ABSTRACT

DYNAMIC RECOMPOSITION OF DOCUMENTS FROM DISTRIBUTED DATA SOURCES

by
Abhishek Verma

Dynamic recomposition of documents refers to the process of on-the-fly creation of documents. A document can be generated from several documents that are stored at distributed data sites. The source can be queried and results obtained in the form of XML. These XML documents can be combined after a series of transformation operations to obtain the target document. The resultant document can be stored statically or in the form of a command, which can be invoked later to recompose this document dynamically. Also, in case a change is made to a document, then only the change can be stored, instead of storing the modified document in its entirety.

The purpose of this research was to provide a way to recompose dynamic documents. A solution is proposed at the level of algebra for update and recomposition of documents stored at distributed data sources. The issue of representation of a document by a command, i.e., a composition operator and/or an editing command along with one or more path expressions has also been researched. The construction of a dynamic document has three phases to it. The first one is the information retrieval. Phase two deals with building of real document: this includes the filtering of retrieved data by selecting relevant subset of a document and then applying update operations, and finally the ordering and assembling of the document. The final phase consists of displaying or storing or exchanging it over the web through a convenient means.

DYNAMIC RECOMPOSITION OF DOCUMENTS FROM DISTRIBUTED DATA SOURCES

by
Abhishek Verma

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer Science

May 2006

APPROVAL PAGE

DYNAMIC RECOMPOSITION OF DOCUMENTS FROM DISTRIBUTED DATA SOURCES

Abhishek Verma

Dr. Vincent Oria, Thesis Advisor — Date
Assistant Professor of Computer Science, NJIT

Dr. Michael P. Bieber, Committee Member — Date
Professor of Information Systems, NJIT

Dr. Dimitrios Theodoratos, Committee Member — Date
Associate Professor of Computer Science, NJIT

BIOGRAPHICAL SKETCH

Author: Abhishek Verma

Degree: Master of Science

Date: May 2006

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2006

- Master of Computer Applications,
  Bangalore University, Bangalore, India, 2003

- Bachelor in Commerce (Honors),
  University of Delhi, New Delhi, India, 2000

Major: Computer Science

To my parents, Dr. Bindra Prasad and Dr. Pushpa Verma;
and my sister, Sonika Sharma.

ACKNOWLEDGMENT

I would like to express my deepest appreciation to Dr. Vincent Oria, who not only served as my research supervisor, providing valuable and countless resources, insight, and intuition, but also constantly gave me support, encouragement, and reassurance. Special thanks are given to Dr. Michael Bieber and Dr. Dimitrios Thedoratos for actively participating in my committee.

I would like to thank my parents for their constant support and encouragement.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Objective

The objective of this thesis is to provide a way to recompose dynamic documents. This research provides a solution at the level of algebra for update and recomposition of documents stored at distributed data sources. The issue of representation of a document by a command, i.e., a composition operator and/or an editing command along with one or more path expressions has been researched as well.

Recently, document regeneration has emerged as an increasingly active research area. A document can be generated from many different documents that are stored at distributed data sites. The source can be queried and results can be obtained in the form of XML. These XML documents can be combined after a series of update and recomposition operations to obtain the target document. The resultant document can be stored statically or in the form of a command, which can be later invoked to recompose this document dynamically. Also, in case any change is made to a document, then only the change can be stored, instead of storing the modified document in entirety.

1.2 Virtual Documents

The model of the Web has shifted the expectations for access to information. Previously, the information was accessed by the retrieval of electronic copies of documents from a large repository of relatively static information. One now expects to access information through the manipulation of a large collection of information resources. Some of these

resources are documents and some of these resources are processes that create documents.

An electronic document consists of both the content and the links associated with that document. Therefore, documents on the Web may be composed of one or more Web pages (Crowston & Williams, 1999). A document can be stored statically and made persistent or it may be generated dynamically whenever the user desires and remain virtual. A virtual document is a document for which no persistent state exists and for which some or all of each instance is generated at run time (Watters 1999). A virtual document can then be multiple pages, application results, and may or may not have associated links. The content may be defined by a database query, a template, a program, or by some application.

1.3 Dynamic Recomposition of Documents

The idea of dynamic recomposition of documents refers to the process of on-the-fly creation of documents. The resulting document not only exactly matches a user's request but even may be adapted to the user's background and personality. Most of these visions of dynamic documents are heavily influenced by the enormous growth of the World Wide Web. The Web is an ideal base for dynamic documents because it can be used as both source and target.

The construction of a dynamic document has three phases to it. The first one is the information retrieval. The second one is the building of real document; this includes the filtering of retrieved data by selecting the relevant subset of documents and then applying update operations, and finally ordering and assembling of the document. The final phase

consists of displaying or storing or exchanging it over the web through a convenient means.

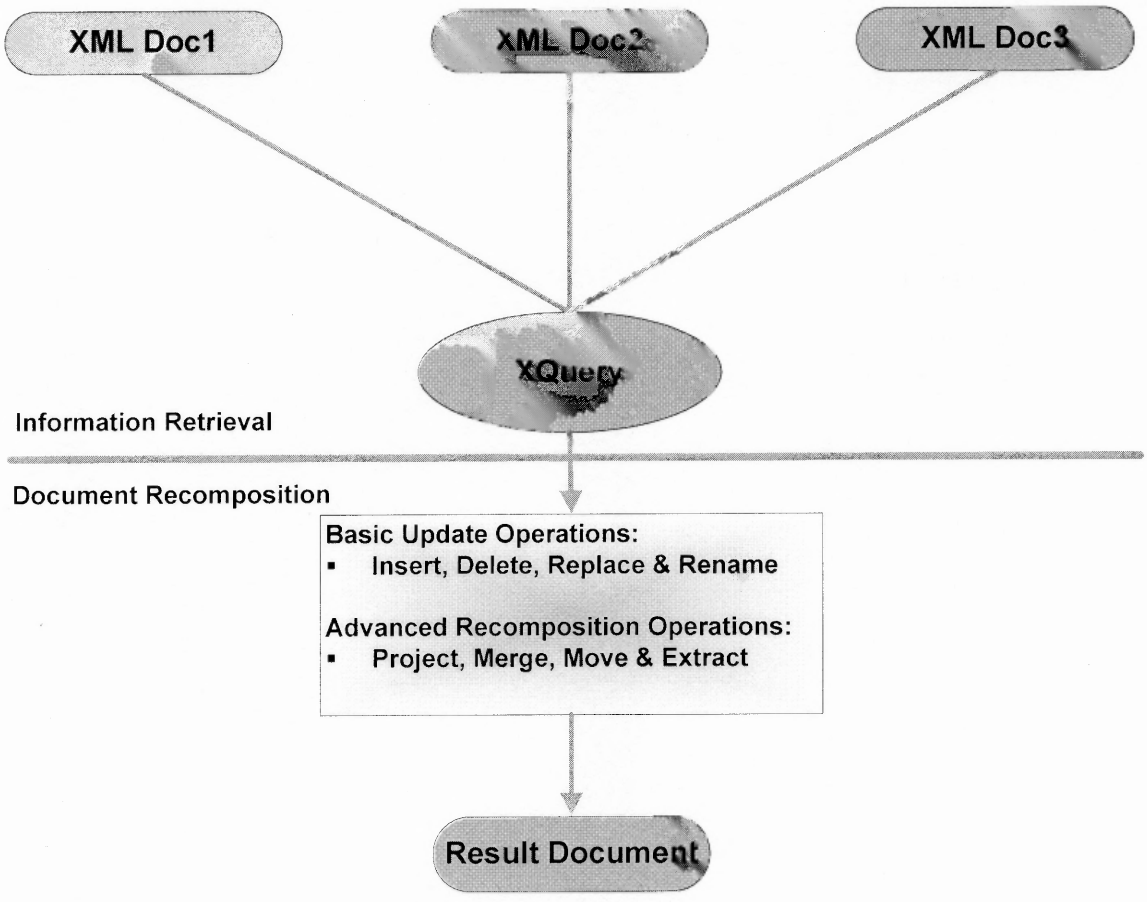

Figure 1.1 Process of dynamic recomposition of documents.

Figure 1.1 above shows the process of dynamic recomposition of documents. The XML documents XML Doc1, XML Doc2 and XML Doc3 are generated by distributed data sources. In the information retrieval phase, XQuery engine selects the subset of these documents and sends it to the document generation phase. During the document generation phase the intermediate document is modified by applying a series of update operations, i.e., insert that inserts copies of one or more nodes into a designated position,

delete for deletion of one or more nodes, replace to replace the value of node or node itself and rename to replace the name property of a data model node with a new name. The intermediate results are then subjected to advanced recomposition operations like projection, merge, move and extract. In final phase the result document may be presented to the user, stored in a database as a static document or represented in form of a command, or exchanged as an instance of XPath/XQuery Data Model.

