

## INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

**The quality of this reproduction is dependent upon the quality of the copy submitted.** Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning  
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA  
800-521-0600

UMI<sup>®</sup>



Data Extraction and Integration of Semistructured Documents

BY

Yip Chung

DISSERTATION

Submitted in partial satisfaction of the requirement for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

in the


OFFICE OF GRADUATE STUDIES


of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:







Committee in Charge

2001

UMI Number: 3018986

UMI<sup>®</sup>

---

UMI Microform 3018986

Copyright 2001 by Bell & Howell Information and Learning Company.

All rights reserved. This microform edition is protected against  
unauthorized copying under Title 17, United States Code.

---

Bell & Howell Information and Learning Company  
300 North Zeeb Road  
P.O. Box 1346  
Ann Arbor, MI 48106-1346

*To my family.*

## Acknowledgements

I am deeply indebted to my advisors Michael Gertz and Karl Levitt. They have been extremely supportive during my whole course of graduate study and gave me a lot of freedom to pursue research areas that interest me. Oceans of thanks to Michael who taught me how to do research, gave valuable feedback to my research, was always there when I needed help, and took the time to read the manuscript of this dissertation. Karl has been most generous in supporting me throughout the graduate program, even during times of financial difficulties. Without their unfailing support, this dissertation would not have been possible.

I am extremely grateful to Neel Sundaresan for giving me the opportunity to pursue this research project during my summer internship at the IBM Almaden Research Center. This work benefited from the discussions with Neel, Magnus Stensmo, Wayne Niblack, Jason Zien, Howard Ho, Murali Mani and K. Moidin Mohiuddin at IBM. I owe special thanks to Norm Pass who provides an excellent research environment to turn innovative ideas into inventions. I would also like to thank Ron Olsson through whom I joined IBM. The internship not only started this dissertation, but also gave me a valuable chance of working with and learning from talented and brilliant world-class researchers.

It is my luck to have Mary Fernandez as my mentor who advised me on various issues in graduate school, and never failed to be a responsive, friendly, excellent mentor. I

also received much advice and support from my previous mentor Lisa Purvis. It is a great pleasure to know successful women engineers like them who set role models for me.

Special thanks go to Matt Bishop, Richard Walters, Jim Diederich for being on my qualifying examination committee. I enjoyed the company of the Seclab folks, in particular Steven Cheung, Christopher Wee, Steven Templeton, Ricardo Anguiano, Nick Puketza, Dean Sniegowski, Nick Joshi, Nicole Carlson, Raymond Yip, David Klotz, Peter Mell and Scott Miller. Steven C. and Raymond gave me much insider's view of graduate studies at the University of California at Davis. Chris and Steven T. gave me valuable advice when I first started out in graduate school. It is also my pleasure to know members of the Database and Information System Lab.

Lastly, my family has my deepest gratitude. Mom and Dad have never failed to unconditionally support me on all the decisions I made in my life. Without their encouragement and their emotional and financial support, I would never have the chance to pursue graduate studies abroad, which not only benefits me in academic but also in personal development. Despite our occasional arguments, Wan is always there to offer a helping hand and has been most caring in all these years.

# Abstract

Before data from the Web can be shared in a meaningful and effective way, we have to deal with large amounts of legacy HTML documents. Information in the documents is buried in the text because HTML is for visual rendering purpose only, not for describing the semantics of data. State-of-the-art information retrieval techniques rely on keyword-based search engines. They do not support structured queries on documents. A user may prefer an integrated view that abstracts the heterogeneity among the documents and that facilitates visual browsing and data management. Existing approaches do not support an automated integration of highly heterogeneous HTML documents.

This dissertation aims to address these issues to make information buried in the HTML documents more accessible to users and applications. Building a practical system to transform the whole Web into a structured collection of documents is very difficult. Thus, we focus our attention on topic specific HTML documents - documents pertaining to a specific topic and created by different authors from different web sites.

We present Quixote, an automated document integration and schema discovery tool that transforms heterogeneous topic specific HTML documents into homogeneous XML documents conforming to a global schema. Quixote consists of three components:



(1) Document Converter: It extracts information from HTML documents and encodes such information in XML repository. It automatically extracts the information by rules that are insensitive to changes of the data formats and are applicable to diverse sources of data. It does not assume that topic specific documents follow a known format. It only assumes the records within a document follow some regular format.

(2) Schema Miner: We propose a new type of approximate schema called *majority schema* that describes only prevalent structures in a collection of XML documents. The Schema Miner infers a majority schema from the documents, which is small in size and can abstract the heterogeneity among the documents.

(3) Document Transformer: It automatically integrates XML documents based on a majority schema discovered. It adapts techniques from schema integration approaches on relational data to XML data. It addresses the unique challenge of preserving semantics of the documents in the integration process since a majority schema does not cover all structures in the documents.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	2
1.2	Overview . . . . .	7
1.2.1	Goals . . . . .	8
1.2.2	Architecture . . . . .	12
1.2.3	Environment . . . . .	16
1.3	Related Work . . . . .	18
1.3.1	Information Extraction . . . . .	18
1.3.2	Schema Discovery . . . . .	19
1.3.3	Schema Integration . . . . .	20

## CONTENTS

1.4	Contributions of this Dissertation . . . . .	22
1.5	Dissertation Organization . . . . .	24
1.6	Notations . . . . .	25
<b>2</b>	<b>Background</b>	<b>26</b>
2.1	XML Data Model . . . . .	27
2.1.1	Background . . . . .	27
2.1.2	Specialization . . . . .	32
2.1.3	Functions on XML Documents . . . . .	33
2.2	Document Type Definition (DTD) . . . . .	35
2.2.1	Background . . . . .	35
2.2.2	Formal Model for DTD . . . . .	36
2.2.3	Validity . . . . .	38
2.3	Tree Schema . . . . .	39
2.3.1	Description . . . . .	40

## *CONTENTS*

2.3.2	Formalism . . . . .	42
2.3.3	Relationship of Tree Schema with DTD . . . . .	46
2.3.4	Conformance . . . . .	48
<b>3</b>	<b>Converting HTML Documents to XML Form</b>	<b>50</b>
3.1	Document Information Content . . . . .	51
3.2	Approach . . . . .	55
3.2.1	Tokenization . . . . .	57
3.2.2	Concept Identification . . . . .	59
3.2.3	Grouping Rule . . . . .	68
3.2.4	Consolidating Rules . . . . .	73
3.3	Evaluation . . . . .	82
3.3.1	Computational Complexity . . . . .	82
3.3.2	Empirical Study . . . . .	87
3.3.3	Accuracy . . . . .	90

## CONTENTS

3.4	Related Work . . . . .	91
3.4.1	Classification . . . . .	91
3.4.2	Comparison . . . . .	94
<b>4</b>	<b>Schema Mining</b>	<b>97</b>
4.1	Problem Formulation . . . . .	98
4.1.1	Majority Schema . . . . .	98
4.1.2	Problem Definition . . . . .	100
4.2	Approach . . . . .	104
4.2.1	Computing Frequent Label Paths . . . . .	104
4.2.2	Using Domain Knowledge . . . . .	107
4.2.3	Unification . . . . .	110
4.2.4	Enriching DTD Content Model . . . . .	122
4.3	Evaluation . . . . .	126
4.3.1	Time Complexity . . . . .	126

## CONTENTS

4.3.2	Quality of Majority Schema . . . . .	128
4.4	Related Work . . . . .	135
4.4.1	Classification . . . . .	135
4.4.2	Comparison . . . . .	137
<b>5</b>	<b>Document Transformation</b>	<b>142</b>
5.1	Problem Formulation . . . . .	143
5.1.1	Semantic Preservation . . . . .	143
5.1.2	Problem Definition . . . . .	146
5.2	Approach . . . . .	149
5.2.1	Derivation of Semantic Mapping . . . . .	149
5.2.2	Transformation . . . . .	169
5.3	Evaluation . . . . .	177
5.3.1	Correctness . . . . .	177
5.3.2	Computational Complexity . . . . .	181

## *CONTENTS*

5.4	Related Work . . . . .	188
5.4.1	Existing Approaches . . . . .	188
5.4.2	Comparison . . . . .	191
<b>6</b>	<b>Conclusion</b>	<b>194</b>
6.1	Contributions . . . . .	195
6.2	Applications . . . . .	197
6.2.1	Structured Queries . . . . .	197
6.2.2	Summarization . . . . .	198
6.2.3	Data Presentation . . . . .	198
6.2.4	Indexing . . . . .	200
6.2.5	Storage . . . . .	201
6.3	Future Work . . . . .	201
	<b>Symbols</b>	<b>225</b>

# Chapter 1

## Introduction

In Section 1.1, we first motivate the problem we aim to address in this dissertation - integrating a heterogeneous collection of topic specific HTML documents into an XML repository conforming to a global schema. In Section 1.2, we give an overview of Quixote [CGS01], an automated document integration and schema mining tool we propose to solve this problem. We discuss related work in Section 1.3. In Section 1.4, we highlight our major contributions. Finally, the organization of the dissertation and notations are described in Section 1.5 and Section 1.6.



## 1.1 Motivation

The vision of the Semantic Web is that it will be the medium for the exchange, sharing, and retrieval of information in a meaningful and effective way [BLFD99, ABS99]. An important data exchange standard to realize this vision is the eXtensible Markup Language XML [W3C98b]. It provides means to enrich semistructured documents by more structural and semantic information. A number of formalisms have been proposed to describe structures of XML documents ([LC00, W3C00d, W3C99a, W3C00f, W3C00g, BMR99, FSSW]), and several sophisticated approaches to processing XML data have been developed, in particular XML query languages and query processing techniques, e.g., [BC00, W3C00e, CRF00, DFF<sup>+</sup>98, AQM<sup>+</sup>97, JR98, W3C00b, BMR99, FSSW].

**Problem dimension:** However, before the vision of the Semantic Web can be realized, we have to deal with large volumes of legacy data marked up in HTML. Transforming the whole Web into a collection of structured XML documents is very difficult. Thus, we focus our attention on large collections of topic specific HTML documents which are documents pertaining to a specific topic, authored by different people from diverse data sources. Examples include course offerings and seminar schedules from universities, descriptions of movies and movie show times, company financial information, flight itinerary from individual airlines, or résumés from personal homepages or Web sites like `monster.com`.

**Limitations of existing techniques:** Figure 1.1 shows some example résumés HTML documents. Let us take a look at the actual HTML source codes of these résumés documents (Figure 1.2). Existing technology provides limited support in locating information from these documents which is buried in the HTML text:

1. State-of-the-art information retrieval techniques rely on keyword-based search engines which do not support structured queries on these documents. For example, one may ask "show me all résumés of people with Java programming skills". Keyword-based search engines may return résumés of people from Java, Indonesia. The issue here is that the semantics of the information content of HTML documents is unknown to the applications since HTML is a markup language for visual rendering purpose only, not for describing information.
2. Heterogeneity among query results may not be desirable from a user's point of view. For example, although there may be more than one commonly accepted résumé document style, an employer may prefer a uniform integrated view on the documents that can abstract out their commonality. Existing schema mining approaches do not address this problem well. They either infer exact schemas which are too large in size, or approximate schemas which may be too general.
3. From the point of view of an administrator, it is desirable to have an integrated repository of the heterogeneously structured résumés in order to facilitate data management. Such an integrated repository would also facilitate querying and

displaying information about candidates. Existing technology does not support an automated integration of heterogeneous topic specific documents.

<b>Andrew Young</b>	<b>Srinivasan Chakravarthi</b>
<b>Objective</b>	201, Pearl Street #3, Cambridge, MA 02139 USA. Ph: 617 353 5885 (w) Ph : 617 864 9611 (h) Email: <a href="mailto:srini@bu.edu">srini@bu.edu</a>
To obtain my Ph.D. teach undergraduate introductory astronomy and research extragalactic objects.	<b>Education</b>
<b>Education</b>	<ul style="list-style-type: none"> <li>• <i>Aug '95 - Present</i> Ph.D student, Boston University, Manufacturing Engineering Department Current GPA - 3.9 Research Adviser : Prof. Scott Dunham</li> <li>• <i>Aug '91 - May '95</i> Bachelor of Technology in Metallurgical Engineering, from Institute Of Technology, Banaras Hindu University, India</li> </ul>
<p>September 1997 to Present     <u>The University of Minnesota, Twin Cities Campus</u> - Minneapolis, MN 2nd Year Ph.D. student in Astrophysics</p> <p>September 1992 to May 1996     <u>Boston University, College of Arts and Sciences</u> - Boston, MA B.A. in Astronomy and Physics Minor in Archaeological Studies and Mathematics Distinction in Astronomy Cum Laude Honors</p>	<b>Work Experience</b>
<b>Computer Experience</b>	TCAD Engineer <i>May '97 - Aug '97</i> Intel Corporation Hillsboro OR.

