPathFull + "[" + start + "]" );
        }

        int currentPositionValue = start;
        Hashtable hash = new Hashtable();
        while ( indexToUpdate != -1 ) {
          currentPositionValue++;
          hash.put( new Integer( indexToUpdate ), new Integer( currentPositionValue ) );
          indexToUpdate = matchings.indexOf( similarPathFull + "[" + currentPositionValue + "]" );
        }

        for ( Enumeration e = hash.keys(); e.hasMoreElements(); ) {
          Integer indexInteger = (Integer)e.nextElement();
          int index = indexInteger.intValue();
          int newPosition = ((Integer)hash.get( indexInteger )).intValue();
          matchings.setElementAt( similarPathFull + "[" + newPosition + "]", index );
        }
  }
}
```

9.5 UpdateMergeTool.java

This file contains a Java implementation of the updateAndMerge algorithm of Figure 5-12 (pg. 46).

```
package edu.mit.metamedia.util;

import java.util.*;
import java.sql.*;
import java.io.*;
import org.jdom.input.SAXBuilder;
import org.jdom.*;
import org.apache.oro.text.regex.*;

import org.jdom.output.XMLOutputter; // For debugging

public class UpdateMergeTool {
  public static void main( String[] args ) {
        // Declare Vector of matchings.  Each element is a String[] array with two elements.
        // The first element is the match key (an XPath, or the empty string), and the
        // second element is the match value (an XPath, or the empty string)
```

```
Vector matchings = new Vector();

// Connect to database and retrieve Matchings data
try {
   Class.forName( "org.postgresql.Driver" );
   String user = "xxxxxx";
   String pass = "xxxxxx";
   String url = "xxxxxxxxxx";
   Connection con = DriverManager.getConnection( url, user, pass );
   Statement stat = con.createStatement();
   String sqlstr = "SELECT original, derived FROM Matchings ORDER BY original"; // Must do updates before inserts
                  // and deletes
   ResultSet rs = stat.executeQuery( sqlstr );
   String[] matchPair = null;
   while ( rs.next() ) {
            matchPair = new String[ 2 ];
            matchPair[0] = ( rs.getString( 1 ) == null ) ? "" : rs.getString( 1 );
            matchPair[1] = ( rs.getString( 2 ) == null ) ? "" : rs.getString( 2 );
            matchings.addElement( matchPair );
   }
   rs.close();
   stat.close();
   con.close();
}
catch ( Exception ex ) {
   ex.printStackTrace();
}

// Parse original and derived documents into their respective DOM's
Document originalDoc = null;
Document derivedDoc = null;
try {
   SAXBuilder builder = new SAXBuilder( "org.apache.xerces.parsers.SAXParser" );
   File originalDocFile = new File ( "/nfshome/fabian/Backup/MEng_Thesis/implementation/xml/MEng_tei_original.xml"
);
   originalDoc = builder.build( originalDocFile );
   File derivedDocFile = new File ( "/nfshome/fabian/Backup/MEng_Thesis/implementation/xml/MEng_tei_derived.xml"
);
   derivedDoc = builder.build( derivedDocFile );
}
catch ( Exception ex ) {
   ex.printStackTrace();
}

// The crux of the algorithm
String[] matchPair = null;
String matchKey = null;
String matchValue = null;
try {
   for ( int i =0; i < matchings.size(); i++ ) {
            matchPair = (String[])matchings.elementAt( i );
            matchKey = matchPair[0];
            matchValue = matchPair[1];
            if ( !(matchKey.equals( "" ) ) && !(matchValue.equals( "" ) ) ) {
               updateElement( getElement( originalDoc, matchKey ),
                              getElement( derivedDoc, matchValue ) );
            }
            else if ( !(matchKey.equals( "" ) ) && matchValue.equals( "" ) ) {
               markElementToDelete( getElement( originalDoc, matchKey ) );
            }
            else if ( matchKey.equals( "" ) && !(matchValue.equals( "" ) ) ) {
               insertElement( originalDoc, getElement( derivedDoc, matchValue ),
                              matchValue );
            }
   }

   // Do a post order traversal of originalDoc and delete all elements
   // that were marked to be deleted
   deleteMarkedElements( originalDoc.getRootElement() );
```

```
			// Serialize final merged file to disk
			XMLOutputter outputter = new XMLOutputter();
			outputter.setTextNormalize( true );
			outputter.setIndent( true );
			outputter.setNewlines( true );

			File mergedDocFile = new File( "/nfshome/fabian/Backup/MEng_Thesis/implementation/xml/MEng_tei_merged.xml"
);
			FileWriter writer = new FileWriter( mergedDocFile );
			writer.write( outputter.outputString( originalDoc ) );
			writer.close();
		}
		catch ( Exception ex ) {
			ex.printStackTrace();
		}
	}