1.4 System Description

Figure 1.2 shows interaction among various components of the system. The update or recomposition query goes to parser. Parser after it processes the query, forwards parsed

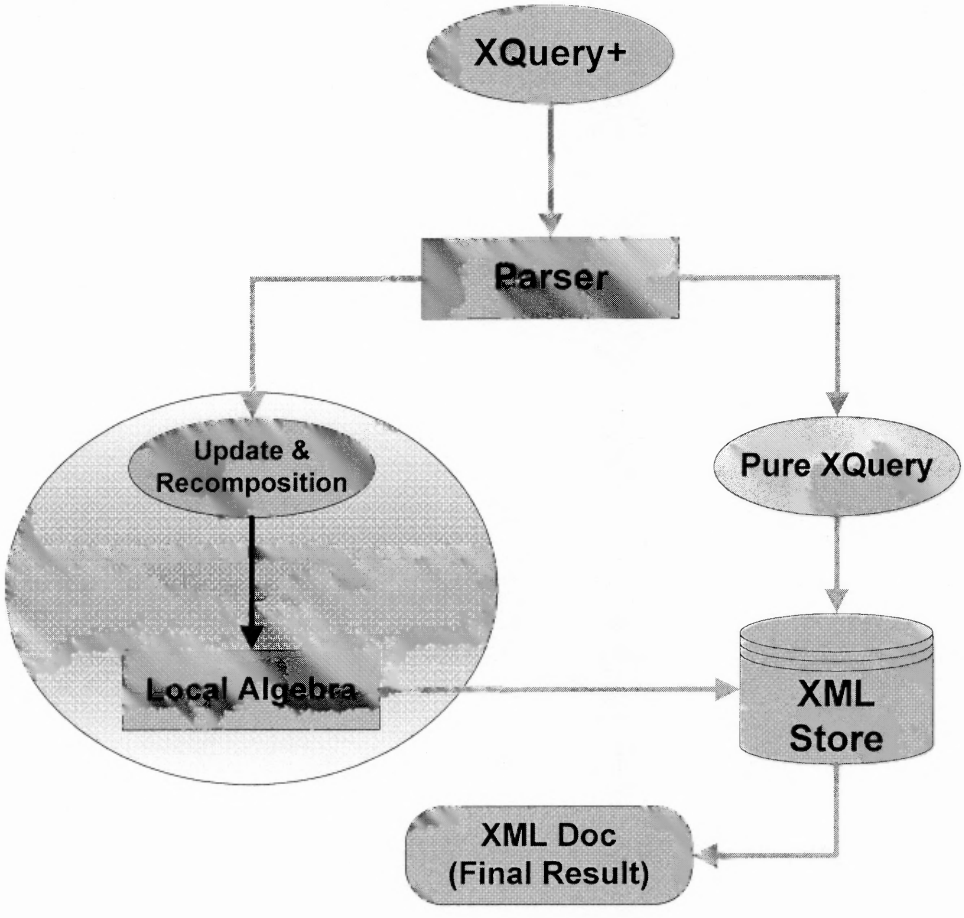

Figure 1.2 Components of the system and their interaction.

result to the module responsible for implementing various operations at the level of algebra. The local algebra module handles insert, delete, rename, replace, project, merge, move and extract operations. Resultant document is then forwarded to the XML store for persistent storage.

CHAPTER 2
BACKGROUND WORK

2.1 XML Document and Schema

XML stands for EXtensible Markup Language, it is a markup language much like HTML. The primary purpose of XML is to describe data. It does not have any predefined tags rather they can be defined by user giving utmost flexibility. A Document Type Definition (DTD) or an XML schema is used to describe data and structure of the document.

With XML, plain text files can be used to share data. Since documents are stored in plain text format, XML provides a software and hardware-independent way of sharing data. This makes it much easier to create data that different applications can work with. It also makes it easier to expand or upgrade a system to new operating systems, servers, applications, and new browsers. Another purpose of XML is to store data in form of plain text files or in databases. Applications can be written to store and retrieve information from the store, and generic applications can be used to display the data.

XML can make the data more useful by making it available to more users. Since XML is independent of hardware, software and application, the data is available to other than only standard HTML browsers. Other clients and applications can access XML files as data sources, like they are accessing databases. Data can be made available to all kinds of "reading machines" (agents), and it is easier to make it available for blind people, or people with other disabilities. It is used to exchange data between incompatible systems. In the real world, computer systems and databases contain data in incompatible formats.

One of the most time-consuming challenges for developers has been to exchange data between such systems over the internet. Converting data to XML can greatly reduce this complexity and create data that can be read by many different types of application.

The purpose of an XML schema is to define legal building blocks of an XML document, just like a DTD. It defines elements and attributes that can appear in a document. It defines which elements are child elements, their order and number. Also, it defines whether an element is empty or can include text along with their data types. The XML schema has certain advantages, like they are extensible to future additions. It is written in XML and hence it is richer and more useful than DTDs. Also, it supports data types and namespaces.

2.2 Introduction to XQuery/XPath

XPath is a language for finding information in an XML document. It is used to navigate through elements and attributes in an XML document. XPath is a syntax for defining parts of an XML document it uses path expressions to navigate in XML documents and contains a library of standard functions. In XPath, there are seven kinds of nodes: element, attribute, text, namespace, processing-instruction, comment, and document (root) nodes. XML documents are treated as trees of nodes. The root of the tree is called the document node (or root node). XPath uses path expressions to select nodes in an XML document. Predicates are used to find a specific node or a node that contains a specific value. Predicates are always embedded in square brackets.

XQuery is designed to query XML data, not just XML files, but anything that might appear as XML, including databases. It is built on XPath expressions and

supported by all major database vendors like IBM, Oracle, Microsoft, etc. It can be used to extract information to use in a web service, to generate summary reports, to transform XML data to XHTML, and to search web documents for relevant information.

CHAPTER 3
BASIC UPDATE OPERATIONS ON A DOCUMENT

This chapter describes the basic update operations on a document, the operations discussed are: insert, delete, rename and replace. The update primitives for Insert operation have been discussed in detail. Use cases provide the actual implementation of the update algebra and cover variety of update scenarios.

3.1 Insert Operation

3.1.1 Definition

The Insert operation inserts copies of one or more nodes into a designated position in a document instance. The Insert operation is an updating operation. It has the following categories:

- Insert Into – Inserts copies of one or more nodes into a document. The position of the inserted nodes within their parent is implementation-dependent.
- Insert Before – Inserts copies of one or more nodes into a document before the target expression.
- Insert After – Inserts copies of one or more nodes into a document after the target expression.
- Insert Into as First – Inserts copies of one or more nodes into a document as the first expression.
- Insert Into as Last – Inserts copies of one or more nodes into a document as the last expression.

3.1.2 Symbolic Notation

Insert Into – **I**

Insert Before – $\mathbf{I_B}$

Insert After – $\mathbf{I_A}$

Insert Into as First – $\mathbf{I_F}$

Insert Into as Last – $\mathbf{I_L}$

3.1.3 Usage

I *<source expression>* (D) *<target expression>*

$\mathbf{I_B}$ *<source expression>* (D) *<target expression>*

$\mathbf{I_A}$ *<source expression>* (D) *<target expression>*

$\mathbf{I_F}$ *<source expression>* (D) *<target expression>*

$\mathbf{I_L}$ *<source expression>* (D) *<target expression>*

Where 'D' is the name of document.

3.1.4 General Constraints

- The *<source expression>* must not be an updating expression. The source expression is a sequence of nodes to be inserted.
- The *<target expression>* must not be an updating expression. The target expression is evaluated. If *into* is specified the result must be a single element node or a single document node. If *before* or *after* is specified, the result must be a single element node.

EMPLOYEE (ENO, ENAME, TITLE)
PROJECT (PNO, PNAME, START_DATE, BUDGET)
WORKS (ENO, PNO, RESP, DUR)