Figure 1.1: Resume HTML documents

```

<html>
<head>
<title>Andrew Young's Curriculum Vitae</title>
<body bgcolor="#b117">
</head>
<body>
<center>
<h1><a href="mailto:eyoung@uss-enterprise.bu.edu">And:
</center>
<br>
</center>
<hr noshade>
<h2>Objective</h2>
<TABLE CELLSPACING=5>
<TR>
<TD VALIGN=TOP>
To obtain my Ph.D. teach undergraduate introductory as:
</TD> </TR> </TABLE>
<hr>
<h2>Education</h2>
<TABLE CELLSPACING=5>
<TR>
<TD VALIGN=TOP>
September 1997 to Present
</TD>
<TD VALIGN=TOP>
</td>
<TD VALIGN=TOP>
<a href="http://www.umn.edu/tc">The University of Minn
2nd Year Ph.D. student in Astrophysics<br>
</TD> </TR>
<TR>
<TD VALIGN=TOP>
September 1992 to May 1996
</TD>
<TD VALIGN=TOP>
</td>
<TD VALIGN=TOP>
</td>
</TR>
</TABLE>
</body>
</html>

```

```

<html>
<head>
<title> Resume Of Scini Chakravarthi </title>
<META
name="description" content="Resume Of Scini Chakravar
Student at Boston University working on Modeling Of T
Diffusion.">
</head>
<BODY
BGCOLOR="#ffffff"
TEXT="#100000"
VLINK="#100000">
<body>
<h1><center><strong> Srinivasan Chakravarthi </center>
<p>
<center> 201, Pearl Street #3.
<dt> Cambridge, MA 02139 USA.
<dt>Ph: 617 353 5885 (w)
<dt> Ph : 617 864 9611 (h)
<dt> Email: <a href="mailto:scini@bu.edu"> scini@bu.
<p>
<hr>
<h3> Education </h3>
<ul>
<li><i> Aug '95 - Present </i>
<dt>Ph.D student, Boston University, Manufacturing En
<dt>Current GPA - 3.9
<dt>Research Adviser : Prof. Scott Dunham
<p>
<li><i> Aug '91 - May '95 </i>
<dt>Bachelor of Technology in Metallurgical Engineeri
Institute Of Technology, Banaras Hindu University. In
</ul>
</body>

```

Figure 1.2: Source HTML code for resume documents

To recap, what we would like to have is an integrated repository of XML documents in which semantic information is encoded conforming to a global schema as shown below:

```

<RESUME>
  <OBJECTIVE val="Andrew Young # Objective # To obtain my Ph.D.
    teach undergraduate introductory astronomy and
    research extragalactic objects."/>
  <EDUCATION val="Education">
    <ORGANIZATION val="The University of Minnesota #
      Twin Cities Campus - Minneapolis MN">
      <DATE val="September 1997 to Present">
        <DEGREE val="2nd Year Ph.D. student in Astrophysics"/>
      </DATE>
    </ORGANIZATION>

```

```

...
</EDUCATION>
<EXPERIENCE val="Research Experience"> ... </EXPERIENCE>
<EXPERIENCE val="Work Experience"> ... </EXPERIENCE>
<EXPERIENCE val="Laboratory Experience"> ... </EXPERIENCE>
<SKILLS val="Computer Experience"> ... </SKILLS>
<AWARDS val="Awards"> ... </AWARDS>
<ACTIVITIES val="Memberships"> ... </ACTIVITIES>
<ACHIEVEMENTS val="Publications/Presentations">...</ACHIEVEMENTS>
</RESUME>

```

```

<RESUME>
  <CONTACT val="Srinivasan Chakravarthi # 201, Pearl Street #3,
    Cambridge, MA 02139 USA. Ph:617 353 5885(w)
    Ph:617 864 9611(h) Email:srini@bu.edu"/>
  <EDUCATION val="Education">
    <ORGANIZATION val="Boston University #
      Manufacturing Engineering Department">
      <DATE val="Aug '95-Present">
        <DEGREE val="Ph.D Student #
          Research Adviser: Prof. Scott Dunham"/>
        <GPA val="Current GPA - 3.9"/>
      </DATE>
    </ORGANIZATION>
    ...
  </EDUCATION>
  <EXPERIENCE val="Work Experience"> ... </EXPERIENCE>
  <SKILLS val="Software Skills"> ... </SKILLS>
  <FIELDS val="Relevant Courses"> ... </FIELDS>
</RESUME>

```

The information is now more accessible to users and other applications. The following are some possible applications:

- Users can issue structured queries against the integrated collection using XML query languages.
- It gives users unfamiliar with the documents a bird's eye view on the documents, much like a table of content that leaves out the details.
- A user may prefer to have all documents represented in XML in the most typical way so that a master format stylesheet can be applied to all of them for presentation purposes.
- The global schema of the documents can be used to build index structures in order to facilitate efficient query processing on these documents.
- The global schema can be used to optimize the storage of the XML documents. XML structures can be stored together according to their proximity in the schema to facilitate retrieval of information.

## 1.2 Overview

This dissertation aims to address the issues discussed in Section 1.1 so that information from topic specific HTML documents is readily available to users and other applications. We propose Quixote, an automated document integration and schema

mining tool that transforms heterogeneous topic specific HTML documents into homogeneous XML documents conforming to a global schema.

Section 1.2.1 states the goals of Quixote. The components of Quixote and the overall system are described in Section 1.2.2. In Section 1.2.3, we describe the environment in which Quixote operates and state the assumptions we make on topic specific documents.

### 1.2.1 Goals

Quixote facilitates the information filtering process by achieving the following goals:

- *Goal 1:* It extracts information from HTML documents and encodes such information in XML documents. Information is encoded in XML documents in two ways: (1) in XML tag names describing the information objects, and (2) the tree structure of XML documents describing the logical information organization and relationships of the documents.
- *Goal 2:* It infers a global schema, called *majority schema*, from the documents which abstracts the commonality among the documents by describing their prevalent structures only. Such kind of schema results in a concise description of the documents at a higher level of abstraction.

- *Goal 3*: It integrates the documents based on the majority schema discovered. Integrating these documents provides users a uniform view on the documents and offers diverse opportunities for query optimization and for building index structures.

The following illustrates these goals. Suppose we have two topic specific HTML documents about résumés:

Resume HTML Document 1

```
<html>
  <title> Resume of John Dole </title>
  <body>
    <h1> Email:jdole@ucdavis.edu </h1>
    <h1> Educational Background </h1>
    <table>
      <tr><td> 1999 </td>
        <td> M.Sci.(Comp.Sci.) </td>
        <td> UC Davis </td>
        <td> GPA=4.0/4.0 </td>
      </tr>
      <tr><td> 1997 </td>
        <td> B.Sci.(Comp.Sci.) </td>
        <td> UC Davis </td>
        <td> GPA=3.7/4.0 </td>
      </tr>
    </table>
  </body>
</html>
```



## Resume HTML Document 2

```
<html>
  <title> Resume of Mary Smith </title>
  <body>
    <h1> Academic Background </h1>
    Ph.D., Stanford University, 1998
  </body>
</html>
```

Goal 1 aims to extract the information content of these two HTML documents and to encode it in the following XML documents:

## XML Document 1

```
<RESUME val="Resume of John Dole">
  <CONTACT-INFO val="Email:jdole@ucdavis.edu">
  <EDUCATION val="Educational Background">
    <DATE val="1999">
      <DEGREE val="M.Sci.(Comp.Sci.)"/>
      <ORGANIZATION val="UC Davis"/>
      <GPA val="4.0/4.0"/>
    </DATE>
    <DATE val="1997">
      <DEGREE val="B.Sci.(Comp.Sci.)"/>
      <ORGANIZATION val="UC Davis"/>
      <GPA val="3.7/4.0"/>
    </DATE>
  </EDUCATION>
</RESUME>
```

## XML Document 2

```
<RESUME val="Resume of Mary Smith">
  <EDUCATION val="Academic Background">
    <DEGREE val="Ph.D.">
      <ORGANIZATION val="Stanford University"/>
  </EDUCATION>
</RESUME>
```

```

        <DATE val="1998"/>
        <THESIS val="Semistructured Data"/>
    </DEGREE>
</EDUCATION>
</RESUME>

```

Suppose we have a collection of XML documents similar to the above XML documents. Goal 2 aims to derive a majority schema from these topic specific XML documents:

Majority Schema

```

<RESUME>
  <EDUCATION>
    <DATE>
      <DEGREE>
        <ORGANIZATION>
          <GPA>
        </DATE>
      </EDUCATION>
    </RESUME>

```

Goal 3 aims to integrate these XML documents based on this majority schema. The integrated XML documents are given below. Information specific to an XML document but not common among other documents is marked up in the attribute `val`, e.g., (`CONTACT-INFO:jdole@ucdavis.edu`), (`THESIS:Semistructured Data`).

Integrated XML Document 1 Based on XML Document 1

```

<RESUME val="Resume of John Dole (CONTACT-INFO:jdole@ucdavis.edu)">
  <EDUCATION val="Educational Background">
    <DATE val="1999">
      <DEGREE val="M.Sci.(Comp.Sci.)"/>
    </DATE>
  </EDUCATION>
</RESUME>

```

```
        <ORGANIZATION val="UC Davis"/>
        <GPA val="4.0/4.0"/>
    </DATE>
    <DATE val="1997">
        <DEGREE val="B.Sci.(Comp.Sci.)"/>
        <ORGANIZATION val="UC Davis"/>
        <GPA val="3.7/4.0"/>
    </DATE>
</EDUCATION>
</RESUME>
```

Integrated XML Document 2

```
<RESUME val="Resume of Mary Smith">
    <EDUCATION val="Academic Background">
        <DATE val="1998"/>
        <DEGREE val="Ph.D. (THESIS:Semistructured Data)">
        <ORGANIZATION val="Stanford University"/>
    </DATE>
    </EDUCATION>
</RESUME>
```

## 1.2.2 Architecture

These goals are realized by the three components of Quixote, namely (1) the Document Converter, (2) the Schema Miner, and (3) the Document Transformer. The architecture of Quixote is illustrated in Figure 1.3. The user initiating the integration process gathers a collection of topic specific HTML documents, typically by deploying a web crawler. A topic consists of concepts describing the information pertaining to the topic. The user specifies concept names of the topic and information on how to associate HTML text with the concept names. The Document Converter extracts

information from HTML documents and encodes it in XML documents in topic specific XML tags using the information on concept names and domain independent heuristics. The Schema Miner then infers a majority schema from these XML documents which describes prevalent structures among them. The majority schema can be represented as a DTD. Finally, the Document Transformer transforms the XML documents to conform to the majority schema with semantics preserved in the transformation process. Semantics are preserved in the sense that the textual content of nodes are stored in some related nodes in the transformed document. The HTML document collection is thus converted to a homogeneous repository of XML documents with a global DTD describing their structures. This repository can be used by other applications for indexing, formatting, storage or querying.

The processes of Quixote are illustrated in Figure 1.4. Topic specific HTML documents are first gathered from the Web. The Document Converter converts these documents into XML documents in which XML tags are chosen from concept names about the topic. The user gives information on how to identify XML tags by keywords or by examples. The Document Converter restructures XML documents so that their tree structures represent their logical layout by using domain-independent restructuring rules and concept constraints. The Schema Miner then abstracts out commonalities among the XML documents by discovering a majority schema and its associated DTD which describe prevalent structures among the XML documents. Finally, the Document Transformer restructures the XML documents to conform to the majority schema.

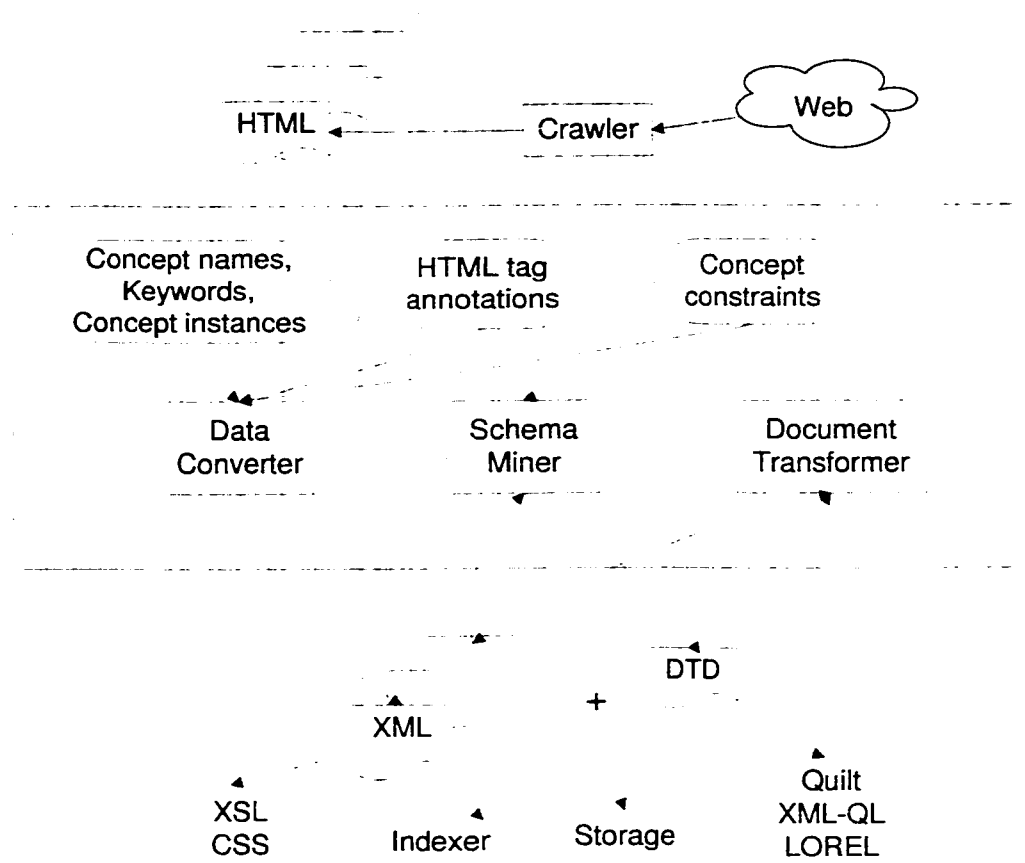


Figure 1.3: Architecture of Quixote

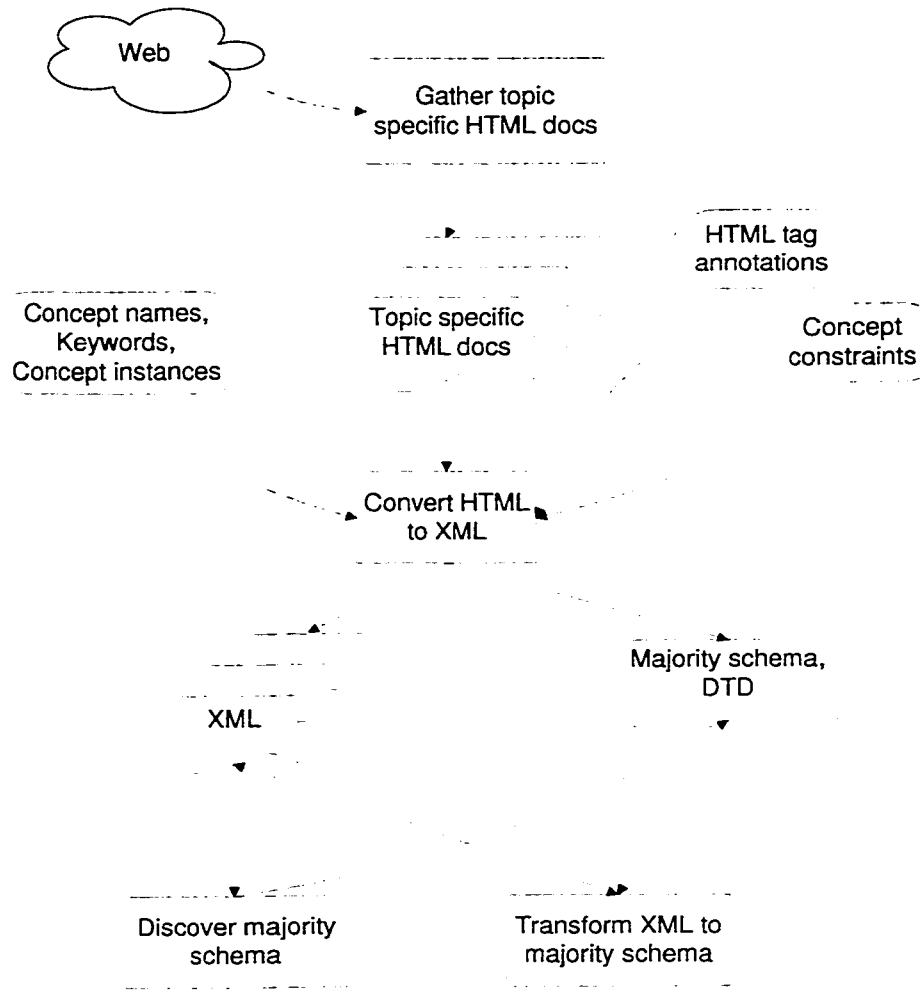


Figure 1.4: Dataflow

### 1.2.3 Environment

We make several general assumptions on the properties of topic specific documents:

1. *Information Content Property*: The information content of a document consists of information objects described in HTML text. A concept name can be associated with an information object describing its information content. The textual content of an information object in an XML document is its associated HTML text. An information object is also called a *concept node* in an XML document. For example, `<h1> B.Sci.(Comp.Sci.) </h1>` in an HTML document corresponds to an information object DEGREE in an XML document. The textual content of DEGREE is `B.Sci.(Comp.Sci.)`.
2. *Concept Hierarchy Property*: Information objects are organized at different levels of abstractions in a tree-like structure. Higher level information objects are described in terms of lower level ones. For example, in a résumé, an information object on a person's educational background is a high level information object. It can be refined further by lower level information objects about the degrees, dates of receiving the degrees and institutions granting the degrees.
3. *Homonym Property*: The context of an information object is described by three components: (1) its associated concept name, (2) its textual content which is its associated HTML text, and (3) its relationship to the high level information object it details information about, and its lower level information objects that

refine it. For example, a date information object under a person's educational background is different from a date information object under a person's working experience. The former is a chronological information item about a person's education which may be refined by information on degrees, GPA and institutions granting the degrees. The latter is a chronological information item about a person's working experience which may be refined by information on job titles, job descriptions and organizations. In other words, there are homonyms among the concept names. Homonyms are information objects of the same name but with different contexts.

4. *Synonym Property*: Since there is a single user initiating the integration process, we assume the concept names about the topic specified by the user do not contain synonyms. Synonyms are information objects of different names but with the same context. For example, phone and ph both refer to the same information object.
5. *Tokenizable Property*: Separate information objects can be recognized from the documents. Identification of information objects can be based on punctuation delimiters, newlines characters, spacing, or more sophisticated language parsing techniques. For example, B.Sci.(Comp.Sci.), University of California at Davis, 1996 contain three information objects separated by the punctuation ",", namely B.Sci.(Comp.Sci.), University of California at Davis and 1996.



6. *Regular Intradocument Format Property*: Although the formats of different documents can be different, the records *within* a document is highly regular - high level information objects are described by groups of lower level information objects in the same way. Take for an example, a person may organize information objects about her educational background chronologically, or according to the degrees, but not a mix of both ways within a document.

## 1.3 Related Work

In this section, we briefly describe existing approaches related to the three problems addressed in this dissertation, namely information extraction, schema mining and schema integration.

### 1.3.1 Information Extraction

Several approaches have been developed to close the gap in translating HTML documents into XML data using *wrappers* . Wrappers extract records from an information source which is typically unstructured.

First generation wrappers - MedMaker ([PGMU96]), [HGMC<sup>+</sup>97], Editor ([AM97]), W4F([SA99]) and YAT ([CDSS98]) - are manual wrappers that require users to specify how to extract data from HTML documents. They provide a language to facilitate

the specification process. The problems of manual wrappers are: (1) They assume the data are in a known format, which is inapplicable for data gathered over several data sources. (2) The format of the data from a given source may change over time. Every change of format would require a new handcrafted wrapper.

Second generation wrappers are automatic wrappers that can learn the extraction rules ([Ade98, ECJ+98, AK97, Kus00, DYKR00]). However, they all assume that the data sources are relatively homogeneous - all the records in an HTML document or all HTML documents follow a particular format. Some of these wrappers ([Ade98, ECJ+98]) require the user to specify the structures for the documents. While these are reasonable assumptions for data from a single data source, they are unlikely to hold for documents gathered from diverse data sources.

### 1.3.2 Schema Discovery

In order to process XML documents converted from HTML documents effectively, it is essential to know the schema underlying the documents. Such a schema may not be available or known to the user.

Several approaches to the mining of schemas have been proposed. [BDFS97, NUWC97, WYW00, NAM98, NAM97, WL99, WL98] take a data-centric view on a collection of HTML documents based on Object Exchange Model (OEM) [NUWC97]. Entities of the documents are objects and documents are linked by hyperlinks. The link-

age structure is described by labels on edges. A collection of documents is viewed as a web of interconnected objects modeled as edge-labeled graphs. The schemas discovered by these approaches describe possible incoming and outgoing labels from objects. [GGR<sup>+</sup>00, PV00] take a document view on a collection of XML documents using a data model similar to [W3C99a] which models documents as node-labeled trees. Relationships between objects are described by sibling, ancestor and descendant relationships. These approaches use DTDs to describe the content model of the objects.

Existing approaches infer two types of schemas. [NUWC97, GGR<sup>+</sup>00, PV00] discover *exact* schemas that describe all structures found in the documents and those structures only. They are inappropriate for heterogeneous documents because exact schemas are large in size. To alleviate this problem, [WYW00, NAM98, WL98, WL99, GGR<sup>+</sup>00] infer *approximate schema* that describe the structures in a more general way. In doing so, they may cover irrelevant structures not found in the documents. In a collection of heterogeneous documents, this can lead to an approximate schema that is too general to convey useful information.

### 1.3.3 Schema Integration

The major tasks in solving the schema integration problem are [AES98]: (1) translation of local schemas into a common data model, (2) identification of related objects in the local schemas and classification of their relationships, (3) deriving a global

schema from local schemas, and (4) integrating the local databases into a global database based on the global schema.

[LNEM89, NEML86, SPD92, HR90, Ber91, KDN90, GMP<sup>+</sup>92] perform schema integration at the conceptual level. They identify conflicts among the schemas and propose methods to resolve these conflicts. [DeM89, PRSL93, Pu91, CS91, RR95, PRSL93] integrate at the data level. They identify related objects and conflicts by looking at the values of the objects.

Schema integration approaches can be classified based on the underlying data models. Schema integration approaches developed for relational data ([LNEM89, EMN84, NEML86, SP94, DeM89, CS91, PRSL93]) are unsuitable for XML data because the data model of XML is distinctly different from the entity-relationship or relational model. Object-oriented approaches ([TS93, GMP<sup>+</sup>92, Ber91, KDN90]) integrate class hierarchies based on their types, inheritance, methods and dynamic behavior. The context of information content in XML documents involves the neighboring nodes in document trees which shares some similarities with the context of classes in object-oriented data. The insights offered by this work can be used for future research in object-oriented data. Recent approaches on integrating XML data address the schema integration problem in a different setting. Specifically, [MZ98] concerns the integration of data with the same schema in different data models whereas our goal is to integrate data with different schemas but in the same data model. [Mur97] provides a mechanism to allow a user to specify how to integrate the documents whereas we aim to derive how the integration can be achieved.

A major issue with many of these approaches is that they depend heavily on user intervention. This may be feasible if the objects of integration include several local schemas. For topic specific documents created by different authors, each document can be thought of as a local schema. Heavy user intervention typically is infeasible in such scenarios.

## 1.4 Contributions of this Dissertation

In view of the limitations of existing approaches, we propose a coherent and integrated approach that combines the steps (1) of converting HTML documents into XML documents, (2) of discovering a global schema underlying these documents, and (3) of integrating these documents to conform to a global schema. This dissertation makes several contributions in processing and integrating legacy HTML documents as a first step to realize the vision of the Semantic Web:

1. With respect to extracting information from HTML documents, unlike existing manual wrappers, the document conversion process can automatically extract data by employing rules that are insensitive to changes in the format of the data and are applicable to diverse sources of data. The intuition is that the visual hints and inherent tree structure of HTML documents give us strong clues on the layout of their information content. The rules employed consider the format clues of HTML markup tags, the tree structure of documents and the names of

XML tags. The format clues and tree structures are independent of the topic domain.

The Document Converter does not assume that the documents follow a known format. It only assumes that the records within a document follow some regular pattern. For example, a person either organizes her educational background information items chronologically or by the organization granting the degree, but not both. It provides a simple, extensible mechanisms to allow users to specify partial apriori knowledge on the structures of the information content of the topic. However, unlike other approaches (e.g., [Ade98, ECSL98]), this knowledge is not mandatory.

2. With respect to discovering a schema for XML documents, we propose a new type of approximate schema called *majority schema* that describes only prevalent structures found in the documents. The intuition is that there is usually a typical way of describing the information content of topic specific documents. Although documents may be marked up in slightly different ways, they typically follow some commonly accepted format. A majority schema is applicable to heterogeneous documents because it is small in size and can abstract the heterogeneity among the documents. It can be used as a basis to integrate documents into a consistent and concise repository, which greatly facilitates information retrieval and data management.

3. With respect to integrating XML documents based on their global schema, we present an approach that automates the integration process by making use of domain knowledge on topic specific documents. Existing schema integration approaches developed for relational and object-oriented data are not directly applicable to XML data. The Document Transformer adapts integration techniques of these approaches to XML data. It also addresses the unique challenge of preserving semantics of the documents in the integration process since a majority schema does not cover all structures found in the documents. The semantics considered are the textual content of information objects and their tree structures.

## 1.5 Dissertation Organization

The rest of the dissertation is organized as follows. In Chapter 2, we give the background knowledge and define important concepts for subsequent chapters. In Chapters 3, 4, 5, we present a detailed description and analysis of the approaches of the main components of Quixote, namely the Document Converter, the Schema Miner and the Document Transformer. Finally, in Chapter 6, we summarize our contributions, highlight some applications of Quixote and discuss possible extensions.

Throughout this dissertation, we use *résumé* documents as an example of topic specific documents to illustrate the idea of our approach. However, the presented approach is applicable to other topic domains satisfying the general properties stated in Section 1.2.3.

## 1.6 Notations

Throughout this dissertation, we adopt the following notations:

The domain of an object  $obj$  is denoted by  $\mathbf{obj}$ . The domain of integers, natural numbers, real numbers, boolean, strings and characters are  $\mathbf{Z}$ ,  $\mathbf{Z}^+$ ,  $\mathbf{R}$ , **boolean**,  $\mathbf{S}$ ,  $\Sigma$  respectively. A string is a concatenation of characters. Groups of characters separated by the space character  $\sqcup$  in a string are words in the string.

We denote infinity by  $\infty$ , empty by  $\epsilon$ , null (undefined) by  $\perp$ , and the empty set by  $\emptyset$ .

The operator  $\circ$  concatenates two strings. If  $p$  is a substring of the string  $q$ , it is denoted by  $p \in q$ .

Let  $f$  be a polymorphic function. We use  $Dom_1$  or  $Dom_2$  to denote the domain of  $f$  and use  $Dom_3$  to denote the image of  $f$ . Its signature is written as  $(Dom_1 \cup Dom_2) \rightarrow Dom_3$ .

Let  $X$  be a set. We use  $X^*$ ,  $Set(X)$ ,  $PowerSet(X)$  to denote a sequence, a set and a power set built over elements from  $X$  respectively. A sequence may also be denoted by  $\langle x_1, \dots, x_n \rangle$ .  $P \oplus Q$  appends/adds the element/sequence/set  $Q$  to the sequence/set  $P$ . The size (or length) of the sequence/set  $P$  is denoted by  $|P|$ . If the sequence/set  $P$  is a subsequence/subset of another sequence/set  $Q$ , it is denoted by  $P \in Q$ .



## Chapter 2

# Background

In this chapter, we introduce the formalism used throughout the dissertation which is a general formalism for XML documents. In Section 2.1, we describe the data model for and operations on XML documents. In Section 2.2, we give the formalism of Document Type Definition (DTD) and define the validity of XML documents with respect to a DTD. In Section 2.3, we propose and define the notion of tree schema. We discuss its relationship with DTDs, and define the notion of conformance of an XML document with respect to a tree schema.

## 2.1 XML Data Model

In Section 2.1.1, we first give the formalism of the data model for XML documents. Functions on XML documents are described in Section 2.1.3.

### 2.1.1 Background

An XML document is made up of elements. One is the document element (root). It is the document type of the XML document and serves as the entry point to the document. An element may be associated with a set of attributes. A non-empty element node consists of a sequence of child nodes or it can be a text element which is simply a string (PCDATA).

Unlike relational data, there is no standard data model for XML data yet. There are two popular data models, proposed in the Object Exchange Model OEM ([AQM<sup>+</sup>97]) and Infoset ([W3C99a]), among others ([BMR99, FSSW, BDFS97, BDHS96]). OEM takes a data-centric view on a collection of documents in which elements are vertices and their relationship is described by labels on edges in an edge-labeled graph. Documents are connected by hyperlinks (references to another XML document) and they are viewed as a web of interconnected objects. Infoset takes a document-centric view which organizes elements of an XML document into a node-labeled tree. The document structure is reflected in the structure of the tree, in which one can describe the siblings, ancestors and descendants of a node. We favor Infoset because we are inter-

ested in document structures. We use a data model similar to [FS00] as a formalism for the Infoset data model.

In our data model, an XML document is modeled as a node-labeled tree. Elements are vertices with element names being their labels. Attributes of elements are considered as properties of elements. Each element has a unique identifier. Without loss of generality, we can assume the identifier is stored in a special attribute *ID*. Since we do not consider hyperlinks, referential keys (IDREFs) are ignored. The atomic values of all attributes are of type string.