	/** DOM Modifier methods **/
	public static void updateElement( Element origElem, Element derivedElem ) throws Exception {
		updateAttributes( origElem, derivedElem );

		List origElemContent = origElem.getContent();
		List derivedElemContent = derivedElem.getContent();
		Text origElemText = null;
		Text derivedElemText = null;
		Object content1 = null;
		Object content2 = null;
		boolean content1IsLeaf = false;
		boolean content2IsLeaf = false;
		Vector newContent = new Vector();
		for ( int i = 0; i<origElemContent.size(); i++ ) {
			content1 = origElemContent.get( i );
			try {
				content2 = derivedElemContent.get( i );
			}
			catch ( IndexOutOfBoundsException ioobe ) {
				if ( isTextElement( content1 ) ) {
					newContent.addElement( ((Text)content1).clone() );
				}
				else {
					newContent.addElement( ((Element)content1).clone() );
				}
				continue;
			}
			content1IsLeaf = isTextElement( content1 );
			content2IsLeaf = isTextElement( content2 );
			if ( content1IsLeaf && content2IsLeaf ) {
				newContent.addElement( ((Text)content2).clone() );
			}
			else if ( content1IsLeaf && !(content2IsLeaf) ) {
				newContent.addElement( new Text( "" ) );
			}
			else if ( !(content1IsLeaf) && content2IsLeaf ) {
				newContent.addElement( ((Text)content2).clone() );
			}
			else {
				newContent.addElement( ((Element)content1).clone() );
			}
		}

		// Add additional text nodes from the derived Document to the
		// original Document we are modifying
		if ( derivedElemContent.size() > origElemContent.size() ) {
			for ( int i = origElemContent.size(); i < derivedElemContent.size(); i++ ) {
				content2 = derivedElemContent.get( i );
				content2IsLeaf = isTextElement( content2 );
				if ( content2IsLeaf ) {
					newContent.addElement( ((Text)content2).clone() );
				}
			}
```

```
        }
        origElem.setContent( newContent );
}

public static void markElementToDelete( Element origElem ) throws Exception {
        origElem.setAttribute( "metamediaDeleteElement", "true" );
}

public static void insertElement( Document originalDoc, Element elemToInsert,
                                        String xpath ) throws Exception {
        String pathToParent = xpath.substring( 0, xpath.lastIndexOf( '/' ) );

        Element parent = getElement( originalDoc, pathToParent );

        // Determine if end of xpath string had any position information, so
        // we would know where to insert the new Element
        String endOfXpath = xpath.substring( xpath.lastIndexOf( '/' ) + 1, xpath.length() );

        // Use Jakarta ORO to parse for this position() information
        PatternCompiler compiler = new Perl5Compiler();
        PatternMatcher matcher = new Perl5Matcher();
        PatternMatcherInput matcherInput = new PatternMatcherInput( endOfXpath );
        MatchResult matchResult = null;
        Pattern pattern = null;
        String regexp = "(.*?)\\[(.)\\]";
        pattern = compiler.compile( regexp, Perl5Compiler.CASE_INSENSITIVE_MASK );
        Vector newContent = new Vector();
        if ( matcher.matches( matcherInput, pattern ) ) {
          matchResult = matcher.getMatch();

          String desiredElementName = matchResult.group( 1 );
          int positionNum = Integer.parseInt( matchResult.group( 2 ) );
          List parentContent = parent.getContent();
          int desiredElementCount = 0;
          Object currentContent = null;
          boolean addedNewElement = false;
          for ( int i = 0; i<parentContent.size(); i++ ) {
                currentContent = parentContent.get( i );
                if ( isTextElement( currentContent ) ) {
                  newContent.addElement( ((Text)currentContent).clone() );
                }
                else if ( ((Element)currentContent).getName().equals( desiredElementName ) ) {
                  desiredElementCount++;

                  if ( addedNewElement ) {
                        newContent.addElement( ((Element)currentContent).clone() );
                  }
                  // Handle case where element you want to insert has position = 1, meaning
                  // it should be added before any other element with the same name
                  else if ( positionNum == 1 ) {
                        // Insert the new node
                        newContent.addElement( ((Element)elemToInsert).clone() );
                        addedNewElement = true;