Figure 3.1 Relational database schema.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <xs:import namespace="http://www.w3.org/XML/1998/namespace"/>
    <xs:element name="employee">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="employee_tuple" minOccurs="0"
maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="employee_tuple">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="eno"/>
                <xs:element ref="ename"/>
                <xs:element ref="title"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="eno">
        <xs:complexType mixed="true">
            <xs:choice minOccurs="0" maxOccurs="unbounded"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="ename">
        <xs:complexType mixed="true">
            <xs:choice minOccurs="0" maxOccurs="unbounded"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="title">
        <xs:complexType mixed="true">
            <xs:choice minOccurs="0" maxOccurs="unbounded"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Figure 3.2 XML schema for employee.xml.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xs:schema                                    xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
        <xs:import namespace="http://www.w3.org/XML/1998/namespace"/>
        <xs:element name="project">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element          ref="project_tuple"          minOccurs="0"
maxOccurs="unbounded"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="project_tuple">
                <xs:complexType>
                        <xs:sequence>
                                <xs:element ref="pno"/>
                                <xs:element ref="pname"/>
                                <xs:element ref="start_date" minOccurs="0"/>
                                <xs:element ref="budget"/>
                        </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="pno">
                <xs:complexType mixed="true">
                        <xs:choice minOccurs="0" maxOccurs="unbounded"/>
                </xs:complexType>
        </xs:element>
        <xs:element name="pname">
                <xs:complexType mixed="true">
                        <xs:choice minOccurs="0" maxOccurs="unbounded"/>
                </xs:complexType>
        </xs:element>
        <xs:element name="start_date">
                <xs:complexType mixed="true">
                        <xs:choice minOccurs="0" maxOccurs="unbounded"/>
                </xs:complexType>
        </xs:element>
        <xs:element name="budget">
                <xs:complexType mixed="true">
                        <xs:choice minOccurs="0" maxOccurs="unbounded"/>
                </xs:complexType>
        </xs:element>
</xs:schema>
```

Figure 3.3 XML schema for project.xml.

3.1.5 Use Cases

This use case is based on three separate input documents named employee.xml, project.xml, and works.xml. Each of the documents represents one of the tables in the relational database described in Figure 3.1, using the XML schema in Figures 3.2, 3.3 and 3.4. The instances of the relational table are shown in Tables 3.1, 3.2 and 3.3. Figure 3.5 contains an abbreviated set of data showing the XML format of instances.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
	<xs:import namespace="http://www.w3.org/XML/1998/namespace"/>
	<xs:complexType name="works">
		<xs:sequence>
			<xs:element ref="works_tuple" minOccurs="0" maxOccurs="unbounded"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="works" type="works"/>
	<xs:complexType name="works_tuple">
		<xs:sequence>
			<xs:element ref="eno"/>
			<xs:element ref="pno"/>
			<xs:element ref="resp"/>
			<xs:element ref="dur"/>
		</xs:sequence>
	</xs:complexType>
	<xs:element name="works_tuple" type="works_tuple"/>
	<xs:complexType name="eno" mixed="true">
		<xs:choice minOccurs="0" maxOccurs="unbounded"/>
	</xs:complexType>
	<xs:element name="eno" type="eno"/>
	<xs:complexType name="pno" mixed="true">
		<xs:choice minOccurs="0" maxOccurs="unbounded"/>
	</xs:complexType>
	<xs:element name="pno" type="pno"/>
	<xs:complexType name="resp" mixed="true">
		<xs:choice minOccurs="0" maxOccurs="unbounded"/>
	</xs:complexType>
	<xs:element name="resp" type="resp"/>
	<xs:complexType name="dur" mixed="true">
		<xs:choice minOccurs="0" maxOccurs="unbounded"/>
	</xs:complexType>
	<xs:element name="dur" type="dur"/>
</xs:schema>
```

Figure 3.4 XML schema for works.xml.

```
<employee>
        <employee_tuple>
                <eno>E1</eno>
                <ename>G. Elmasry</ename>
                <etitle>Electrical Engineer</etitle>
        </employee_tuple>
        <!-- !!! Snip !!! -->

<project>
        <project_tuple>
                <pno>P1</pno>
                <pname>Data Quality Management</pname>
                <start_date>05/01/2004</start_date>
                <budget>500000</budget>
        </project_tuple>
        <!-- !!! Snip !!! -->

<works>
        <works_tuple>
                <eno>E1</eno>
                <pno>P1</pno>
                <resp>Manager</resp>
                <dur>12</dur>
        </works_tuple>
        <!-- !!! Snip !!! --!>
```

Figure 3.5 Abbreviated set of data showing the XML format of the instances for employee.xml, project.xml and works.xml.

Table 3.1 Snapshot of Employee Table

ENO	ENAME	ETITLE
E1	G. Elmasry	Electrical Engineer
E2	C. Peckham	System Analyst
E3	H. Garcia	Mechanical Engineer
E4	P. Petrov	Programmer
E5	A. Flynn	System Analyst
E6	H. Dobbs	Electrical Engineer
E7	I. Kilpatrick	Mechanical Engineer
E8	M. Roman	System Analyst

3.1.5.1 Add a New Project (with No Start Date) to project.xml.

Source Expression:

src_expr ← {<project_tuple>

<pno>P6</pno>

<pname>ERP</pname>

<budget>250000</budget>

</project_tuple>}

Target Expression:

trg_expr ← doc("project.xml")/project

Update Operation:

I $_{src\_expr}$ (project.xml) $_{trg\_expr}$

Expected resulting content of project.xml:

```
<project>
        <project_tuple>
                <pno>P1</pno>
                <pname>Data Quality Management</pname>
                <start_date>05/01/2004</start_date>
                <budget>500000</budget>
        </project_tuple>
        <!-- !!! Snip !!! -->
        <project_tuple>
                <pno>P6</pno>
                <pname>ERP</pname>
                <budget>250000</budget>
        </project_tuple>
</project>
```

Table 3.2 Snapshot of Project Table

PNO	PNAME	START_DATE	BUDGET
P1	Data Quality Management	05/01/2004	500000
P2	Instrumentation	06/01/2003	200000
P3	Maintenance	03/15/2005	100000
P4	Database Development	07/12/2005	350000
P5	CAD/CAM	02/01/2006	600000

Table 3.3 Snapshot of Works Table

ENO	PNO	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	26
E7	P5	Engineer	23
E8	P3	Manager	40

3.1.5.2 Add a New Employee as First to employee.xml.

Source Expression:

src_expr ← {<employee_tuple>

<eno>E9</eno>

<ename>A. Petrock</ename>

<etitle>Accountant</etitle>

</employee_tuple>}

Target Expression:

trg_expr ← doc("employee.xml")/employee

Update Operation:

$I_{F\ src\_expr}$ (employee.xml) $_{trg\_expr}$

Expected resulting content of employee.xml:

```
<employee>
        <employee_tuple>
                <eno>E9</eno>
                <ename>A. Petrock</ename>
                <etitle>Accountant</etitle>
        </employee_tuple>

        <employee_tuple>
                <eno>E1</eno>
                <ename>G. Elmasry</ename>
                <etitle>Electrical Engineer</etitle>
        </employee_tuple>
        <!-- !!! Snip !!! -->
```

3.1.5.3 Insert a New Project in project.xml After Tuple with Project Name "Database Development". Project Number is P6, Project Name is "Datawarehousing", Start Date as 09/10/2005 and Budget as 150000.

Source Expression:

src_expr ← {<project_tuple>

<pno>P6</pno>

<pname>Datawarehousing</pname>

<start_date>09/10/2005</start_date>

<budget>150000</budget>

</project_tuple>}

Target Expression:

trg_expr ← doc("project.xml")/project/project_tuple[pname="Database Development"]

Update Operation:

$I_{A\ src\_expr}$ (project.xml) $_{trg\_expr}$

Expected resulting content of project.xml:

```
        <!-- !!! Snip !!! -->
        <project_tuple>
                <pno>P4</pno>
                <pname>Database Development</pname>
                <start_date>07/12/2005</start_date>
                <budget>350000</budget>
        </project_tuple>
        <project_tuple>
                <pno>P6</pno>
                <pname>Datawarehousing</pname>
                <start_date>09/10/2005</start_date>
                <budget>150000</budget>
        </project_tuple>
        <!-- !!! Snip !!! -->
</project>
```

3.1.6 Algebraic Laws

The Insert operation is non-commutative and non-associative for a document where left to right order of nodes within the tree representation of the document is significant.

3.1.7 Primitive Insert Operations

The primitive insert operations are implemented internally, they are not directly available to users.

3.1.7.1 Insert Before (I_b). Insert Before inserts source node immediately after target node.

Symbolic Notation: I_b <source_node> (D) <target_node>

Constraints: Target node must be an element, text, processing instruction, or comment node. Source node must be a sequence containing only element, text, processing

instruction, and comment nodes.

Effects on nodes in source and target:

- For each node in source, the parent property is set to parent of target.
- The children property of the parent of target node is modified to add the nodes in source just before target, preserving their order.
- If, as a result of the previous step, the children property of the parent of target contains adjacent text nodes, these adjacent text nodes are merged into a single text node.

3.1.7.2 Insert After (I_a). Insert After inserts source node immediately after target node.

Symbolic Notation: I_a <source_node> (D) <target_node>

Constraints: Target node must be an element, text, processing instruction, or comment node. Source node must be a sequence containing only element, text, processing instruction, and comment nodes.

Effects on nodes in source and target:

- For each node in source, the parent property is set to parent of target.
- The children property of the parent of target node is modified to add the nodes in source just after target, preserving their order.
- If, as a result of the previous step, the children property of the parent of target contains adjacent text nodes, these adjacent text nodes are merged into a single text node.

3.1.7.3 Insert Into (I). Insert Into inserts source node as children of the target node, in an implementation-defined position.

Symbolic Notation: I <source_node> (D) <target_node>

Constraints: Target node must be an element or document node. Source node must be a sequence containing only element, text, processing instruction, and comment nodes.

Effects on nodes in source and target:

- For each node in source, the parent property is set to parent of target.
- The children property of the target node is changed to include the nodes in the source node. The order among the children and their position within the parent node is implementation-dependent.
- If, as a result of the previous step, the children property of the parent of target contains adjacent text nodes, these adjacent text nodes are merged into a single text node.

3.1.7.4 Insert Into as Last (I_l). Insert Into as Last inserts source node as the last children of target node.

Symbolic Notation: I_l <source_node> (D) <target_node>

Constraints: Target node must be an element or document node. Source node must be a sequence containing only element, text, processing instruction, and comment nodes.

Effects on nodes in source and target:

- For each node in source, the parent property is set to parent of target.
- The children property of the target node is modified to include the nodes in the source node as last children, preserving their order.
- If, as a result of the previous step, the children property of the parent of target contains adjacent text nodes, these adjacent text nodes are merged into a single text node.