**Definition 2.1 (XML Document).** *Given a set of element names (labels)  $E$ , a set of attribute names  $A$ , and a set of string values  $S$  which is the type of all atomic values. An XML document is an ordered tree with labeled nodes, denoted by  $(V, root, Label, Children, Attr, AttrVal)$  where*

- *$V$  is a set of vertices.*
- *$root$  is a distinguished vertex in  $V$ . It is also the document type of the XML document.*
- *$Label : V \rightarrow E$  is a partial function that maps a vertex to its label.*
- *$Children : V \rightarrow V^*$  is a total function that maps a vertex to its sequence of children in the document tree. If a vertex does not have any children, it returns  $\emptyset$ .*
- *$Attr : V \rightarrow Set(A)$  is a partial function that maps a vertex to its set of attributes.*

- *AttrVal* :  $\mathbf{V} \times \mathbf{A} \rightarrow \mathbf{S}$  is a partial function that gives the atomic (string) value of an attribute for a vertex. The value of  $v.a$  is denoted by  $Value(v.a)$ , or  $v.a$  if the context is clear. □

The size of an XML document  $\mathbf{X}$ , denoted as  $|\mathbf{X}|$ , is  $|\mathbf{X.V}|$ . The size of the collection of XML documents  $\mathcal{X} = \{\mathbf{X}_1 \dots \mathbf{X}_n\}$ , denoted by  $|\mathcal{X}|$ , is  $\sum_{\mathbf{X} \in \mathcal{X}} |\mathbf{X}|$ .

The value of a node is denoted by  $Value(v)$ . For an element node, it is its unique identifier encoded in its attribute  $ID$ . For a text node, it is the string itself.

An *XML document fragment* is a subtree in an XML document. The subtree rooted at the vertex  $v$  in  $\mathbf{X}$  is denoted as  $T(v)$ . It can be modeled in the same way as an XML document.

XML tag names are in UPPER CAPITAL, text in lower capital and attributes in *italics lower capital*.

We use the term "node" and "vertex" interchangeably. A node  $v$  of an XML document  $\mathbf{X}$  is denoted as  $v \in \mathbf{X}$ . Let  $\mathcal{X}$  be a collection of XML documents. If  $v$  is found in an XML document in this collection, we denote it as  $v \in \mathcal{X}$ . The domain of vertices is  $\mathbf{V}$ . The domain of XML documents is  $\mathbf{X}$ .

**Example 2.1: XML Document**

An example of XML document is given below. Its document type is `RESUME` which consists of elements `OBJECTIVE` and `EDUCATION`. Element `OBJECTIVE` is made up of a text node "To obtain a summer intern position". Element `EDUCATION` consists of two `DEGREE` elements. Both `DEGREE` elements have attributes `title` and `date`. The first `DEGREE` element is described by `ORGANIZATION` followed by `SPECIALIZATION`. The second `DEGREE` element consists of `ORGANIZATION`, followed by `GPA`.

```
<RESUME>
  <OBJECTIVE>
    To obtain a summer intern position.
  </OBJECTIVE>
  <EDUCATION>
    <DEGREE title="M.Sci.(Comp.Sci.)" date="1996">
      <ORGANIZATION> U.C. Davis </ORGANIZATION>
      <SPECIALIZATION> Data mining </SPECIALIZATION>
    </DEGREE>
    <DEGREE date="1994" title="B.Sci.(Comp.Sci.)">
      <ORGANIZATION> Stanford University </ORGANIZATION>
      <GPA> 3.9/4.0 </GPA>
    </DEGREE>
  </EDUCATION>
</RESUME>
```

The description for this document in the model is  $(v, RESUME, Label, Children, Attr, AttrVal)$  where

- $V = \{v_1, \dots, v_{14}\}$

- *Label* is the following function:

$v_1 \rightarrow \text{RESUME}$

$v_2 \rightarrow \text{OBJECTIVE}$

$v_3 \rightarrow \text{EDUCATION}$

$v_4 \rightarrow \text{DEGREE}$

$v_5 \rightarrow \text{ORGANIZATION}$

$v_6 \rightarrow \text{SPECIALIZATION}$

$v_7 \rightarrow \text{DEGREE}$

$v_8 \rightarrow \text{ORGANIZATION}$

$v_9 \rightarrow \text{GPA}$

- *Children* is the following function:

$v_1 \rightarrow \langle v_2, v_3 \rangle$

$v_3 \rightarrow \langle v_4, v_7 \rangle$

$v_4 \rightarrow \langle v_5, v_6 \rangle$

$v_7 \rightarrow \langle v_8, v_9 \rangle$

$v_2 \rightarrow \langle v_{10} \rangle$

$v_4 \rightarrow \langle v_{11} \rangle$

$v_5 \rightarrow \langle v_{12} \rangle$

$v_7 \rightarrow \langle v_{13} \rangle$

$v_8 \rightarrow \langle v_{14} \rangle$

$v \rightarrow \perp$ , otherwise

- *Attr* is the following function:

$$v \rightarrow \{title, date\}, v \in v_4, v_7$$

$$v \rightarrow \perp, \text{ otherwise}$$

- *AttrVal* is the following function:

$$v_4 \times title \rightarrow \text{"M.Sci.(Comp.Sci.)"}$$

$$v_4 \times date \rightarrow \text{"1996"}$$

$$v_7 \times title \rightarrow \text{"B.Sci.(Comp.Sci.)"}$$

$$v_7 \times date \rightarrow \text{"1994"}$$

### 2.1.2 Specialization

Based on the Information Content Property (Section 1.2.3), the children text nodes of an element node are its textual content. Since we are interested in the structures of element nodes in an XML document and not in text nodes, as a technical convenience, we assume the children text nodes of an element node in an XML document are consolidated into an attribute named *val*. Therefore, throughout the dissertation, we refer to XML documents that do not contain text nodes. All text nodes are encoded in the attribute *val* of their parent element nodes. The XML document given in Section 1 would then be represented as follows:

```
<RESUME>
  <OBJECTIVE val="To obtain a summer intern position"/>
  <EDUCATION>
    <DEGREE title="M.Sci.(Comp.Sci.)" date="1996">
```

```

    <ORGANIZATION val="U.C. Davis"/>
    <SPECIALIZATION val="Data mining"/>
  </DEGREE>
<DEGREE date="1994" title="B.Sci.(Comp.Sci.)">
  <ORGANIZATION val="Stanford University"/>
  <GPA val="3.9/4.0"/>
</DEGREE>
</EDUCATION>
</RESUME>

```

### 2.1.3 Functions on XML Documents

Based on the XML data model presented in Section 2.1.1, we define additional functions on XML documents. Let  $\mathbf{x}$  be an XML document from a collection  $\mathcal{X}$  of XML documents:

- Function  $Depth : (\mathbf{V} \cup \mathbf{X}) \rightarrow \mathbf{Z}^+$  computes the depth of an element in  $\mathbf{x}$ . The depth of the root is zero. The depth of the whole tree is the maximum depth of its vertices.
- Function  $Degree : (\mathbf{V} \cup \mathbf{X} \cup Set(\mathbf{X})) \rightarrow \mathbf{Z}$  computes the degree of a vertex (its number of children vertices), the degree of an XML document (the maximum degree of all its vertices), and the degree of a set of documents (the maximum degree of all of its documents).
- Function  $IsLeaf : \mathbf{V} \rightarrow \mathbf{boolean}$  returns *true* if an element does not have any children, i.e.  $Children(v) = \emptyset$ , *false* otherwise.



- Function  $Parent : \mathbf{V} \rightarrow \mathbf{V}$  computes the parent of a vertex. The parent of the root is  $\perp$ .
- Function  $Ancestors : \mathbf{V} \rightarrow \mathbf{V}^*$  computes the sequence of vertices along the path from  $root$  to a vertex:

$$Ancestors(v_k) := \{\langle v_0, \dots, v_k \rangle \mid v_j \in Children(v_{j-1}), 1 \leq j \leq k, v_0 = \mathbf{X}.root\}$$

- Function  $Descendants : \mathbf{V} \rightarrow Set(\mathbf{V})$  computes the set of vertices that are reachable from a vertex in  $\mathbf{X}$ , i.e.,

$$Descendants(v_0) := \{v_i \mid \exists v_1 \cdots v_{i-1} : (v_j \in Children(v_{j-1}), 1 \leq j \leq i)\}$$

In addition, we define functions that operate on an XML document tree  $\mathbf{X}$ , similar to the APIs in DOM [W3C00a] which is based on the Infoset data model:

- The function  $Create(\mathbf{X}, e)$  creates a new node labeled with  $e$ .
- The function  $Delete(v)$  deletes the node  $v$  from the document tree  $\mathbf{X}$ .
- The function  $AppendChild(p, c)$  appends the node  $c$  as the last child of the node  $p$  in the document tree  $\mathbf{X}$ . If  $c$  is in  $\mathbf{X}$ , it is first deleted before being inserted into  $\mathbf{X}$ .
- The function  $InsertFirst(p, c)$  inserts the node  $c$  as the first child of the node  $p$  in the document tree  $\mathbf{X}$ . If  $c$  is in  $\mathbf{X}$ , it is first deleted before being inserted into  $\mathbf{X}$ .

- The function *InsertAfter*( $n, r$ ) inserts the node  $n$  as a sibling immediately after the node  $r$  in the document tree  $X$ . If  $n$  is in  $X$ , it is first deleted before being inserted into  $X$ .

## 2.2 Document Type Definition (DTD)

In Section 2.2.1, we describe a standard format used for XML documents called Document Type Definition (DTD). In Section 2.2.2, we give a formal model for a DTD. Section 2.2.3 defines validity of an XML document with respect to a DTD.

### 2.2.1 Background

An XML document can have an optional *Document Type Definition (DTD)* which describes admissible relationship among elements. A DTD specifies the content model of an element - choices of its constituent elements (" $|$ "), and the number of occurrences of these constituent elements (whether it can occur zero times (?), exactly one time, at least one time (+), or more than one time (\*)). It can also specify the set of attributes of an element, and whether an attribute is optional (#IMPLIED), required (#REQUIRED) or fixed (#FIXED), and its optional default values.

An example DTD for the XML document in Section 1 is given below. It specifies that the XML document is a **RESUME** document which consists of an **OBJECTIVE** element

followed by an EDUCATION element. The content of OBJECTIVE is text (PCDATA). The element EDUCATION is made up of at least one DEGREE element. There are two choices for the structure of element DEGREE. It can consist of ORGANIZATION optionally followed by SPECIALIZATION, or it can consist of ORGANIZATION followed by GPA. The attributes of DEGREE are *title* and *date*, both of which are mandatory.

```

<!DOCTYPE RESUME (OBJECTIVE, EDUCATION)>
<!ELEMENT OBJECTIVE (#PCDATA)>
<!ELEMENT EDUCATION (DEGREE+)>
<!ELEMENT DEGREE (ORGANIZATION,SPECIALIZATION?) |
(Organizational,GPA)>

<!ATTLIST DEGREE
TITLE #PCDATA, #REQUIRED
DATE #PCDATA, #REQUIRED>
<!ELEMENT ORGANIZATION (#PCDATA)>
<!ELEMENT SPECIALIZATION (#PCDATA)>
<!ELEMENT GPA (#PCDATA)>

```

## 2.2.2 Formal Model for DTD

Here we describe a formal model for DTDs. We assume a set of language elements  $C = \{ " + " , " , " , " | " , " ( " , " ) " , " * " , " ? " , \epsilon , " = " \}$ . " + " , " , " , " \* " , " | " " ) " , " ( " are union, concatenation, Kleene closure, choice and nesting, respectively.

**Definition 2.2 (DTD).** A DTD is denoted as  $(E, root, Type, Attr)$  where

- $E$  is a set of element names.
- $root$  is a distinguished element name, the document type.
- $Type : E \rightarrow \alpha$  is a function that gives the type of an element by a regular expression  $\alpha$ :

$$\alpha := S \mid e \mid \epsilon \mid \alpha + \alpha \mid \alpha, \alpha \mid \alpha^* \mid (\alpha)$$

$e$  is an element in  $E$ ,  $S$  is a string.

- $Attr : E \rightarrow A$  is a function that gives the set of attributes of an element. The domain of attribute values is  $S$ . □

The content model of an element can be expressed as regular expressions, as illustrated below:

DTD Content Model	Regular Expression
<code>&lt;!ELEMENT e (e1)&gt;</code>	$e = e1$
<code>&lt;!ELEMENT e (e1   e2)&gt;</code>	$e = e1 + e2$
<code>&lt;!ELEMENT e (e1,e2)&gt;</code>	$e = e1, e2$
<code>&lt;!ELEMENT e (e1?)&gt;</code>	$e = \text{empty} + e1$
<code>&lt;!ELEMENT e (e1+)&gt;</code>	$e = e1, e1^*$
<code>&lt;!ELEMENT e (e1*)&gt;</code>	$e = e1^*$

Let  $CM2RE : E \times C \dots E \times C$  be a function that takes a list of element names and their content models, and expresses them as a regular expression.

**Example 2.2: DTD**

Take the DTD given in Section 2.2.1 as an example. It is denoted as  $(E, root, Type, Attr)$  where

- $E = \{ \text{RESUME, OBJECTIVE, EDUCATION, DEGREE, ORGANIZATION, SPECIALIZATION, GPA} \}$
- $root = \text{RESUME}$
- $Type$  is determined by the following mappings:
  - $\text{RESUME} \rightarrow (\text{OBJECTIVE, EDUCATION})$
  - $\text{EDUCATION} \rightarrow (\text{DEGREE, DEGREE}^*)$
  - $\text{DEGREE} \rightarrow (\text{ORGANIZATION, SPECIALIZATION}?) + (\text{ORGANIZATION, GPA})$
  - $\text{ORGANIZATION} \rightarrow \text{S}$
  - $\text{SPECIALIZATION} \rightarrow \text{S}$
  - $\text{GPA} \rightarrow \text{S}$
- $Attr$  is the following function:
  - $\text{DEGREE} \rightarrow \{\text{title, date}\}$

**2.2.3 Validity**

A DTD schema describes permissible structures of XML documents. XML documents satisfying this condition are *valid* with respect to the DTD.

**Definition 2.3 (Validity).** *An XML document  $X$  is valid with respect to a DTD  $D$  if there is a mapping  $\mu : V \rightarrow E$  such that:*

- $\mu(X.root) = D.root$
- Children vertices of a vertex  $v$  in  $X$  satisfy the content model of  $\mu(v)$  in  $D$ :

$$\forall v \in V: (Children(v) = \langle v_1, \dots, v_n \rangle) \Rightarrow \mu(v_1) \cdots \mu(v_n) \in L(Type(\mu(v)))$$

where  $L(\alpha)$  is the language defined by the regular expression  $\alpha$ .