                        newContent.addElement( ((Element)currentContent).clone() );
                  }
                  else if ( desiredElementCount == positionNum - 1 ) {
                        newContent.addElement( ((Element)currentContent).clone() );
                        // Insert the new node
                        newContent.addElement( ((Element)elemToInsert).clone() );
                        addedNewElement = true;
                  }
                  else {
                        newContent.addElement( ((Element)currentContent).clone() );
                  }
                }
                else {
                  newContent.addElement( ((Element)currentContent).clone() );
                }
          }
```

```
            if ( !(addedNewElement) ) {  // Then we should add element at end of current list
                    newContent.addElement( ((Element)elemToInsert).clone() );
            }
            parent.setContent( newContent );
        }
        else { // no position() information in last bit of xpath
            parent.addContent( (Element)elemToInsert.clone() );
        }
}

public static void updateAttributes( Element origElem, Element derivedElem ) {
        List atts = derivedElem.getAttributes();
        Iterator iter = atts.iterator();
        Attribute attribute = null;
        while ( iter.hasNext() ) {
            attribute = (Attribute)iter.next();
            origElem.setAttribute( attribute.getName(), attribute.getValue() );
        }
}

// Breadth-first search implementation of post-order traversal
public static void deleteMarkedElements( Element rootElement ) {
        List childrenList = rootElement.getChildren();
        Element tmpElem = null;
        if ( childrenList == null )
            return;
        else {
            for ( int i = 0; i<childrenList.size(); i++ ) {
                    tmpElem = (Element)childrenList.get( i );
                    if ( tmpElem.getAttribute( "metamediaDeleteElement" ) != null ) {
                        // Node was marked for deletion
                        tmpElem.getParent().removeContent( tmpElem );
                    }
            }
        }
        // Recursively process children of these nodes
        for ( int i = 0; i<childrenList.size(); i++ ) {
            tmpElem = (Element)childrenList.get( i );
            deleteMarkedElements( tmpElem );
        }
}

/** End DOM Modifier methods **/

/** Helper methods **/
public static Element getElement( Document doc, String xpath ) throws Exception {
        StringTokenizer tokenizer = new StringTokenizer( xpath, "/" );
        Element rootElement = doc.getRootElement();
        String elementString = null;

        // We want to skip the first token, since that contains the name
        // of the root element
        boolean foundFirst = false;
        while ( tokenizer.hasMoreTokens() ) {
            if ( !foundFirst ) {
                    foundFirst = true;
                    tokenizer.nextToken();
                    continue;
            }
           elementString = tokenizer.nextToken();
            rootElement = getElementHelper( rootElement, elementString );
        }
        return rootElement;
}

public static Element getElementHelper( Element elem, String elementString )
        throws Exception {
        // See if this xpath contains any position() information

        // Use Jakarta ORO to parse for this position() information
```

```
        PatternCompiler compiler = new Perl5Compiler();
        PatternMatcher matcher = new Perl5Matcher();
        PatternMatcherInput matcherInput = new PatternMatcherInput( elementString );
        MatchResult matchResult = null;
        Pattern pattern = null;
        String regexp = "(.*?)\\[(.)\\]";
        pattern = compiler.compile( regexp, Perl5Compiler.CASE_INSENSITIVE_MASK );
        Element returnElement = null;
        if ( matcher.matches( matcherInput, pattern ) ) {
          matchResult = matcher.getMatch();
          String desiredElementName = matchResult.group( 1 );
          int positionNum = Integer.parseInt( matchResult.group( 2 ) );
          List childrenList = elem.getChildren();
          int desiredElementCount = 0;
          for ( int i = 0; i<childrenList.size(); i++ ) {
                Element currentElem = (Element)childrenList.get( i );
                if ( currentElem.getQualifiedName().equals( desiredElementName ) ) {
                  desiredElementCount++;
                  if ( desiredElementCount == positionNum ) {
                        returnElement = currentElem;
                        return returnElement;
                  }
                }
          }
          // If code gets here, then we haven't found the element we were looking
          // for, so throw Exception
          throw new Exception( "Element at specified position didn't have " +
                                        "correct name" );
        }
        else { // no position() information in xpath
          // returnElement = elem.getChild( elementString );

          List childrenList = elem.getChildren();
          for ( int i = 0; i<childrenList.size(); i++ ) {
                Element currentElem = (Element)childrenList.get( i );
                if ( currentElem.getQualifiedName().equals( elementString ) ) {
                  returnElement = currentElem;
                  return returnElement;
                }
          }
        }
        return returnElement;
  }
  public static boolean isTextElement( Object obj ) {
        try {
          Text txt = (Text)obj;
          return true;
        }
        catch ( ClassCastException ex ) {
        }
        return false;
  }
  /** End Helper methods **/
}
```

9.6 TEI_to_DC.xsl

This is a stylesheet that converts an XML document conforming to the TEI schema to a document conforming to the Dublin Core schema. Note that it contains extension function calls to the MatchingManager object.