3.1.7.5 Insert Attributes ($I_@$). Insert Attributes inserts source node as attributes of target node.

Symbolic Notation: $I_@$ <source_node> (D) <target_node>

Constraints: None

Effects on nodes in source and target:

- For each node in source, the parent property is set to target.
- The attributes of target node are modified to include the nodes in source.

3.2 Delete Operation

3.2.1 Definition and Symbolic Notation

Definition: The Delete operation deletes one or more nodes from a document instance. It is an updating operation.

Symbolic Notation: X

3.2.2 Usage

X $_{<target\_expression>}$ (D)

Where 'D' is the name document.

3.2.3 General Constraint

The target expression must not be an updating expression. The target expression is evaluated. The result must be a sequence of nodes.

3.2.4 Use Case

Delete all records for M. Roman from works.xml.

Target Expression:

$eid := doc("employee.xml")/employee/employee_tuple[ename="M. Roman"]/eid

trg_expr := doc("works.xml")/works/works_tuple[eno=$eid]

Update Operation:

X $_{<trg\_expr>}$ (works.xml)

Expected resulting content of works.xml:

```
        <!-- !!! Snip !!! --!>
<works>
        <works_tuple>
                <eno>E7</eno>
                <pno>P3</pno>
```

```
            <resp>Engineer</resp>
            <dur>26</dur>
        </works_tuple>
        <works_tuple>
            <eno>E7</eno>
            <pno>P5/pno>
            <resp>Engineer</resp>
            <dur>23</dur>
        </works_tuple>
<works>
```

3.2.5 Algebraic Laws

A **X** B = B **X** A

(A **X** B) **X** C = A **X** (B **X** C)

Delete operation is both commutative and associative.

3.3 Rename Operation

3.3.1 Definition and Symbolic Notation

Definition: The Rename operation replaces name property of the target expression with a new name. It is an updating operation.

Symbolic Notation: P

3.3.2 Usage

P *<new_name_expression>* (D) *<target_expression>*

3.3.3 General Constraints

- The *<target_expression>* must not be an updating expression. It is evaluated. The result must be a single element, attribute, or processing instruction node.
- The *<new_name_expression>* must not be an updating expression. It is evaluated.

3.3.4 Use Cases

3.3.4.1 Rename the eno Element of the First Employee to employee-number in employee.xml.

Target Expression:

trg_expr := doc("employee.xml")/employee/employee_tuple[1]/eno

Update Expression:

P $_{\text{"employee-number"}}$ (D) $_{\text{trg\_expr}}$

Expected resulting content of employee.xml:

```
<employee>
        <employee_tuple>
                <employee-number>E1</employee-number>
                <ename>G. Elmasry</ename>
                <etitle>Electrical Engineer</etitle>
        </employee_tuple>
        <!-- !!! Snip !!! -->
        <employee_tuple>
                <eno>E8</eno>
                <ename>M. Roman</ename>
                <etitle>System Analyst</etitle>
        </employee_tuple>
</employee>
```

3.3.4.2 Rename the pname Element of All Projects to project-name in project.xml that is the Value of the Variable $newname.

Target Expression:

trg_expr := doc("project.xml")/project/project_tuple/pname

Update Expression:

P $_{\{\$\text{newname}\}}$ (D) $_{\text{trg\_expr}}$

Expected resulting content of project.xml:

```
<project>
```

```
        <project_tuple>
                <pno>P1</pno>
                <project-name>Data Quality Management</project-name>
                <start_date>05/01/2004</start_date>
                <budget>500000</budget>
        </project_tuple>
        <!-- !!! Snip !!! -->
        <project_tuple>
                <pno>P5</pno>
                <project-name>CAD/CAM,/project-name>
                <start_date>02/01/2006</start_date>
                <budget>600000</budget>
        </project_tuple>
</project>
```

3.3.5 Algebraic Laws

A **P** B = B **P** A

(A **P** B) **P** C = A **P** (B **P** C)

Rename operation is both commutative and associative. It is assumed that A, B and C have distinct target expressions.

3.4 Replace Operation

3.4.1 Definition and Symbolic Notation

Definition: A Replace operation replaces a node or modifies the value of a node. A replace operation has two forms, depending on whether *value of* is specified. If the *value of* is specified then the value of the node is replaced, otherwise each replacement replaces one node with a new sequence of zero or more nodes. It is an updating operation. Replace can also be implemented by delete followed by insert operation.

Symbolic Notation: Γ

3.4.2 Usage

Γ *<source expression>* (D) *<target expression>*

Γ *<source expression>* (D) value_of*(<target expression>)*

3.4.3 General Constraints

The *<source expression>* must not be an updating expression. The source expression is evaluated and the result is a sequence of nodes called the replacement sequence. The *<target expression>* must not be an updating expression. It is evaluated and the result should be a single node. If value of is specified, a replace expression is used to modify the value of a node while preserving its node identity.

3.4.4 Use Cases

3.4.4.1 Replace name A. Flynn with A. Reinhardt in employee.xml.

Target Expression:

trg_expr := doc("employee.xml")/employee/employee_tuple[ename="A. Flynn"]/ename

Update Expression:

Γ "A. Reinhardt" (D) value_of(trg_expr)

Expected resulting content of employee.xml:

```
</employee>
        <!-- !!! Snip !!! -->
        <employee_tuple>
                <eno>E5</eno>
                <ename>A. Reinhardt</ename>
                <etitle>System Analyst</etitle>
        </employee_tuple>
        <!-- !!! Snip !!! -->
        <employee_tuple>
                <eno>E6</eno>
                <ename>H. Dobbs</ename>
                <etitle>Electrical Engineer</etitle>
```

```
        </employee_tuple>
</employee>
```

3.4.4.2 Replace Budget for Project Name "Database Development" with Budget for Project Number "P1" in project.xml.

Source Expression:

src_expr := doc("project.xml")/project/project_tuple[pno="P1"]/budget

Target Expression:

trg_expr := doc("project.xml")/project/project_tuple[pname="Database Development"]/budget

Update Expression:

Γ_{src_expr} (project.xml) $_{trg\_expr}$

Expected resulting content of employee.xml:

```
<project>
        <project_tuple>
                <pno>P1</pno>
                <pname>Data Quality Management</pname>
                <start_date>05/01/2004</start_date>
                <budget>500000</budget>
        </project_tuple>
        <!-- !!! Snip !!! -->
        <project_tuple>
                <pno>P4</pno>
                <pname>Database Development</pname>
                <start_date>07/12/2005</start_date>
                <budget>500000</budget>
        </project_tuple>
        <!-- !!! Snip !!! -->
```

3.4.5 Algebraic Laws

A ⌐ B = B ⌐ A

(A ⌐ B) ⌐ C = A ⌐ (B ⌐ C)

Replace operation is both commutative and associative. It is assumed that A, B and C have distinct target expressions.

CHAPTER 4
ADVANCED RECOMPOSITION OPERATIONS ON DOCUMENTS

4.1 Project Operation

4.1.1 Definition and Symbolic Notation

Definition: Project operation projects on nodes of a document. It retrieves a node or a subtree to form a new document having the root element of the source document. The resulting document contains the nodes in order they are specified in the expression list.

Symbolic Notation: Π

4.1.2 Usage

Π *<expression list>* (D)

Where 'D' is the name of source document.

4.1.3 General Constraints

The *<expression list>* is a list of expressions that must be evaluated to get the list of nodes to be projected. If a condition is specified in the expression then the nodes must be filtered according to the condition. In case no predicate is specified and there exists more than one node with the same name then all nodes along with their corresponding subtree must be included in the result document.

4.1.4 Use Case

All use cases in this chapter are based on three xml documents named book1.xml, book2.xml and book3.xml shown in Figures 4.1, 4.2 and 4.3 respectively the schema is given in Figure 4.4.

```
<ebook>
    <title>RFID Essentials</title>
    <author>George Prescott</author>
    <publisher>O Really</publisher>
    <year>2004</year>
    <price>105.95</price>
    <overview>Radio Frequency Identification (RFID) is rapidly changing the way businesses track inventory and assets. From Wal-Mart and Tesco to the tment of Defense, early efforts are already showing benefits, but software, integration, and data processing for RFID still present a challenge.</overview>
    <chapter_1>
        <title>An Introdution to RFID</title>
        <content>RFID technologies offer practical benefits to almost anyone who needs to keep track of physical assets. Manufacturers improve supply-chain planning and execution by incorporating RFID technologies. Retailers use RFID to control theft, increase efficiency in their supply chains, and improve demand planning.</content>
    </chapter_1>
    <chapter_2>
        <title>RFID Architecture</title>
        <content>For our purposes, an architecture may be defined as a decomposition of a particular computer system into individual components to show how the components work together to meet the requirements for the entire system. With this definition in mind, we can confidently say that there is no such thing as a single, universal RFID architecture that fits all requirements for all systems.</content>
    </chapter_2>
    <chapter_3>
        <title>RFID Information Service</title>
        <content>One of the promises of RFID is that business partners will be able to automatically collect and share up-to-the-minute tracking information about items in their supply chains. To realize this benefit, businesses need to agree on what information will be collected (and its semantics), when and how this information will be collected, where and how it will be stored, and, finally, where and how to access it.</content>
    </chapter_3>
    <conclusion>This is conclusion to the RFID book.</conclusion>
</ebook>
```