- An attribute of a vertex  $v \in X.V$  is defined in  $\mu(v)$ :

$$\forall v \in V: (a \in X.Attr(v)) \Rightarrow a \in D.Attr(\mu(v))$$

□

## 2.3 Tree Schema

Our goal is to discover a majority schema for a collection of XML documents.<sup>1</sup> We would like to have a formalism that is expressive enough to describe a majority schema, and is in a form similar to XML documents to facilitate the transformation process (Chapter 5). Since a majority schema describes only prevalent structures among documents, it suffices that the formalism for a majority schema is less expressive than for a DTD. In view of this, we introduce the notion of a *tree schema* which is less expressive than a DTD. We use XML syntax to describe this schema.

---

<sup>1</sup>The definition of majority schema is given in Section 4.1.2.

We first describe the semantics of a tree schema, followed by its formal definition. Then we show its relationship to a DTD and define conformance of an XML document with respect to a tree schema.

### 2.3.1 Description

A tree schema can be interpreted as a simplified DTD in form of a tree with certain information not modeled. (1) Ordering information between sibling vertices is ignored. Hence, a tree schema is an unordered tree. (2) Grouping information among siblings is not considered. For instance, the content model  $(e1*, e2*)$  can be modeled in a tree schema, but not  $(e1, e2)*$ . (3) Choices are not modeled since we are interested in discovering the most prevalent structures.

A tree schema is a labeled tree. Each vertex in a tree schema corresponds to an element in a DTD. The content model of an element in the DTD is encoded in the attribute *content* of the vertex in the tree schema. For example, in the following fragment of a tree schema, the content model of the element  $e1$  under  $e$  is  $"*"$ , i.e.  $e1$  can occur multiple times in  $e$ . The content model of  $e2$  under  $e1$  is  $"?"$ , i.e. it is optional. The content model of the element  $e3$  under  $e$  is  $\epsilon$ , i.e.  $e3$  occurs exactly once in  $e$ . Hence elements in a tree schema are decorated by their content model.

A tree schema

```
<e>
  <e1 content="*">
    <e2 content="?" />
  </e1>
  <e3 content="" />
</e>
```

Since a schema describes permissible structures of XML documents, we first need to define the basic units of schematic structures of an XML document. Similar to [WL98], we view an XML document (which is an ordered tree) as a sequence of paths. We distinguish two types of paths, namely label paths and node paths. A label path is a sequence of labels whereas a node path is a sequence of nodes from the root of an XML document.

An XML document conforms to a tree schema if all its label paths can be found in the tree schema. For example, the following XML document conforms to the above tree schema.

An XML document conforming to the above tree schema

```
<e>
  <e1>
    <e2 />
  </e1>
  <e1 />
  <e3 />
</e>
```



### 2.3.2 Formalism

A label path is a concatenation of labels.

**Definition 2.4 (Label Path).**  $p = e_1 \circ \dots \circ e_n$  is a label path of length  $n$  over  $E$  iff  $\forall i: (e_i \in E \wedge 1 \leq i \leq n)$ . The size (or length) of  $p$  is denoted by  $|p|$ .

Its  $k$ -subpath,  $0 \leq k \leq n-1$ , determined by the function  $Subpath(p, k) : E^+ \times \mathbf{Z} \rightarrow E^*$ , is  $p_k = e_1 \circ \dots \circ e_{n-k}$ . In particular, it is a 0-subpath of itself.  $p$  is called a superpath of  $p_k$ .  $\square$

A node path in an XML document is a sequence of nodes in the document.

**Definition 2.5 (Node Path).**  $p_v = \langle v_1 \dots v_n \rangle$  is a node path in an XML document  $X$  denoted as  $p_v \in X$ , iff  $\langle v_1 \dots v_{n-1} \rangle \in Ancestors(v_n)$ . Hence, a node path may not necessarily start from the root.

Its  $k$ -subpath,  $0 \leq k \leq n-1$ , determined by the function  $Subpath(p, k) : \mathbf{V}^+ \times \mathbf{Z}^+ \rightarrow \mathbf{V}^*$ , is  $p_k = \langle v_1, \dots, v_{n-k} \rangle$ . In particular, it is a 0-subpath of itself.  $p_v$  is called a superpath of  $p_k$ .

A label path of  $p_v$  is  $Label(v_1) \circ \dots \circ Label(v_n)$ , computed by the function  $LabelPath : \mathbf{V}^* \rightarrow E^*$ . As a shorthand, we use the same notation  $LabelPath(v_n)$  to denote the label path of the node path from the root to the vertex  $v_n$ . Note that several node paths may have the same label path.  $\square$

A label path  $p_e \in E^*$  is a label path in  $X$ , denoted by  $p_e \in X$ , iff there is a node path  $p_v \in X$  such that  $LabelPath(p_v) = p_e$ . For simplicity, given a set  $\mathcal{X}$  of XML documents, we also use the notation  $p \in \mathcal{X}$  to denote that a path  $p$  is found in some document in  $\mathcal{X}$ .

As in [WL98], a *set* of paths starting with the same label can be "glued" together into an unordered tree whereas a *sequence* of paths can be glued into an ordered tree. For example, in Figure 2.1, the unordered tree is obtained from the set of label paths shown. Let the function *Paths2Tree* build the unordered/ordered tree from a set/sequence of paths.

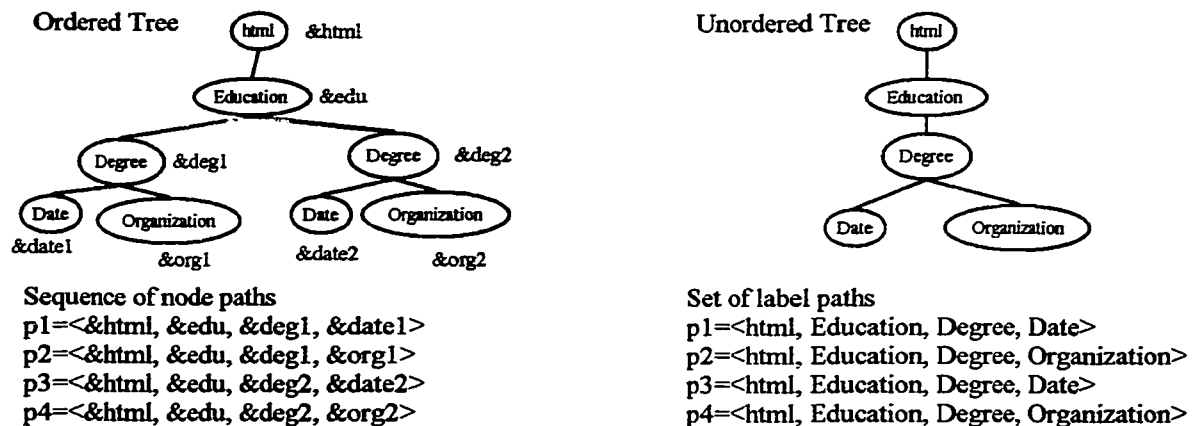


Figure 2.1: Trees and node paths

An ordered tree can be built from a sequence of node paths. An unordered tree can be built from a set of label paths. The ID of a node along a node path is shown as  $\&obj$ .

Now we can give the formalism for a tree schema:

**Definition 2.6 (Tree Schema).** *A tree schema is an XML document  $S = (V, root, Label, Children, Attr, AttrVal)$  with the following properties:*

- *A tree schema is an unordered tree.  $Children : V \rightarrow Set(V)$  is a function that computes the set of children of a vertex.*
- *There are no redundant label paths in a tree schema:*

$$\forall p = \langle root, p_1, \dots, p_m \rangle, q = \langle root, q_1, \dots, q_n \rangle \in S:$$

$$(p \neq q \Rightarrow LabelPath(p) \neq LabelPath(q))$$

- *The value of  $v.content$  gives the content model of a vertex  $v$  within its parent vertex.* □

The content model of an element is stored in the element's *content* attribute. Thus, the set of attribute names is  $\{content\} \subseteq A$ .

**Example 2.3: Tree Schema**

A tree schema for the XML document in Section 2.1.2 can be modeled as a tree schema  $S$  with:

- $V = \{v_1 \cdots v_7\}$
- $root = \text{RESUME}$
- The function *Label* determines the mappings:
  - $v_1 \rightarrow \text{RESUME}$
  - $v_2 \rightarrow \text{OBJECTIVE}$
  - $v_3 \rightarrow \text{EDUCATION}$
  - $v_4 \rightarrow \text{DEGREE}$
  - $v_5 \rightarrow \text{ORGANIZATION}$
  - $v_6 \rightarrow \text{SPECIALIZATION}$
  - $v_7 \rightarrow \text{GPA}$
- The function *Children* determines the mappings:
  - $v_1 \rightarrow \langle v_2, v_3 \rangle$
  - $v_3 \rightarrow \langle v_4 \rangle$
  - $v_4 \rightarrow \langle v_5, v_6, v_7 \rangle$
  - $v \rightarrow \emptyset$ , otherwise
- The function *Attr* determines the mappings:
  - $\forall v \in V: (v \rightarrow \{\text{content}\})$

- The function *AttrVal* determines the mappings:

$$v_2 \times \text{content} \rightarrow \epsilon$$

$$v_3 \times \text{content} \rightarrow \epsilon$$

$$v_4 \times \text{content} \rightarrow " + "$$

$$v_5 \times \text{content} \rightarrow \epsilon$$

$$v_6 \times \text{content} \rightarrow "?"$$

$$v_7 \times \text{content} \rightarrow \epsilon$$

### 2.3.3 Relationship of Tree Schema with DTD

A tree schema can be converted to a DTD. There is a subtlety in this process. Based on the Homonyms Property (Section 1.2.3), concept nodes of the same label in the tree schema refer to different concepts. Naming conflicts thus occur in deriving elements for these concept nodes in the DTD.

Naming conflicts can be resolved by renaming vertices to another set of labels, if necessary, so that no two vertices have the same label in a tree schema. We assume that there is a renaming function that takes a concept node from the tree schema and computes a label for its corresponding element in the DTD. If there is no naming conflict for the concept node, its label in the DTD is unchanged. Otherwise, it is renamed to another label. For instance, two vertices of the same label DATE may be renamed to elements DATE1 and DATE2, respectively.

Each concept node in the tree schema has a *rename* attribute which stores the (re-named) label of the concept node. Hence,

$$\forall u, v \in V: (u \neq v \Rightarrow u.rename \neq v.rename)$$

Let  $E$  be a set of element names (labels) in a tree schema. Let  $E'$  be the set of element names derived from  $E$  by a renaming function that resolves naming conflicts. A tree schema  $S$  can be converted to a DTD  $D$  by the following mapping:

- The elements in  $D$  are the renamed labels in  $S$ :  $D.E' := \{v.rename \mid v \in V\}$
- $D.root := S.root.rename$
- The content models of elements in  $D$  are given by the attribute *content* of vertices in  $S$ :

$$Type(e) = CM2RE(Rename(v_1), content(v_1), \dots, Rename(v_n), content(v_n)),$$

$$Rename(v) = e, Children(v) = Set(v_1 \cdots v_n)$$

- Attributes different from content are not considered in  $S$ , i.e.,  $D.A = \emptyset$ .

Not all DTDs can be converted to a tree schema because a tree schema is less expressive than a DTD. Choice, grouping and order are not modeled in a tree schema.

### 2.3.4 Conformance

A tree schema describes permissible label paths in XML documents. An XML document satisfying this condition is said to *conform* to the tree schema.

**Definition 2.7 (Conformance).** *Given an XML document  $X$  and a tree schema  $S$ .*

*A node path  $p_x \in X$  conforms to a node path  $p_s \in S$  (or conforms to  $S$ ) if*

$$\text{LabelPath}(p_x) = p_s.$$

*An XML document  $X$  conforms to a tree schema  $S$  if there is a conformance mapping  $\mu : X.V \rightarrow S.V$  such that:*

- $\mu(X.\text{root}) = S.\text{root}$ .
- *Vertices in the document can only be mapped to vertices of the same label in the tree schema:*

$$\forall v \in X.V: (\text{Label}(v) = \text{Label}(\mu(v)))$$

- *All child nodes of  $v \in X.V$  have to be mapped to some child nodes of  $u = \mu(v) \in S$  of the same label:*

$$\forall v \in X.V:$$

$$(\text{Children}(v) = \langle v_1, \dots, v_n \rangle \wedge$$

$$u = \mu(v) \wedge \text{Children}(u) = \{u_1, \dots, u_m\}) \Rightarrow$$

$$(\forall v_i: (\exists u_j: (\text{Label}(u_j) = \text{Label}(v_i))))$$

- *Child nodes of  $v$  in  $X$  satisfy the content model of  $\mu(v)$  in  $S$ :*

$\forall v \in X.V:$

$$(Children(v) = \langle v_1, \dots, v_n \rangle \wedge$$

$$u = \mu(v) \wedge Children(u) = \{u_1, \dots, u_m\} \wedge$$

$$\alpha = CM2RE(u_1, u_1.content, \dots, u_m, u_m.content)) \Rightarrow$$

$$(\mu(v_1), \dots, \mu(v_n)) \in L(\alpha)$$

□



## Chapter 3

# Converting HTML Documents to XML Form

The goal of the document conversion process is to extract semantic information from topic specific HTML documents and to encode such information in XML documents. This is related to the work on wrappers methods. The document conversion process is realized by the Document Converter component of Quixote [CS99]. Semantics are encoded in the XML documents in two ways: (1) XML tags which carry semantic information of their associated text, and (2) tree structures of XML documents matching the logical information content of the documents.

In Section 3.1, we first describe how the information content of an HTML document is captured. In Section 3.2, we present the data extraction approach. In Section

3.3, we give the results of an analytical and empirical study of the data extraction approach. In Section 3.4, we describe related work on and compare them to the Document Converter.