```
<?xml version="1.0"?>
<!-- Fabian F. Morgan
   April 11, 2002
   Stylesheet that translates TEI to Dublin Core
-->
```

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0"
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:dc="http://purl.org/dc/elements/1.1/"
      xmlns:matcher="edu.mit.metamedia.util.MatchingManager"
      extension-element-prefixes="matcher"
      exclude-result-prefixes="matcher">

<xsl:output method="xml" indent="yes" />

<!-- Create new instance of extension object -->
<xsl:variable name="matchingManager" select="matcher:new()" />

<xsl:template match="/">
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:dc="http://purl.org/dc/elements/1.1/">
    <xsl:variable name="path" select="matcher:addPathElement( 'rdf:RDF' )" />
    <rdf:Description>
      <xsl:variable name="path" select="matcher:addPathElement( 'rdf:Description' )" />
      <!-- <xsl:apply-templates select="/TEI.2/teiHeader" /> -->
      <xsl:apply-templates select="/TEI.2/teiHeader" />
    </rdf:Description>
    <xsl:variable name="path" select="matcher:removeCurrentPathElement()" />
  </rdf:RDF>
  <xsl:variable name="path" select="matcher:removeCurrentPathElement()" />

  <!-- Serialize extension object to persistent store -->
  <matcher:serialize />
</xsl:template>

<xsl:template match="fileDesc/titleStmt">
  <xsl:apply-templates select="title | author" />
</xsl:template>

<xsl:template match="title">
  <xsl:variable name="matching" select="matcher:createMatch( ancestor-or-self::*, 'dc:title' )" />
  <xsl:choose>
    <xsl:when test="@type = 'main'">
      <dc:title><xsl:value-of select="."/></dc:title>
    </xsl:when>
    <xsl:when test="@type = 'sub'">
      <dc:title type="subtitle"><xsl:value-of select="."/></dc:title>
    </xsl:when>
    <xsl:otherwise>
      <dc:title><xsl:value-of select="."/></dc:title>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template match="author">
  <xsl:variable name="matching" select="matcher:createMatch( ancestor-or-self::*, 'dc:creator' )" />
  <dc:creator><xsl:value-of select="."/></dc:creator>
</xsl:template>

<xsl:template match="fileDesc/publicationStmt">
  <xsl:apply-templates select="pubPlace | publisher | idno | date"/>
</xsl:template>

<xsl:template match="pubPlace">
  <xsl:variable name="matching" select="matcher:createMatch( ancestor-or-self::*, 'dc:publisher' )" />
  <dc:publisher><xsl:value-of select="."/></dc:publisher>
</xsl:template>

<xsl:template match="publisher">
  <xsl:variable name="matching" select="matcher:createMatch( ancestor-or-self::*, 'dc:publisher' )" />
  <dc:publisher><xsl:value-of select="."/></dc:publisher>
</xsl:template>

<xsl:template match="idno">
  <xsl:variable name="matching" select="matcher:createMatch( ancestor-or-self::*, 'dc:identifier' )" />
  <dc:identifier><xsl:value-of select="."/></dc:identifier>
</xsl:template>
```

```
<xsl:template match="date">
  <xsl:variable name="matching" select="matcher:createMatch( ancestor-or-self::*, 'dc:date' )" />
  <dc:date><xsl:value-of select="."/></dc:date>
</xsl:template>

<xsl:template match="fileDesc/sourceDesc">
  <xsl:apply-templates select="bibl | biblFull" />
</xsl:template>

<xsl:template match="bibl | biblFull">
  <xsl:variable name="matching" select="matcher:createMatch( ancestor-or-self::*, 'dc:source' )" />
  <dc:source><xsl:value-of select="normalize-space( text() )"/></dc:source>
</xsl:template>

<xsl:template match="profileDesc/langUsage">
  <xsl:variable name="matching" select="matcher:createMatch( ancestor-or-self::*, 'dc:language' )" />
  <dc:language><xsl:value-of select="."/></dc:language>
</xsl:template>

<xsl:template match="profileDesc/textClass">
  <xsl:apply-templates select="keywords"/>
</xsl:template>

<xsl:template match="keywords">
  <xsl:variable name="matching" select="matcher:createMatch( ancestor-or-self::*, 'dc:subject' )" />
  <dc:subject><xsl:value-of select="."/></dc:subject>
</xsl:template>

<!-- Ignore Text/Extra Whitespace -->
<xsl:template match="text()">
</xsl:template>

</xsl:stylesheet>
```