Figure 4.1 XML document - Book1.xml.

```
<ebook>
    <title>Leo Laporte PC Help Desk</title>
    <author>Leo Laporte</author>
    <publisher>Que</publisher>
    <year>2005</year>
    <price>65</price>
    <overview>Nurse your PC back to health with a little help from Leo Laporte. Leo
Laporte PC Help Desk in a Book uses a unique, medical dictionary approach, complete
with symptoms, diagnosis, and treatment for all of your common and not-so-common PC
maladies.</overview>
    <chapter_1>
        <title>PC Anatomy</title>
        <content>This chapter introduces you to the major components you will
find in typical computers, including those prone to being a "point of failure." Think of
this as an anatomy lesson, but without the formaldehyde and nasty smells.</content>
    </chapter_1>
    <chapter_2>
        <title>Troubleshooting Storage Devices</title>
        <content>If you are reading this book after your old hard disk has been
packed off to the manufacturer for replacement, we can still help. You have got to get the
new one installed, so let us help you with that process, too.</content>
    </chapter_2>
    <chapter_3>
        <title>Troubleshooting Your Printer</title>
        <content>The troubleshooting tips and methods in this chapter apply
equally well to standalone printers and the printer portion of an all-in-one (multifunction)
device.</content>
    </chapter_3>
    <conclusion>This is conclusion to the PC Help Desk book.</conclusion>
</ebook>
```

Figure 4.2 XML document – Book2.xml.

```
<ebook>
        <title>Digital Photography</title>
        <author>Derrick Story</author>
        <publisher>Thompson Press</publisher>
        <year>2003</year>
        <price>75.95</price>
        <overview>Going beyond the standard fare of most digital photography books,
this shares the knowledge that professional photographers have learned through
thousands of shots worth of experience and years of experimentation.</overview>
        <chapter_1>
                <title>Digital Camera Attachments</title>
                <content>Digicams are good for more than just hanging around your neck.
You have a wealth of accessories available to expand their capability. The threaded
socket on the bottom enables you to secure your camera to a variety of unique stabilizing
devices.</content>
        </chapter_1>
        <chapter_2>
                <title>Daytime Photo Secrets</title>
                <content>Photography requires daytime light. And the best place to find
light is outdoors. It is cheap, abundant, and, at times, stunningly beautiful. Indeed, this is
the appropriate place for us to begin the hacks on shooting technique.</content>
        </chapter_2>
        <chapter_3>
                <title>The Computer Connection</title>
                <content>To really appreciate the power of your digital camera, you have
to plug it into a computer. This is where you turn average photos into a great ones, create
glorious prints that used to take days to return from the photo lab, make digital
slideshows that rival professional presentations, paste together video snippets into short
movies, and even add voice and music to your images.</content>
        </chapter_3>
        <conclusion>This is conclusion to Digital Photography.</conclusion>
</ebook>
```

Figure 4.3 XML document – Book3.xml.

4.1.4.1 Project on Title, Author and Publisher from book1.xml

Target Expression:

trg_expr 1 ← doc("book1.xml")/ebook/title

trg_expr 2 ← doc("book1.xml")/ebook/author

trg_expr 3 ← doc("book1.xml")/ebook/publisher

Recomposition Operation:

$\Pi_{\{trg\_expr\ 1,\ trg\_expr\ 2,\ trg\_expr\ 3\}}$ (book1.xml)

Expected result of query:

```
<ebook>
        <title>RFID Essentials</title>
        <author>George Prescott</author>
        <publisher>O Really</publisher>
</ebook>
```

4.1.5 Derivability

This operation can also be implemented by copying target nodes from the source document and then inserting them as last in the new document.

4.1.6 Algebraic Laws

Commutativity:
Project(Project(Doc, A), B) ≠ Project(Project(Doc, B), A)

Associativity:
Project(Project(Project(Doc, A), B), C) ≠ Project(Project(Project(Doc, B), C), A)

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
	<xs:element name="content" type="xs:string"/>
	<xs:element name="ebook">
		<xs:complexType>
			<xs:sequence>
				<xs:element ref="title"/>
				<xs:element name="author" type="xs:string"/>
				<xs:element name="publisher" type="xs:string"/>
				<xs:element name="year" type="xs:string"/>
				<xs:element name="price" type="xs:string"/>
				<xs:element name="overview" type="xs:string"/>
				<xs:element name="chapter_1">
					<xs:complexType>
						<xs:sequence>
							<xs:element ref="title"/>
							<xs:element ref="content"/>
						</xs:sequence>
					</xs:complexType>
				</xs:element>
				<xs:element name="chapter_2">
					<xs:complexType>
						<xs:sequence>
							<xs:element ref="title"/>
							<xs:element ref="content"/>
						</xs:sequence>
					</xs:complexType>
				</xs:element>
				<xs:element name="chapter_3">
					<xs:complexType>
						<xs:sequence>
							<xs:element ref="title"/>
							<xs:element ref="content"/>
						</xs:sequence>
					</xs:complexType>
				</xs:element>
				<xs:element name="conclusion" type="xs:string"/>
			</xs:sequence>
		</xs:complexType>
	</xs:element>
	<xs:element name="title" type="xs:string"/>
</xs:schema>
```

Figure 4.4 XML schema for Book1.xml, Book2.xml and Book3.xml.

4.2 Merge Operation

4.2.1 Definition and Symbolic Notation

Definition: Merge operation merges the contents of two documents and returns a new document. The order of nodes in the new document is determined by the order in expression list. The root element of the result document is specified in the operation.

Symbolic Notation: U

4.2.2 Usage

U *<expression list 1><expression list 2>* (D_1, D_2)*<root tag>*

Where $D_1$and D_2 are source documents.

4.2.3 General Constraints

The *<expression list 1>* and *<expression list 2>* are lists of expressions that must be evaluated to get the list of nodes to be merged. If a condition is specified in the expression then nodes must be filtered according to the condition. In case no predicate is specified and there exists more than one node with the same name then all nodes along with their corresponding subtree must be included in the result document. The *<root tag>* is the name of the root node for new document. In case any of the expression list is empty then all nodes in the document are merged.

4.2.4 Use Case

4.2.4.1 Merge Title, Author, Publisher from book1.xml, and Price and Overview from book2.xml.

Target Expression:

trg_expr 1 ← doc("book1.xml")/ebook/title

trg_expr 2 ← doc("book1.xml")/ebook/author

trg_expr 3 ← doc("book1.xml")/ebook/publisher

trg_expr 4 ← doc("book2.xml")/ebook/price

trg_expr 5 ← doc("book2.xml")/ebook/overview

Recomposition Operation:

$\mathbf{U}_{\{trg\_expr\ 1,\ trg\_expr\ 2,\ trg\_expr\ 3\}\ \{trg\_expr\ 4,\ trg\_expr\ 5\}}$ (book1.xml, book2.xml) $_{(\text{"ebook"})}$

Expected result of query:

```
<ebook>
        <title>RFID Essentials</title>
        <author>George Prescott</author>
        <publisher>O Really</publisher>

</ebook>
```

4.2.5 Derivability

This operation can also be implemented by creation of an empty document, then insertion of root element followed by copy of nodes from the two source document and then insertion as last in the new document.

4.2.6 Algebraic Laws

Commutativity: Merge(Doc1, Doc2) ≠ Merge(Doc2, Doc1)

Associativity: Merge(Merge(Doc1, Doc2), Doc3) ≠ Merge(Doc1, Merge(Doc2, Doc3))

4.3 Move Operation

4.3.1 Definition and Symbolic Notation

Definition: Move operation moves nodes or subtrees from the source document to the target document. The source nodes are appended at the end of the target document as children of the root node.

Symbolic Notation: μ

4.3.2 Usage

$\mu_{<source\ expression\ list>}(D_1, D_2)$

Where D_1 is the source document and D_2 is the target document.

4.3.3 General Constraints

The *<source expression list>* is a list of expressions that must be evaluated to get the list of nodes to be moved. The source nodes are removed from the source document.

4.3.4 Use Case

4.3.4.1 Move chapter_3 from book2.xml to book3.xml, Rename it to chapter_4.

Target Expression:

trg_expr ← doc("book2.xml")/ebook/chapter_3

Recomposition Operation:

$\mu_{\{trg\_expr\}}$ (book2.xml, book3.xml)

Expected resulting content of book2.xml:

```
<ebook>
        <title>Leo Laporte PC Help Desk</title>
        <author>Leo Laporte</author>
        <publisher>Que</publisher>
        <year>2005</year>
```