## 3.1 Document Information Content

The information content of HTML documents is described by concepts and the context of the concepts, similar to an ontology ([FMHA98, GMV99]). Concepts are names of relevant concepts specific to the topic. They reflect the meanings of information objects (or concept nodes in a tree) that typically constitute a topic specific document at different levels of abstraction. We may use the term *concept* to refer to the name of a concept or an information object related to a concept when the context is clear. High level concepts are described by lower level concepts in a tree-like hierarchy. The context of an information object is described by its textual content and its relationship with other information objects in the document tree. For example, the context of the information object `DATE` under `EDUCATION` in the following fragment of a document tree includes (1) its textual content `1993`, (2) lower level information objects `DEGREE` and `ORGANIZATION` that further describe it, and (3) the high level information object `EDUCATION` it details information about. The context of this `DATE` object is different from that of the `DATE` object under `EXPERIENCE`. The former describes a chronological information object for the educational background of a person while the latter for her working experience. The context of the first `DATE` object is not

only "1993", but also lower level concepts detailing it and the higher level concept it refines. In other words, there may be homonyms among concept names.

```
<EDUCATION>
  <DATE val="1993">
    <DEGREE val="B.Sci."/>
    <ORGANIZATION val="UCD"/>
  </DATE>
</EDUCATION>
<EXPERIENCE>
  <DATE val="1994">
    <TITLE val="Research Assistant"/>
    <ORGANIZATION val="UCD"/>
  </DATE>
</EXPERIENCE>
```

Although the collection of XML documents is heterogeneous in terms of how the documents are structured, since HTML documents pertain to a particular topic, the information content of the XML documents is quite homogeneous. We assume that the user has some domain knowledge on the topic of the documents. This is captured in three ways: (1) concept names specific to the topic of the documents, (2) information on how to identify concepts from text, (3) and permissible or prohibited structures on concepts.

**Concept Names:** We assume the user specifies a set of concept names for the topic. This is a reasonable assumption because, in practice, the user has some knowledge on what the topic is about. For example, in a *résumé*, one would expect some educational background information about the institutions where the person received her

degree(s). Thus, ORGANIZATION, DEGREE and EDUCATION would be concepts for the topic. Since concept names are provided by a single user initiating the document conversion process, we assume there are no synonyms among concept names. Concepts provide the domain of tags to be chosen as element names in XML documents, i.e., they are the labels **E** of the XML documents to be obtained from HTML documents.

**Concept Instances:** The user gives information on how to identify concepts from text by concept instances. Concept instances specify text patterns as they might occur in the HTML documents either as keywords or as examples to a classifier (described in Section 3.2.2). For example, concept instances for the concept DEGREE can be B.Sci., MBA, M.Sci., Ph.D. etc. Some concept instances are often already present in order for a topic specific crawler to gather respective documents from the Web. In practice, identifying concept instances for concepts can be done by the user after inspecting a few of the retrieved HTML documents.

**Concept Constraints:** The user may have some knowledge on how concepts are related in the topic. This knowledge is captured by concept constraints. Concept constraints describe permissible or prohibited structures on concepts. They are optional input to the process and do not have to be complete. They can be used to improve the accuracy of data extraction. A complete specification of all permissible and prohibited relationships between concepts would basically resemble a schema for the XML documents. This would be much too restrictive for a flexible, easy to employ framework for translating heterogeneous HTML documents into XML documents.

Since our goal is to design a flexible framework, we assume minimal domain knowledge on the permissible or prohibited structures of concepts. Therefore, we use a simple mechanism of constraints to specify such domain knowledge. Concept constraints are specified in a first order predicate logic language with predicates to constrain permissible structures of concepts.

**Definition 3.1 (Concept Constraint).** *Let  $E$  denote concept names,  $e, e_1, e_2 \in E$ ,  $d, level \in \mathbf{Z}^+$ .  $L$  is a set of concept constraints. A concept constraint is a predicate of one of the following*

- *$MaxDepth(e, d)$  is true iff the depth of  $e$  in the document tree must not be larger than  $d$ .*
- *$MinDepth(e, d)$  is true iff the depth of  $e$  in the document tree must not be smaller than  $d$ .*
- *$NonAncestorDescendant(e_1, e_2, d)$  is true if  $e_2$  cannot be a descendant of  $e_1$  in a document within  $d$  levels. In particular, if  $d = \infty$ ,  $e_2$  cannot be a descendant of  $e_1$ . If  $d = 1$ ,  $e_2$  cannot be an immediate child of  $e_1$ .*
- *$NonSiblings(e_1, e_2)$  is true iff  $e_1, e_2$  must not be siblings in a document tree.  $\square$*

In practice, the user can gain an idea of how deeply nested the documents of the topic are by inspecting samples of the document collection or by apriori knowledge. Setting  $d$  in *NonAncestorDescendant* to a value greater than the maximum depth of the documents in effect sets it to  $\infty$ .

Concept constraints reflect the user's perception on how the information content should look like. They are soft constraints used to guide the Document Converter not to restructure the document tree in a way that violates these constraints. Since the input documents are authored by different people, there may be documents that violate the concept constraints. Concept constraints are *not* used to resolve this heterogeneity. Violations of these constraints can serve as a feedback to the user, but will not cause termination of the process. Instead, concept constraints are used by the Schema Miner (Chapter 4) to infer a schema that does not violate any concept constraints. In short, the Document Converter will not restructure a document tree in a way that violates concept constraints. The Schema Miner infers a schema that does not violate any concept constraints. If there are structures in the document collection that violate concept constraints, these structures will be converted to structures satisfying concept constraints in the schema by the Document Transformer.

## 3.2 Approach

The data extraction problem can be divided into two subproblems: (1) identify concepts from text in an HTML document, and (2) infer structures of the concepts in the document. Typically, the meaning of an object is buried in the text in an HTML document. The first task concerns identifying concepts from the HTML text and their associated textual contents. This is accomplished by the processes *tokenization* and *concept identification* (Sections 3.2.1 and 3.2.2). Although the HTML document

can be represented as a tree, information objects in the tree are not organized in a way that reflects the logical layout of the document. This is because HTML is a markup language for visual rendering, not for describing information content. The goal of the second task is to restructure the objects in the document tree so that the tree structure of the resulting XML document matches the logical layout of the information content of the HTML document. This is accomplished by the *grouping* and *consolidating* restructuring rules (Sections 3.2.3 and 3.2.4).

The function *DocumentConversion* below describes the overall process. The user specifies the following domain knowledge: concept instances *C* (Section 3.2.2), concept constraints *L*, punctuation delimiters *punc* (Section 3.2.1), group tags *group* (Section 3.2.3), list tags *list* (Section 3.2.4) for tokenization and the restructuring rules. We assume the user uses tools like Tidy [Rag98] to translate an HTML document into one satisfying the HTML specification which can be thought as a well-formed XML document with tags being HTML markup tags. Without loss of generality, as a technical convenience, each node in the HTML tree has an attribute named *val* which stores the values of all its text child nodes, i.e. its textual content. The Document Converter then executes tokenization, concept identification, grouping and consolidating restructuring rules in sequence to convert the input HTML document *X* (now as well-formed XML document) into an XML document with tag names from concept names in *C* whose tree structure matches its logical information content. Each of the processes will be described in detail in the following sections.

**Algorithm 3.1: DocumentConversion**

*Function DocumentConversion*(*X*, *C*, *L*, *punc*, *group*, *list*) : *X*

**begin**

*X* = *Tokenization*(*X*, *punc*)

*X* = *ConceptIdentification*(*X*, *C*)

*X* = *Grouping*(*X*, *group*, *L*)

*X* = *Consolidating*(*X*, *list*, *L*)

**return** *X*

**end**

### 3.2.1 Tokenization

According to the Concept Hierarchy Property (Section 1.2.3), an information object related to a high level concept is refined by information objects related to lower level concepts. We observe that very often information objects are separated by punctuation delimiters, e.g., semicolon ";", comma ",", etc. For example, "University of California at Davis, B.Sci.(Computer Science), June 1996, GPA 3.8/4.0" represents an information object about the educational background in a résumé. It is refined by lower level concepts related to the organization granting the degree, the degree, date and GPA. Based on this observation, tokenization decomposes a text into tokens according to the punctuation delimiters given by the user.



The function *Tokenization* implements tokenization. It is recursively applied to each node in the input tree  $X$  in a top-down fashion from the root to leaf nodes. For each node  $v$ , it takes its textual content (value of the attribute *val*)  $c_1 \dots c_n$ , if *val* exists at the node. The punctuation delimiters *punc* are used to identify tokens  $c_{j+1} \dots c_i$  in the text node. A new node with label `TOKEN` and attribute *val* =  $c_{j+1} \dots c_i$  is then created as a child of the text node.

### Algorithm 3.2: Tokenization

*Function Tokenization*( $X, \text{punc}$ ) :  $X$

**begin**

$v = X.\text{root}$

*(This is a text node, i.e. a leaf node.)*

**if**  $v.\text{val} \neq \epsilon$

**then**

*( $v.\text{val}$  consists of  $n$  characters  $c_1 \dots c_n$ )*

$v.\text{val} = c_1 \dots c_n$

$j = 0$

*(For each character  $c_i$  in  $v.\text{val}$ )*

**for**  $i = 1$  **to**  $n$  **do**

*( $c_i$  is a punctuation delimiter or the last character.)*

**if**  $c_i \in \text{punc} \vee i == n$

**then**

$\text{token} = c_{j+1} \dots c_i$

```

    u = Create(x, TOKEN)
    u.val = token
    AppendChild(v, u)
    j = i
  fi
od
else
  (Recursively apply Tokenization in a top-down fashion)
  for each u  $\in$  Children(v) do
    x = Tokenization(T(u), punc)
  od
fi
return x
end
```

After tokenization, new text nodes are created for each text node and are grouped under a node labeled with `TOKEN`.

### 3.2.2 Concept Identification

For each token obtained through tokenization, the token is checked if it can be related to a concept. There are several tools that have been developed in the information

retrieval arena, e.g., [SC99, FMHA98], and that are suitable for this task. In Quixote, concepts are associated with text by concept instances. There are two forms of concept instances: (1) as keywords, or (2) as examples to a multinomial naïve Bayes classifier [Cha00].

### Concept Instances Based On Keywords

For concept instances based on keywords, the user specifies keywords that identify a concept from a text. For example, keywords `university`, `company`, `co.`, `ltd.` can be used to identify the concept `ORGANIZATION` in a résumé. If a keyword is found in a text, the concept associated with the keyword is identified from the text and the text is the textual content of the concept.

Let  $C = \{(e, k) \mid e \in E, k \in S\}$  denote a set of concept instances. It is a set of tuples  $(e, k)$  that associate the keywords  $k$  with the concept name  $e$ . A keyword can consist of more than one word.

The function *ConceptInstanceByKeywords* takes a string *text* and a set of concept instances  $C$  and computes the set of concepts identified from the string and its associated textual content  $r$ . A string consists of words, which are consecutive characters separated by the space character  $\sqcup$ . It scans the words  $w_1 \dots w_n$  in *text*. If  $w_i \sqcup \dots \sqcup w_{i+p-1}$  matches the keyword  $k$ , the concept  $e$  associated with the keyword is related to  $w_i \sqcup \dots \sqcup w_{i+p-1}$  and all the words afterwards in *text* until another concept

is identified. All words before the first concept are associated with it. The string  $r$  computed is a concatenation of tuples of the form  $(re, rk)$  where  $re$  is a label and  $rk$  its associated words in  $text$ .

**Algorithm 3.3: ConceptInstanceByKeywords**

*Function*  $ConceptInstanceByKeywords(text, C) : S$

**begin**

*(re is the current label and rk is its associated text)*

$r = re = rk = \epsilon$

*(text consists of words  $w_i$  separated by space  $\sqcup$ )*

$text = w_1 \sqcup \dots \sqcup w_n$

$i = 1$

**while**  $i \leq n$  **do**

*(The next  $p$  words,  $w_i \sqcup \dots \sqcup w_{i+p-1} = k$ , in text match the keyword  $k$ )*

**if**  $\exists (e, k) \in C, p \in \mathbf{Z}^+ : (w_i \sqcup \dots \sqcup w_{i+p-1} = k)$

**then**

**if**  $re \neq \epsilon$

**then**

$r = r \circ (re, rk)$

$rk = \epsilon$

**fi**

$re = e$

$rk = rk \circ \sqcup \circ k$

```

         $i = i + p$ 
      else
         $rk = rk \circ \sqcup \circ w_i$ 
         $i = i + 1$ 
      fi
    od
    if  $re \neq \epsilon$ 
      then
         $r = r \circ (re, rk)$ 
      fi
    return  $r$ 
  end

```

### Concept Instances Based On Bayes Classifier

For concept instances based on a naïve Bayes classifier, the user gives examples on how to associate text with concepts. For example, the user may label the text "University of California at Davis" as the concept ORGANIZATION, "Davis, California" as LOCALE etc. Based on these examples, the Bayes classifier computes the statistics of associating words in the text with concepts. Given a new text, the classifier classifies it to the concept with the highest probability using statistics computed from the examples.

The concept instances  $\mathcal{C} = \{(e, k)\}$  give a set of  $(e, k)$  tuples as examples for the Bayes classifier.  $k$  is a text and  $e$  the concept it is associated with. In Bayes classifier terminology,  $k$  is a document and  $e$  the class of the document to which it belongs. Let  $\{t\}$  denote the terms in the documents, i.e. words in a string. One can compute the number of occurrences of term  $t$  in a document  $k$ , denoted by  $n(k, t)$ . One can also estimate the apriori probability of a class  $e$  by counting the number of occurrences of  $e$  in  $\mathcal{C}$ , denoted by  $Prob(e)$ .

Based on  $\mathcal{C}$ , one can estimate the probability of having the term  $t$  in a class  $e$  at least once with the Laplace corrected estimate by [Lap95]:

$$\Theta_{e,t} := \frac{1 + \sum_{k,(e,k) \in \mathcal{C}} n(k, t)}{|\{t\}| + \sum_t \sum_{k,(e,k) \in \mathcal{C}} n(k, t)}$$

A document is modeled as a bag of words. A document is generated by choosing terms from a set of words. In the multinomial model, the conditional probability of generating document  $k$  from class  $e$  is:

$$Prob(k | e) := \left( \frac{|k|!}{\prod_{t \in k} n(k, t)!} \right) \prod_{t \in k} \Theta_{e,t}^{n(k,t)}$$

This is based on the assumption that terms in  $k$  are conditionally independent.

By Bayes rule, we have

$$Prob(e | k) Prob(k) := Prob(k | e) Prob(e)$$

Since  $Prob(k)$  is constant for  $e$ , we can compare  $Prob(e | k)$  simply by

$$Prob(k | e) Prob(e)$$

Given a document  $text$ , one can then classify  $text$  to the class  $e_i$  such that  $Prob(e_i | text)$  is maximum among all  $Prob(e_j | text)$ ,  $e_j \in E$ .

The function *ConceptInstanceTrainBayes* first collects statistics on  $Prob(e)$ ,  $\Theta_{e,t}$  for all labels  $e$  in  $C$ . When a new text is given, the function *ConceptInstanceByBayes* classifies it to a label  $e_i$  in  $C$  with the highest posterior probability based on the statistics collected by the function *ConceptInstanceTrainBayes*. In case there is a tie, an arbitrary label is chosen by the function *Pick* which selects an arbitrary element from the set.