```
    <price>65</price>
    <overview>Nurse your PC back to health with a little help from Leo Laporte. Leo Laporte PC Help Desk in a Book uses a unique, medical dictionary approach, complete with symptoms, diagnosis, and treatment for all of your common and not-so-common PC maladies.</overview>
    <chapter_1>
        <title>PC Anatomy</title>
        <content>This chapter introduces you to the major components you will find in typical computers, including those prone to being a "point of failure." Think of this as an anatomy lesson, but without the formaldehyde and nasty smells.</content>
    </chapter_1>
    <chapter_2>
        <title>Troubleshooting Storage Devices</title>
        <content>If you are reading this book after your old hard disk has been packed off to the manufacturer for replacement, we can still help. You have got to get the new one installed, so let us help you with that process, too.</content>
    </chapter_2>
    <conclusion>This is conclusion to the PC Help Desk book.</conclusion>
</ebook>
```

Expected resulting content of book3.xml:

```
<ebook>
    <title>Digital Photography</title>
    <author>Derrick Story</author>
    <publisher>Thompson Press</publisher>
    <year>2003</year>
    <price>75.95</price>
    <overview>Going beyond the standard fare of most digital photography books, this shares the knowledge that professional photographers have learned through thousands of shots worth of experience and years of experimentation.</overview>
    <chapter_1>
        <title>Digital Camera Attachments</title>
        <content>Digicams are good for more than just hanging around your neck. You have a wealth of accessories available to expand their capability. The threaded socket on the bottom enables you to secure your camera to a variety of unique stabilizing devices.</content>
    </chapter_1>
    <chapter_2>
        <title>Daytime Photo Secrets</title>
        <content>Photography requires daytime light. And the best place to find light is outdoors. It is cheap, abundant, and, at times, stunningly beautiful. Indeed, this is the appropriate place for us to begin the hacks on shooting technique.</content>
    </chapter_2>
```

<chapter_3>
<title>The Computer Connection</title>
<content>To really appreciate the power of your digital camera, you have to plug it into a computer. This is where you turn average photos into a great ones, create glorious prints that used to take days to return from the photo lab, make digital slideshows that rival professional presentations, paste together video snippets into short movies, and even add voice and music to your images.</content>
</chapter_3>
<chapter_4>
<title>Troubleshooting Your Printer</title>
<content>The troubleshooting tips and methods in this chapter apply equally well to standalone printers and the printer portion of an all-in-one (multifunction) device.</content>
</chapter_4>
<conclusion>This is conclusion to Digital Photography.</conclusion>
</ebook>

4.3.5 Derivability

This operation can also be implemented by copying the nodes from the source document, inserting them into the target document followed by deletion of nodes from the source document.

4.3.6 Algebraic Laws

Commutativity:

Move(Move(A, Doc1, Doc2), B, Doc1, Doc2) ≠ Move(Move(B, Doc1, Doc2), A, Doc1, Doc2)

Associativity:

Move(Move(Move(A, Doc1, Doc2), B, Doc1, Doc2), C, Doc1, Doc2) ≠ Move(A, Doc1, Doc2(Move(Move(B, Doc1, Doc2), C, Doc1, Doc2))

4.4 Extract Operation

4.4.1 Definition and Symbolic Notation

Definition: Extract operation extracts nodes or a subtree from a document. It retrieves a node or a subtree to form a new document having the root element of the source document. The resulting document contains the nodes in order they are specified in the expression list.

Symbolic Notation: Φ

4.4.2 Usage

$\Phi_{<expression\ list>}(D)$

Where D is the source document.

4.4.3 General Constraints

The *<expression list>* is a list of expressions that must be evaluated to get the list of nodes to be extracted. If a condition is specified in the expression then the nodes must be filtered according to the condition. In case no predicate is specified and there exists more than one node with the same name then all nodes along with their corresponding subtree must be included in the result document. Extracted nodes are removed from the source document.

4.4.4 Use Case

4.4.4.1 Extract Overview, chapter_1 and Conclusion from book3.xml.

Target Expression:

trg_expr 1 ← doc("book3.xml")/ebook/overview

trg_expr 2 ← doc("book3.xml")/ebook/chapter_1

trg_expr 3 ← doc("book3.xml")/ebook/conclusion

Recomposition Operation:

$\Phi_{\{trg\_expr\ 1,\ trg\_expr\ 2,\ trg\_expr\ 3\}}$ (book3.xml)

Expected resulting content of new document:

```
<ebook>
        <overview>Going beyond the standard fare of most digital photography books,
this shares the knowledge that professional photographers have learned through
thousands of shots worth of experience and years of experimentation.</overview>
        <chapter_1>
                <title>Digital Camera Attachments</title>
                <content>Digicams are good for more than just hanging around your neck.
You have a wealth of accessories available to expand their capability. The threaded
socket on the bottom enables you to secure your camera to a variety of unique stabilizing
devices.</content>
        </chapter_1>
        <conclusion>This is conclusion to Digital Photography.</conclusion>
</ebook>
```

Expected resulting content of book3.xml:

```
<ebook>
        <title>Digital Photography</title>
        <author>Derrick Story</author>
        <publisher>Thompson Press</publisher>
        <year>2003</year>
        <price>75.95</price>
        <chapter_2>
                <title>Daytime Photo Secrets</title>
                <content>Photography requires daytime light. And the best place to find
light is outdoors. It is cheap, abundant, and, at times, stunningly beautiful. Indeed, this is
the appropriate place for us to begin the hacks on shooting technique.</content>
        </chapter_2>
        <chapter_3>
                <title>The Computer Connection</title>
                <content>To really appreciate the power of your digital camera, you have
to plug it into a computer. This is where you turn average photos into a great ones, create
glorious prints that used to take days to return from the photo lab, make digital
slideshows that rival professional presentations, paste together video snippets into short
movies, and even add voice and music to your images.</content>
        </chapter_3>
```

</ebook>

4.4.5 Derivability

This operation can also be implemented by copying the nodes from the source document, inserting them into the target document followed by deletion of nodes from the source document.

4.4.6 Algebraic Laws

Commutativity:
Extract(Extract(Doc, A), B) ≠ Extract(Extract(Doc, B), A)

Associativity:
Extract(Extract(Extract(Doc, A), B), C) ≠ Extract(Extract(Extract (Doc, B), C), A)

CHAPTER 5
IMPLEMENTATION

5.1 Insert Operation

Query: Insert new element 'preface' before element 'overview' in book1.xml.

Solution:

src_expr ← {<preface>This is a preface to the RFID book…</preface>}

trg_expr ← doc("book1.xml")/ebook/overview

Insert Operation: $I_{B\ src\_expr}$ (book1.xml) $_{trg\_expr}$

Algorithm:

```
//L and R point to leftmost and rightmost node in doubly linked list, new node is to be
//inserted before M, LPTR and RPTR are the left and right links respectively, new is the
new node
        (Insertion into an empty list)
        If R = Null
        Then LPTR(new) ← RPTR(new) ← Null
                L ← R ← new
                Return
        (Left-most insertion)
        If M = L
        Then LPTR(new) ← Null
                RPTR(new) ← M
                LPTR(M) ← new
                Return
        (insert in middle)
                LPTR(new) ← LPTR(M)
                RPTR(new) ← M
                LPTR(M) ← new
                RPTR(LPTR(new)) ← new
                Return
```

The insert before on average takes O(n/2) for inserting an element in a doubly linked list.

Expected resulting content of book1.xml:

```
<ebook>
        <title>RFID Essentials</title>
        <author>George Prescott</author>
        <publisher>O Really</publisher>
        <year>2004</year>
        <price>105.95</price>
        <preface>This is a preface to the RFID book...</preface>
        <overview>
          Radio Frequency Identification (RFID) is rapidly changing the way
businesses track inventory and assets. From Wal-Mart and Tesco to the Department of
Defense, early efforts are already showing benefits, but software, integration, and data
processing for RFID still present a challenge.
        </overview>
    <!-- !!! Snip !!! -->
```

5.2 Delete Operation

Query: Delete from book1.xml (Figure 4.1) element 'chapter_1' and all its sub-elements.

Solution:

trg_expr := doc("book1.xml")/ebook/chapter_1

Delete Operation: $\mathbf{X}_{\langle trg\_expr \rangle}$ (book1.xml)

Algorithm:

```
//L and R point to leftmost and rightmost node in doubly linked list, old is to be deleted
//LPTR and RPTR are the left and right links respectively
      function delete(Node L, Node R, Node old)
        If L = R (Single node in list)
        Then L ← R ← Null
        Else If old = L (Left-most node being deleted)
            Then L ← RPTR(L)
                    LPTR(L) ← Null
            Else if  old = R (Right-most node being deleted)
                    Then R ← LPTR(R)
                            RPTR(R) ← Null
                    Else RPTR(LPTR(old)) ← RPTR(old)
                            LPTR(RPTR(old)) ← LPTR(old)
            Return(old)
```

The delete on average takes O(n/2) for deleting an element from a linked list.

Expected resulting content of book1.xml:

<ebook>
<title>RFID Essentials</title>
<author>George Prescott</author>
<publisher>O Really</publisher>
<year>2004</year>
<price>105.95</price>
<overview>
Radio Frequency Identification (RFID) is rapidly changing the way businesses track inventory and assets. From Wal-Mart and Tesco to the tment of Defense, early efforts are already showing benefits, but software, integration, and data processing for RFID still present a challenge.
</overview>
<chapter_2>
<title>RFID Architecture</title>
<content>
For our purposes, an architecture may be defined as a decomposition of a particular computer system into individual components to show how the components work together to meet the requirements for the entire system. With this definition in mind, we can confidently say that there is no such thing as a single, universal RFID architecture that fits all requirements for all systems.
</content>
</chapter_2>
<!-- !!! Snip !!! -->

5.3 Replace Operation

Query: Update price to $90.95 in book1.xml.

Solution:

trg_expr := doc("book1.xml")/ebook/price

Replace Operation: $\Gamma_{\text{"90.95"}}$ (book1.xml) $_{\text{value\_of(trg\_expr)}}$

Algorithm:

//L and R point to leftmost and rightmost node in doubly linked list, old is to be replaced
//LPTR and RPTR are the left and right links respectively, new is the new node
function replace(Node L, Node R, Node old, Node new)

```
If old = L (Left-most node being replaced)
Then L ← RPTR(new)
Else if  old = R (Right-most node being replaced)
        Then R ← LPTR(new)
        Else RPTR(LPTR(old)) ← RPTR(new)
                LPTR(RPTR(old)) ← LPTR(new)
Return(old)
```

The replace on average takes O(n/2) for replacing an element from a linked list.

Expected resulting content of book1.xml:

```
<ebook>
        <title>RFID Essentials</title>
        <author>George Prescott</author>
        <publisher>O Really</publisher>
        <year>2004</year>
        <price>90.95</price>
    <!-- !!! Snip !!! -->
```

5.4 Rename Operation

Query: Rename element tag 'conclusion' as 'final_chapter' in book1.xml.

Solution:

trg_expr ← doc("book1.xml")/ebook/conclusion

Rename Operation: $\rho_{\text{"final\_chapter"}}(D)_{\text{trg\_expr}}$

Algorithm:

```
//L and R point to leftmost and rightmost node in doubly linked list, old is to be renamed
//LPTR and RPTR are the left and right links respectively, new is the new node
    function replace(Node L, Node R, Node old, Node new)
        If old = L (Left-most node being renamed)
        Then L ← RPTR(new)
        Else if  old = R (Right-most node being renamed)
                Then R ← LPTR(new)
                Else RPTR(LPTR(old)) ← RPTR(new)
                        LPTR(RPTR(old)) ← LPTR(new)
        Return(old)
```

The rename on average takes O(n/2) for renaming an element from a linked list.

Expected resulting content of book1.xml:

```
<ebook>
        <title>RFID Essentials</title>
        <author>George Prescott</author>
        <publisher>O Really</publisher>
        <year>2004</year>
    <!-- !!! Snip !!! -->
        <chapter_3>
          <title>RFID Information Service</title>
          <content>One of the promises of RFID is that business partners will be
able to automatically collect and share up-to-the-minute tracking information about items
in their supply chains. To realize this benefit, businesses need to agree on what
information will be collected (and its semantics), when and how this information will be
collected, where and how it will be stored, and, finally, where and how to access
it.</content>
        </chapter_3>
          <final_chapter>This is conclusion to the RFID book.</final_chapter>
</ebook>
```

5.5 Project Operation

Query: Project on title, author and price from book2.xml.

Solution:

trg_expr 1 ← doc("book2.xml")/ebook/title

trg_expr 2 ← doc("book2.xml")/ebook/author

trg_expr 3 ← doc("book2.xml")/ebook/price

Recomposition Operation:

$\Pi_{\{trg\_expr\ 1,\ trg\_expr\ 2,\ trg\_expr\ 3\}}$ (book2.xml)

Implementing Projection:

1. Create an empty document.
2. Insert a root node, the node should have the tag name of the root in source

document, use insert operation.

3. Traverse to the element node.
4. Copy element node and all its children one at a time into a document fragment.
5. Insert the fragment in the new document by using the insert operation as a child of root node.
6. If there is more than one subtree to project then insert each subtree at the end of the new document by following steps 3 to 5.

Expected result of query:

```
<ebook>
        <title>Leo Laporte PC Help Desk</title>
        <author>Leo Laporte</author>
        <price>65</price>
</ebook>
```

5.6 Merge Operation

Query: Merge book2.xml and book3.xml to form a new document New_Book.xml. New_Book.xml should contain from book2.xml: title, author, publisher, year and price; from book3.xml: overview, all chapters and conclusion. The root node tag should be ebook.

Solution:

trg_expr 1 ← doc("book2.xml")/ebook/title

trg_expr 2 ← doc("book2.xml")/ebook/author

trg_expr 3 ← doc("book2.xml")/ebook/publisher

trg_expr 4 ← doc("book2.xml")/ebook/year

trg_expr 5 ← doc("book2.xml")/ebook/price

trg_expr 6 ← doc("book3.xml")/ebook/overview

trg_expr 7 ← doc("book3.xml")/ebook/chapter_1

trg_expr 8 ← doc("book3.xml")/ebook/ chapter_2

trg_expr 9 ← doc("book3.xml")/ebook/ chapter_3

trg_expr 10 ← doc("book3.xml")/ebook/conclusion

Merge Operation:

$\mathbf{U}_{\{trg\_expr\ 1, \ldots, trg\_expr\ 5\}\ \{trg\_expr\ 6, \ldots, trg\_expr\ 10\}}$ (book2.xml, book3.xml) ("ebook")

Implementing Merge:

1. Create an empty document.
2. Insert a root node in the empty document as specified by user.
3. Copy each node or subtree one at a time from first source document and insert into the document fragment.
4. Insert the fragment in the new document by using the insert operation as a child of root node.
5. Copy each node or subtree one at a time from second source document and insert into the new document.
6. Insert the fragment in the new document by using the insert operation as a child of root node.
7. Return the new document.

Expected result of operation:

<ebook>
<title>Leo Laporte PC Help Desk</title>
<author>Leo Laporte</author>
<publisher>Que</publisher>
<year>2005</year>
<price>65</price>
<overview>Going beyond the standard fare of most digital photography books, this shares the knowledge that professional photographers have learned through thousands of shots worth of experience and years of experimentation.</overview>
<chapter_1>
<title>Digital Camera Attachments</title>
<content>Digicams are good for more than just hanging around your neck.

You have a wealth of accessories available to expand their capability. The threaded socket on the bottom enables you to secure your camera to a variety of unique stabilizing devices.</content>

</chapter_1>

<chapter_2>

<title>Daytime Photo Secrets</title>

<content>Photography requires daytime light. And the best place to find light is outdoors. It is cheap, abundant, and, at times, stunningly beautiful. Indeed, this is the appropriate place for us to begin the hacks on shooting technique.</content>

</chapter_2>

<chapter_3>

<title>The Computer Connection</title>

<content>To really appreciate the power of your digital camera, you have to plug it into a computer. This is where you turn average photos into a great ones, create glorious prints that used to take days to return from the photo lab, make digital slideshows that rival professional presentations, paste together video snippets into short movies, and even add voice and music to your images.</content>

</chapter_3>

<conclusion>This is conclusion to Digital Photography.</conclusion>

</ebook>

5.7 Move Operation

Query: Move 'chapter_3' from book1.xml to book2.xml.

Solution:

trg_expr ← doc("book1.xml")/ebook/chapter_3

Move Operation:

$\mu_{\{trg\_expr\}}$ (book1.xml, book2.xml)

Implementing Move:

1. Traverse to the node in source document.
2. Copy node and all its children one at a time into a document fragment.
3. Delete node and all its children from the source document.
4. Insert the fragment in the target document by using the insert operation, the node will be appended to the document.

5. If a sibling with the same name already exists then rename the inserted node.

6. If there is more than one node or subtree to move then insert each subtree at the end of the target document by following steps 1 to 5.

Expected resulting content of book1.xml:

<ebook>
<!-- !!! Snip !!! -->
<chapter_1>
<title>An Introdution to RFID</title>
<content>RFID technologies offer practical benefits to almost anyone who needs to keep track of physical assets. Manufacturers improve supply-chain planning and execution by incorporating RFID technologies. Retailers use RFID to control theft, increase efficiency in their supply chains, and improve demand planning.</content>
</chapter_1>
<chapter_2>
<title>RFID Architecture</title>
<content>For our purposes, an architecture may be defined as a decomposition of a particular computer system into individual components to show how the components work together to meet the requirements for the entire system. With this definition in mind, we can confidently say that there is no such thing as a single, universal RFID architecture that fits all requirements for all systems.</content>
</chapter_2>
<conclusion>This is conclusion to the RFID book.</conclusion>
</ebook>

Expected resulting content of book2.xml:

<ebook>
<!-- !!! Snip !!! -->
<chapter_2>
<title>Troubleshooting Storage Devices</title>
<content>If you are reading this book after your old hard disk has been packed off to the manufacturer for replacement, we can still help. You have got to get the new one installed, so let us help you with that process, too.</content>
</chapter_2>
<chapter_3>
<title>Troubleshooting Your Printer</title>
<content>The troubleshooting tips and methods in this chapter apply equally well to standalone printers and the printer portion of an all-in-one (multifunction) device.</content>
</chapter_3>
<conclusion>This is conclusion to the PC Help Desk book.</conclusion>
<chapter_3>

<title>RFID Information Service</title>
<content>One of the promises of RFID is that business partners will be able to automatically collect and share up-to-the-minute tracking information about items in their supply chains. To realize this benefit, businesses need to agree on what information will be collected (and its semantics), when and how this information will be collected, where and how it will be stored, and, finally, where and how to access it.</content>
</chapter_3>
</ebook>

5.8 Extract Operation

Query: Extract title, author, publisher, year and price from book3.xml.

Solution:

trg_expr 1 ← doc("book3.xml")/ebook/title

trg_expr 2 ← doc("book3.xml")/ebook/author

trg_expr 3 ← doc("book3.xml")/ebook/publisher

trg_expr 3 ← doc("book3.xml")/ebook/year

trg_expr 3 ← doc("book3.xml")/ebook/price

Extract Operation:

$\Phi_{\{trg\_expr\ 1,\ trg\_expr\ 2,\ trg\_expr\ 3\}}$ (book3.xml)

Implementing Extract:

1. Create an empty document.
2. Insert a root node, the node should have the tag name of the root in source document, use insert operation.
3. Traverse to the element node.
4. Copy element node and all its children one at a time into a document fragment.
5. Delete the node or the subtree in source document.

6. Insert the fragment in the new document by using the insert operation as a child of root node.

7. If there is more than one subtree to extract then insert each subtree at the end of the new document by following steps 3 to 6.

Expected result of operation:

```
<ebook>
        <title>Digital Photography</title>
        <author>Derrick Story</author>
        <publisher>Thompson Press</publisher>
        <year>2003</year>
        <price>75.95</price>
</ebook>'
```

5.9 Multiple Operations on Distributed Documents

Query: Project on title, author, publisher from Book1.xml
Project on year, price, overview from Book2.xml
Project on chapters from Book3.xml
Merge the projections into NewBook.xml
Replace contents of title by 'This is a Mixture of Three Books!'
Insert <language>English</language> before overview

Solution:

trg_expr 1 ← doc("book1.xml")/ebook/title

trg_expr 2 ← doc("book1.xml")/ebook/author

trg_expr 3 ← doc("book1.xml")/ebook/publisher

Project Operation: prj_1.xml ← $\Pi_{\{trg\_expr\ 1,\ trg\_expr\ 2,\ trg\_expr\ 3\}}$ (book1.xml)

trg_expr 4 ← doc("book2.xml")/ebook/year

trg_expr 5 ← doc("book2.xml")/ebook/price

trg_expr 6 ← doc("book2.xml")/ebook/overview

Project Operation: prj_2.xml ← $\Pi_{\{trg\_expr\ 4,\ trg\_expr\ 5,\ trg\_expr\ 6\}}$ (book2.xml)

trg_expr 7 ← doc("book3.xml")/ebook/chapter_1

trg_expr 8 ← doc("book3.xml")/ebook/chapter_2

trg_expr 9 ← doc("book3.xml")/ebook/chapter_3

Project Operation: prj_3.xml ← $\Pi_{\{trg\_expr\ 7,\ trg\_expr\ 8,\ trg\_expr\ 9\}}$ (book3.xml)

Merge Operation: new_doc.xml ← $\cup$ (prj_1.xml, prj_2.xml, prj_3.xml) ("ebook")

trg_expr ← doc("new_doc.xml")/ebook/title

Replace Operation: $\Gamma_{\text{"This is a Mixture of Three Books!"}}$ (new_doc.xml) $_{value\_of(trg\_expr)}$

src_expr ← {<language>English</language>}

trg_expr 10 ← doc("new_doc.xml")/ebook/overview

Insert Operation: $I_{B\ src\_expr}$ (new_doc.xml) $_{\{trg\_expr\ 10\}}$

Expected result:

```
<ebook>
  <title>This is a Mixture of Three Books!</title>
  <author>George Prescott</author>
  <publisher>O Really</publisher>
  <year>2005</year>
  <price>65</price>
  <language>English</language>
  <overview>Nurse your PC back to health with a little help from Leo Laporte. Leo
Laporte PC Help Desk in a Book uses a unique, medical dictionary approach, complete
with symptoms, diagnosis, and treatment for all of your common and not-so-common PC
maladies.</overview>
  <chapter_1>
    <title>Digital Camera Attachments</title>
    <content>Digicams are good for more than just hanging around your neck. You have
a wealth of accessories available to expand their capability. The threaded socket on the
bottom enables you to secure your camera to a variety of unique stabilizing
devices.</content>
  </chapter_1>
  <chapter_2>
    <title>Daytime Photo Secrets</title>
    <content>Photography requires daytime light. And the best place to find light is
outdoors. It is cheap, abundant, and, at times, stunningly beautiful. Indeed, this is the
appropriate place for us to begin the hacks on shooting technique.</content>
  </chapter_2>
```

```
  <chapter_3>
    <title>The Computer Connection</title>
    <content>To really appreciate the power of your digital camera, you have to plug it
into a computer. This is where you turn average photos into a great ones, create glorious
prints that used to take days to return from the photo lab, make digital slideshows that
rival professional presentation
s, paste together video snippets into short movies, and even add voice and music to your
images.</content>
  </chapter_3>
</ebook>
```

CHAPTER 6
CONCLUSION

In this thesis a set of operations for dynamic recomposition of documents is proposed. The source documents can be stored at distributed data sites and the process of dynamic recomposition can create the target document on-the-fly. The source documents are queried and results can be obtained as instances of XML document. The resultant document can then be stored statically or in form of a command, which can be invoked later to recompose this document dynamically. A solution for update and recomposition is proposed at the level of algebra. The issue of representation of a document by a command, i.e., a composition operator and/or an editing command along with one or more path expressions has also been studied in this research.

While it is felt that a fairly comprehensive study of document recomposition has been made, there are several potential areas for future work. The topic of optimization of operations is an important one. Also, development of a query parser will greatly ease the process of posing update and recomposition queries through an interactive user interface.

REFERENCES

1. Crowston, K., & Williams M. (1999). The Effects of Linking on Genres of Web Documents. *Proceedings of the Thirty-Second Annual Hawaii International Conference on System Sciences*. Maui, Hawaii.

2. Watters, C. (1999). Information Retrieval and the Virtual Document. *Journal of the American Society for Information.*

3. Zhang, L., Bieber, M., Millard, D., & Oria, V. (2004). Supporting Virtual Documents in Just-in-Time Hypermedia Systems. *Proceedings of the 2004 ACM symposium on Document engineering.*

4. Geneves, P., & Vion-Dury, J. (2004). Logic-Based XPath Optimization. *Proceedings of the 2004 ACM symposium on Document engineering.*

5. Caumanns, J. (1999). A Modular Framework for the Creation of Dynamic Documents. *Workshop on "Virtual Documents Hypertext Functionality and the Web" of the 8th Intl World-Wide Web Conference.* Toronto, Canada.

6. Chamberlin, D., Florescu, D., & Robie, J. (2006). XQuery Update Facility, W3C Working Draft 27 January 2006. Retrieved May 15, 2006, from *http://www.w3.org/TR/2006/WD-xqupdate-20060127*

7. Chamberlin, D., & Robie, J. (2006). XQuery Update Facility Requirements. W3C Working Draft 3 June 2005. Retrieved May 15, 2006, from *http://www.w3.org/TR/xquery-update-requirements*

8. Manolescu, I., & Robie (2006). XQuery Update Facility Use Cases. W3C Working Draft 27 January 2006. Retrieved May 15, 2006, from *http://www.w3.org/TR/2006/WD-xqupdateusecases-20060127*

9. Shanmugasundaram, J., Tufte, K., He, G., Zhang, C., DeWitt, D., & Naughton, J. (1999). Relational Databases for Querying XML Documents: Limitations and Opportunities. *Proceedings of the International Conference on Very Large Databases*. (pp. 302-314)

10. Ranwez, S., & Crampes, M. Conceptual Documents and Hypertext Documents are two Different Forms of Virtual Document.

11. Tatarinov, I., Ives, Z., Halevy, A., & Weld, D. (2001). Updating XML. *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data.*

12. Muench, S. (2000). *Building Oracle XML Applications*. O'Reilly & Associates, Inc.

13. Sunderraman, R. (2004). *Oracle9i Programming A Primer*. Pearson Education, Inc.

14. XML Path Language (XPath). Retrieved May 15, 2006, from *http://www.w3.org/xpath*

15. XML Pointer Language (XPointer). Retrieved May 15, 2006, from *http://www.w3.org/xptr*

16. EL-Sayed, M., Wang, L., Ding, L., & Rundensteiner, E. (2002). An Algebraic Approach for Incremental Maintenance of Materialized XQuery Views. *Proceedings of the 4th International Workshop on Web Information and Data Management.*

17. Schmidt, A., Manegold, S., & Kersten, M. (2003). Integrated Querying of XML Data in RDBMSs. *Proceedings of the 2003 ACM Symposium on Applied Computing.*

18. Bruno, E., Maitre, J., & Murisasco, E. (2003). Extending XQuery with Transformation Operators. *Proceedings of the 2003 ACM Symposium on Document Engineering.*

19. W3C XML Query (XQuery). Retrieved May 15, 2006, from *http://www.w3.org/XML/Query*

20. Document Object Model (DOM). Retrieved May 15, 2006, from *http://www.w3.org/DOM*