**Algorithm 3.4: ConceptInstanceTrainBayes**

*Procedure ConceptInstanceTrainBayes*( $C$ )

**begin**

**for each**  $(e, k = t_1 \dots t_n) \in C$  **do**

$n(e, t_i) = n(e, t_i) + 1, 1 \leq i \leq n$

**od**

**for each**  $(e, k) \in C$  **do**

*Compute*  $Prob(e)$

**od**

**for each**  $(e, k = t_1 \dots t_n) \in C$  **do**

*Compute*  $\Theta_{e,t_i}, 1 \leq i \leq n$

**od**

**end**

**Algorithm 3.5: ConceptInstanceByBayes**

*Function* *ConceptInstanceByBayes*(*text*, *C*) : **S**

**begin**

**for each**  $e \in C$  **do**

*Compute*  $Prob(text | e)$

**od**

$e_i = Pick\{e_i \in E | Prob(e_i | text) \geq Prob(e_j | text), e_j \in E\}$

$r = (e_i, text)$

**return**  $r$

**end**

**Identification of Concepts**

Given a function *ConceptInstance* that takes a string and returns a set of (**concept**, **text**) pair where **concept** is a concept and **text** the corresponding textual content of the concept in the string (it can be realized by *ConceptInstanceByKeywords* or *ConceptInstanceByBayes*), the function *ConceptIdentification* realizes the concept identification process. It is recursively applied to each node in an input document **X** in a top-down fashion. For each node  $v$  in **X**, it calls function *ConceptInstance* to identify concepts of its attribute  $v.val$  (its textual content). If no concept can be associated with  $v.val$ ,  $v.val$  is propagated to the attribute  $val$  of its parent. If there is at least one (**concept**, **text**) identified, a new node is created for each concept.



The label of the new node is **concept** and the value of its attribute *val* is **text**. The textual content of a node *v* is now "consumed", i.e. either propagated to its parent or replaced by some concept nodes. Node *v* is hence deleted.

**Algorithm 3.6: ConceptIdentification**

*Function ConceptIdentification(X, C) : X*

**begin**

*v = X.root*

*(This is a text node)*

**if** *v.val*  $\neq \epsilon$

**then**

*con = (con<sub>1</sub>, text<sub>1</sub>) ... (con<sub>n</sub>, text<sub>n</sub>) = IdentifyConcept(v.val, C)*

*(No concept is identified. Propagate v.val to its parent's attribute val)*

**if** *con =  $\epsilon$*

**then**

*Parent(v).val = Parent(v).val  $\circ$  v.val*

**else**

*(At least one concept is identified)*

*lastConcept =  $\epsilon$*

**for** *i = 1 to n* **do**

*w = Create(X, con<sub>i</sub>)* (create a new node for this concept)

*w.val = text<sub>i</sub>* (set its attribute val)

*(Add the new node as next sibling of v)*

```

    if  $lastConcept = \epsilon$ 
      then
         $InsertAfter(w, v)$ 
      else
         $InsertAfter(w, lastConcept)$ 
      fi
       $lastConcept = w$ 
    od
     $Delete(v)$ 
  fi
else
  (Recursively apply ConceptIdentification in a top-down fashion)
  for each  $u \in Children(v)$  do
     $T(u) = ConceptIdentification(Tree(u), C)$ 
  od
fi
return X
end

```

Concepts are identified *after* tokenization. The rationale is that since tokenization partially identifies objects describing a higher level object, applying concept instance identification to tokens can improve the accuracy of identifying information that the instances contain.

If punctuation delimiters are not used consistently or certain punctuation delimiters are not considered by tokenization, there may be more than one concept in a token. This is the case when *ConceptInstanceByKeywords* returns more than one  $(e, k)$  tuple. *ConceptIdentification* then decomposes this token into separate concept nodes.

If no concept can be associated with a token, the token is propagated to the attribute *val* of its parent. The justification is that child nodes detail information of their parent nodes. Propagating the token to its parent ensures that no textual content is lost, and such information is kept in the proper context during the document restructuring process.

### 3.2.3 Grouping Rule

After tokenization and concept identification, concepts are associated with text, and they are arranged according to the HTML document tree. However, the tree structure of the HTML document does not necessarily match its logical information content. The purpose of the Grouping Rule is to restructure the document tree to reflect its logical information content rather than its visual structure.

Certain block level HTML markup tags, called *group tags*, give hints to the grouping of semantically related objects. For example, **block** elements (such as `p` or `hr`), **list** elements (such as `dl`, `ul`), **heading** elements (such as `h1`, `...`, `h6`) and **table** elements (such as `tr`, `td`) are often used to divide text into information objects of the same

level of abstraction. Based on this observation, the Grouping Rule groups objects belonging to the same level of semantic and logical abstraction as siblings.

The function *Grouping* realizes the Grouping Rule. It is applied to each node in the document tree  $\mathbf{X}$  in a top-down fashion. For each node  $v$ , we search for its first child  $u_{start}$  whose label is one of the group tags *group* specified by the user. We locate its next sibling node of the same label (or the last child of  $v$  if there isn't any),  $u_{end}$ . All sibling nodes between  $u_{start}$  and  $u_{end}$  are grouped under a new node with the temporary label **GROUP** as a child of  $u_{start}$ . Groups of nodes related to the same level of abstraction thus "sink" in the document tree and are put into a logical nesting.

Concept constraints are taken into account so that restructuring does not violate any known concept constraint, as implemented by the functions *SinkViolateConstraint* and *PushViolateConstraint*. The function *SinkViolateConstraint*( $L, parent, child, \mathbf{X}$ ) ensures that appending *child* as a child of *parent* does not violate any concept constraint in  $L$ . The function *PushViolateConstraint*( $L, node, \mathbf{X}$ ) ensures that replacing the parent of *node* by *node* does not violate any concept constraint.

**Algorithm 3.7: Grouping**

*Function Grouping*(X, group, L) : X

**begin**

$v = X.root$

*(Locate the first child node of v labeled with a group tag.)*

$\langle u_1 \dots u_n \rangle = Children(X, v)$

$start = n$

**for**  $i = 1$  **to**  $n$  **do**

**if**  $u_i \in group$

**then**

$start = i$

**break**

**fi**

**od**

*("Sink" all siblings between nodes of the same group tag into a subtree)*

**for**  $end = start + 1$  **to**  $n$  **do**

*(Locate the next sibling of  $u_{start}$  labeled with the same group tag or as the last child of v)*

**if**  $Label(u_{end}) = Label(u_{start}) \vee end == n$

**then**

$g = Create(X, "GROUP")$

**for**  $i = start + 1$  **to**  $end$  **do**

**if**  $Label(u_i) \in X.E \wedge (i == n \vee Label(u_i) \neq Label(u_{start})) \wedge$

```

         $\neg \text{SinkViolateConstraint}(\mathbf{L}, u_{start}, u_i, \mathbf{X})$ 
        then
             $\text{AppendChild}(g, u_i)$ 
        fi
    od
     $\text{AppendChild}(u_{start}, g)$ 
     $start = end$ 
fi
od
    (Recursively apply Grouping in a top-down fashion)
for each  $u \in \text{Children}(v)$  do
     $\mathbf{X} = \text{Grouping}(\text{Tree}(u), \text{group}, \mathbf{L})$ 
od
return  $\mathbf{X}$ 
end

```

**Algorithm 3.8: SinkViolateConstraint**

*Function*  $\text{SinkViolateConstraint}(\mathbf{L}, \text{parent}, \text{child}, \mathbf{X})$  : **boolean**

**begin**

**if**  $\langle p_1, p_2, \dots, p_{k-1} \rangle \in \mathbf{X} \wedge \text{Label}(p_1) = a \wedge p_{k-1} = \text{parent}$

$\wedge \text{NonAncestorDescendant}(\text{Label}(\text{child}), a, k) \in \mathbf{L}$

**then**

$\text{violate} = \text{true}$

```

elsif  $v \in Children(parent) \wedge Label(v) = s$ 
     $\wedge NonSiblings(Label(child), s) \in L$ 
     $violate = true$ 
elsif  $Depth(parent) + 1 > d \wedge MaxDepth(Label(child), d) \in L$ 
     $violate = true$ 
elsif  $Depth(parent) + 1 < d \wedge MinDepth(Label(child), d) \in L$ 
     $violate = true$ 
else
     $violate = false$ 
fi
return  $violate$ 
end

```

**Algorithm 3.9: PushViolateConstraint**

*Function*  $PushViolateConstraint(L, node, X) : \text{boolean}$

**begin**

$gp = Parent(Parent(node))$

**if**  $\langle node, p_2, \dots, p_k \rangle \in X \wedge Label(p_k) = desc$

$\wedge NonAncestorDescendant(desc, Label(node), k) \in L$  **then**

$violate = true$

**elsif**  $v \in Children(gp) \wedge Label(v) = s \wedge NonSiblings(Label(node), s) \in L$

$violate = true$

**elsif**  $Depth(gp) + 1 > d \wedge MaxDepth(Label(child), d) \in L$

```

    violate = true
elsif  $Depth(gp) + 1 < d \wedge MinDepth(Label(node), d) \in L$ 
    violate = true
else
    violate = false
fi
return violate
end

```

To further capture the hints given by HTML tags, a rank can be associated with each group tag. For example, grouping right siblings of nodes marked with `h1` has a higher priority than grouping right sibling of nodes marked with `p` at the same level. Since each group sinks down and the rule operates in a top-down fashion, groups related to `p` nodes then will be considered at the next recursion. Function *Grouping* can be refined to take this into account by searching for the group tag with the highest rank among  $u_1 \dots u_n$  in the first **for** loop.

### 3.2.4 Consolidating Rules

The purpose of the Consolidating Rules is to replace nodes labeled with HTML tags and nodes introduced temporarily (e.g., by the Grouping Rule) in the document tree



by concept nodes. In doing so, it also restructures the tree so that it matches its semantic and logical information content.

The Consolidating Rule is applied in a bottom-up fashion. Each non-concept node is replaced by its children concept nodes. The manner it is replaced is based on four observations.

- *Rule 1:* The first rule is based on the domain knowledge that if some HTML tag exhibits a list structure (such as the list elements `ul`, `dl` or the `table` element), its child nodes are likely to be information objects at the same level of abstraction.
- *Rule 2:* A similar case is when sibling nodes carry the same XML element name. This is justified by the Regular Intradocument Format Property (Section 1.2.3) which states that within an information object, its content is described regularly. For example, in the following tree, one can infer that there are two groups of information objects `DEGREE, ORGANIZATION, DATE, THESIS` and `DEGREE, ORGANIZATION, DATE, GPA` under `EDUCATION` by recognizing the repeating concept nodes `DEGREE`.

```
<EDUCATION>
  <DEGREE val="M.Sci."/>
  <ORGANIZATION val="UCD"/>
  <DATE val="1996"/>
  <THESIS val="Semistructured Data"/>
  <DEGREE val="B.Sci."/>
  <ORGANIZATION val="Stanford University"/>
  <DATE val="1994"/>
```

```
<GPA val="3.9/4.0"/>
</EDUCATION>
```

- *Rule 3*: We can apply the same principle to the visual clues of HTML tags. Sibling concept nodes can be grouped together if they are marked up in the same way. For example, in the following tree, one can infer that there are two groups of information objects under EDUCATION since they are both marked up in the same way using the paragraph tag `<p>`.

```
<EDUCATION>
  <p>
    <ADVISOR val="Dr. Dole"/>
    <DEGREE val="M.Sci."/>
    <ORGANIZATION val="UCD"/>
    <DATE val="1996"/>
    <THESIS val="Semistructured Data"/>
  </p>
  <p>
    <DEGREE val="B.Sci."/>
    <ORGANIZATION val="Stanford University"/>
    <DATE val="1994"/>
    <GPA val="3.9/4.0"/>
  </p>
</EDUCATION>
```

- *Rule 4*: If none of the above cases applies, we can view the first object in a group of objects as describing the concept of this group. This is analogous to a topic sentence in which the first token describes the concept underlying the sentence and the remaining tokens refine the concept.<sup>1</sup>

---

<sup>1</sup>Alternatively, an artificial node can be created as the parent of a group of concept nodes. We do not choose this approach because this introduces many artificial nodes with no semantic information.

These four rules are realized in the function *Consolidating*. The function is applied to each node in the document tree  $X$  recursively in a bottom-up fashion. A concept node is left as it is. A non-concept leaf node is deleted from the document tree. An internal non-concept node is replaced by its child concept nodes by matching against the four rules in consecutive order. According to Rule 1, if it is labeled with a list tag in `list`, it is replaced by all of its child nodes. According to Rule 2, sibling concept nodes can be grouped by viewing repeating nodes of the same concept name as delimiters. Child concept nodes of a node  $v$  are grouped together in this way with the first concept node of each group being their representative. These groups then replace the node  $v$ . To check if Rule 3 applies, when a concept node replaces a non-concept node, *Consolidating* needs to keep track of the label of this non-concept node. We introduce a new attribute *prev* for each concept node to store this information. This information is necessary to identify groups of related sibling concept nodes that are marked up in the same way, i.e. they replace non-concept nodes of the same label. The mechanism to replace  $v$  by these concept nodes is the same as in Rule 2. If none of the above rules applies, Rule 4 is used to replace  $v$  by its first concept child node. Similar to the Grouping Rule, care is taken to ensure restructuring will not violate any known constraints. As mentioned, since documents are authored by different people, there may be document structures that violate concept constraints specified by the user. These structures will be converted by the Document Transformer to

---

Moreover, it loses sight of the information that the first information object among the group carries more weight compared with other information objects in the group.

conform to a schema discovered by the Schema Miner. The schema does not violate any concept constraints.

A pictorial explanation of the Consolidating Rule is to "push" up concept nodes to their parents. The four rules make sure that the tree structure after consolidating reflects the logical layout of its information content.

**Algorithm 3.10: Consolidating**

*Function Consolidating*(X, list, L) : X

**begin**

$v = X.root$

*(Recursively apply Consolidating in a bottom-up fashion)*

**for** each  $u \in Children(v)$  **do**

$T(u) = Consolidating(T(u), list)$

**od**

*(Do not replace a concept node)*

**if**  $Label(v) \in X.E$

**then**

**return** X

**fi**

*(Delete leaf non-concept nodes)*

**if**  $IsLeaf(v)$

**then**

```

    Delete(v)
    return X
fi
< u1⋯un >= Children(v)
(Rule 1: v is labeled with a list tag)
if Label(v) ∈ list
    then
        refNode = v
        for i = 1 to n do
            InsertAfter(ui, refNode)
            ui.prev = Label(v)
            refNode = ui
        od
        Delete(v)
    return X
fi
(Rule 2: Children of v are grouped together by
viewing repeating concept names as delimiters)
start = 1
found = false
for end = start + 1 to n do
    if Label(uend) = Label(ustart)
        then

```

```

    found = true
    for  $i = start + 1$  to  $end$  do
        if  $\neg SinkViolateConstraint(L, u_i, u_{start}, X)$ 
             $\wedge (Label(u_i) \neq Label(u_{start}) \vee i == end)$ 
            then
                AppendChild( $u_{start}, u_i$ )
            fi
        od
        start = end
    fi
od
if found == true
    then
        X = PushUpChild(v, X)
        return X
    fi

```

(Rule 3: Children of  $v$  are grouped if they are marked up in the same way)

```

start = 1
found = false
for  $end = start + 1$  to  $n$  do
    if  $u_{end}.prev = u_{start}.prev$ 
        then
            found = true

```

```

    for  $i = start + 1$  to  $end$  do
        if  $\neg SinkViolateConstraint(L, u_i, u_{start}, X)$ 
             $\wedge (Label(u_i) \neq Label(u_{start}) \vee i == end)$ 
            then
                 $AppendChild(u_{start}, u_i)$ 
            fi
        od
         $start = end$ 
    fi
od
if  $found == true$ 
    then
         $PushUpChild(v, X)$ 
    return  $X$ 
fi
(Rule 4: Replace v by its first concept child node that
does not violate any constraint)
 $found = false$ 
for  $i = 1$  to  $n$  do
    if  $\neg PushViolateConstraint(L, u_i, X)$ 
    then
         $found = true$ 
         $InsertAfter(u_1, v)$ 

```

```

        u.prev = Label(v)
        break
    fi
od
    (Cannot locate a child node that does not violate any constraint.
    Push all children up.)
    if found == false
        then
            PushUpChild(v, X)
        fi
        Delete(v)
    return X
end

```

**Algorithm 3.11: PushUpChild**

*Function PushUpChild(v, X) : X*

**begin**

**if**  $\neg \text{IsLeaf}(v)$

**then**

$\langle u_1 \cdots u_m \rangle = \text{Children}(v)$

*refNode = v*

**for**  $i = 1$  **to**  $m$  **do**

*InsertAfter( $u_i, v$ )*



```
         $u_i.prev = Label(v)$ 
         $refNode = u_i$ 
    od
     $Delete(v)$ 
fi
return X
end
```

### 3.3 Evaluation

In this section, we present an empirical study and some analytical results about the Document Converter.

#### 3.3.1 Computational Complexity

We give an analysis of the computational complexity of the function *DocumentConversion*. We assume a unit cost model of the operations on an XML document tree. Since **punc** (punctuation delimiters) is bound by the total number of characters, the set is usually small in size. We can hash the punctuation delimiters so that checking if a character is a punctuation delimiter can be done in constant time. Similarly, **group** and **list** (group and list tags) are bound by the total number of

HTML tags, checking if a label of a node is a group or list tag can be done in constant time. Let  $svc$  and  $pvc$  denote the time for each call to *SinkViolateConstraint* and *PushViolateConstraint*, respectively.

**Tokenization:** Consider the function *Tokenization*. For each node  $v$ , it scans the characters in  $v.val$  once. The function is recursively applied to the nodes in  $X$  in a top-down manner. Only the tree structure of text nodes is modified by appending new "TOKEN" nodes. But *Tokenization* is not recursively applied to these text nodes. Hence, *Tokenization* is applied to each node once. The time complexity thus is  $O(\sum_{v \in X} |v.val|)$  which is the sum of the lengths of the attributes  $val$  of all nodes in  $X$ , i.e. the size of the text nodes in the document.

**ConceptIdentification:** Consider the function *ConceptIdentification*. For each  $v$ , concepts are identified from  $v.val$ . A new node is created for each concept and is appended as child to  $v$ . Let  $ci(text)$  denote the time of identifying concepts from  $text$  (either by *ConceptInstanceByKeywords* or *ConceptInstanceByBayes*). This takes  $O(ci(v.val))$  time. Using a similar argument as in *Tokenization*, *ConceptIdentification* is applied to each node in  $X$  once. Hence, the time is bound by  $O(\sum_{v \in X} ci(v.val))$ .

**Grouping:** For each node  $v$ , the function *Grouping* scans its child nodes. Nothing is done to a child node unless it is a right sibling node of some node labeled with a group tag. In such a case, it is pushed down in the tree. Concept constraints are checked for concept nodes to make sure there are no violations.

The complication here is that the input document tree  $X$  increases in size and in depth as new **GROUP** nodes are created and sibling nodes of  $v$  are pushed down the tree. Hence, after a concept node is pushed down in the tree, it may be checked for concept constraint violations again in the next recursion. Similarly, a non-concept node may be matched for a group tag or pushed down the tree more than once.

Let  $X'$  denote the resulting document tree after applying *Grouping* to  $X$ . The following relationship can be observed between  $X$  and  $X'$

$$(1) \quad Deg(X') \leq Deg(X)$$

$$(2) \quad Depth(X') \leq Deg(X)Depth(X)$$

$$(3) \quad |X'| \leq 2|X|$$

The degree of  $X'$  cannot be greater than that of  $X$  because at each recursion, only a subset of child nodes of  $v$  are pushed down which results in a decrease in fan-out. When child nodes of  $v$  are pushed down the tree, the tree increases in depth by 2. However, this can only occur when there are at least 2 matching child nodes of  $v$  with the same group tag. At each recursion, there are at most  $Deg(v)/2$  such pairs, which leads to at most  $(Deg(v)/2)*2=Deg(v)$  increase in depth in  $X'$ . The size of  $X'$  can increase by at most a factor of 2 in the case that each child node of  $v$  leads to a creation of a new node labeled with **GROUP**.

Let us consider the cost of checking constraint violations. We use the following notations:

- $l$ : the number of times the child nodes of the node  $v$  are pushed down two levels (one for the GROUP node, the other for the sibling nodes) by subsequent recursions in *Grouping*
- $g$ : the number of child nodes of  $v$  which are not concept nodes
- $n$ : the number of child nodes of  $v$  which are concept nodes
- $d$ : the total number of child nodes of  $v$ , i.e.  $d = g + n$

At each iteration, we push some child nodes of  $v$  down the tree only if we find at least a pair of child nodes labeled with the same group tag. There are at most  $\frac{g}{2}$  such pairs. Thus, we have

$$l \leq \frac{g}{2}$$

Constraint violations are only checked for concept nodes. No new concept nodes are created by *Grouping*. At each level, at most  $n$  nodes are checked for constraint violations. Since there are  $l$  levels, the number of times *SinkViolateConstraints* is called is bound by

$$ln \leq \frac{gn}{2} \leq \frac{g(d-g)}{2}$$

Consider the function  $f(g) = \frac{g(d-g)}{2}$ . Differentiating  $f(g)$  gives us a local maximum of  $f(g)$  at  $g_0 = \frac{d}{2}$  with  $f(g_0) = \frac{d^2}{8}$  which is the maximum number of times *SinkViolateConstraints* is called at each node  $v$ . The cost of calling *SinkViolateConstraint* for the whole tree is thus bound by

$$s_{vc} | X' | \frac{Deg(X')^2}{8} \leq 2s_{vc} | X | \frac{Deg(X)^2}{8}$$

A child node of  $v$  that is not a node labeled with the chosen group tag may be matched against the group tags and pushed down in the tree. All these are unit cost operations. Applying a similar analysis, the total cost of such operations for all these nodes is  $O(Deg(X)^2 | X |)$ . The total time for *Grouping* is thus  $O(svc Deg(X)^2 | X |)$ .

**Consolidating:** For each node  $v$ , the function *Consolidating* only restructures it if  $v$  is an internal non-concept node. In Rule 1,  $v$  is replaced by all its child nodes, which takes  $O(Deg(v))$  time. Rule 2 and 3 are similar to *Grouping* which takes  $O(svc Deg(v))$  time. In Rule 4, its child nodes are scanned and it is replaced by the first child node that does not violate any constraint. The time is thus  $O(pvc Deg(v))$ . *Consolidating* is applied to each node in  $X'$  once. The total time is thus  $O(Max\{svc, pvc\}Deg(X') | X' |)$  which in turn is bound by  $O(Max\{svc, pvc\}Deg(X) | X |)$ .

**Constraint Violations:** Consider the function *SinkViolateConstraint*. This requires evaluating each predicate in  $L$ . If the predicate is of type *MaxDepth* or *MinDepth*, constraints can be checked in constant time. If the predicate is of type *NonSiblings*, constraints can be checked in  $Deg(X)$  time. If the predicate is of type *NonAncestorDescendant*, constraints can be checked in  $Depth(X)$  time. This gives the upper bound  $O(| C | (Deg(X) + Depth(X)))$ .

Consider the function *PushViolateConstraint*. This is similar to *SinkViolateConstraint*. The difference is in checking *NonAncestorDescendant*. Now we need to check all the descendants of *node* within  $k$  levels where  $k$  is the max-

imum level in *NonAncestorDescendant* predicates in  $L$ . This is bound by  $Deg(X)^k$ . The time complexity is bound by  $O(|C| (Deg(X) + Depth(X)^k))$ .

Summing up, we have the following time complexity of the whole document conversion process:

**Theorem 3.1 (Time Complexity of Document Conversion).** *Given a document tree  $X$  and a set of concept constraints  $L$ . Let  $k$  denote the maximum level specified in *NonAncestorDescendant*( $a, d, k$ ) in  $L$ . Let  $ci(text)$  denote the time it takes to call the function *ConceptInstance* for concept identification. The time complexity of calling *DocumentConversion* on  $X$  is bound by*

$$O(\sum_{v \in X} ci(v.val) + |C| (Deg(X)^2 + Depth(X)^k) Deg(X) |X|)$$

*If no concept constraint is specified, the time complexity is reduced to*

$$O(\sum_{v \in X} ci(v.val) + Deg(X)^2 |X|)$$

*which is linear to the size of the document.* □

### 3.3.2 Empirical Study

We conducted empirical experiments on the efficiency and effectiveness of the Document Converter on résumés documents gathered using IBM's Grand Central Station web crawler focused on crawling for résumés ([IBM97]). This crawler was programmed to crawl the Web looking for HTML documents that looked like résumés. We ran our

experiments on a Pentium 266MHz processor with 196MB main memory and 512KB cache.

We consider 24 concepts. Concept instances are based on a total of 233 keywords. Punctuation delimiters for tokenization are ",", ":", ";". Group tags for the Grouping Rule are `h1,h2,h3,h4,h5,h6, div,p,tr,dt,dd, li,title,u, strong,b, em, i`. List tags for the Consolidating Rule were `body,table,dl, ul,ol, dir,menu`. No concept constraints were specified.

### 3.3.2.1 Efficiency

We ran the algorithm on 100 HTML documents to evaluate the average running time on a single document. The average number of nodes of a HTML document in the dataset is 203 of which 52 are concept nodes. The average file size is 11.7 KB. The average running time was found to be 0.53 seconds.

### 3.3.2.2 Scalability

We measure the scalability of the data extraction process with respect to the sizes of the HTML documents and the number of concept nodes. Results are shown in Figures 3.1 and 3.2. The numbers demonstrate that the running time scales linearly with the document sizes and the number of concept nodes.

Let us compare it with the analytical result of the time complexity in the case that there are no concept constraints:  $O(\sum_{v \in X} ci(v.val) + Deg(X)^2 |X|)$ . The document sizes give an estimate on the first component in the formula. The number of concept nodes partially describes the second component. The result of the empirical study demonstrates that, in practice, the number of fan-outs of the document tree is not significant. This is because, in practice, fan-outs of documents are typically not very large and they can be considered as a constant.

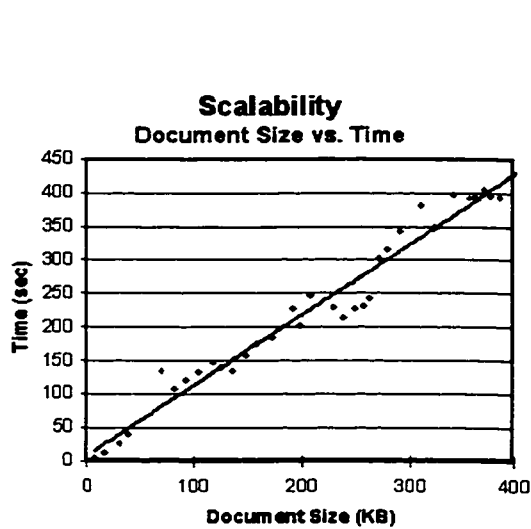


Figure 3.1: Scalability - Document Size vs. Time

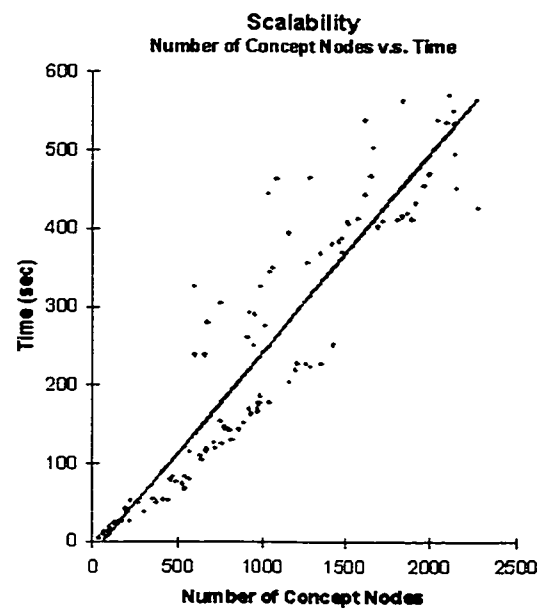


Figure 3.2: Scalability - Number of Concept Nodes vs. Time



### 3.3.3 Accuracy

In order to evaluate the accuracy of the Document Converter, we count the number of incorrect parent-child and sibling relationships in the converted XML tree by manually inspecting it against the author's perception on how the information content of the document should be according to its visual layout. We reorder the nodes in the tree in order to convert it to a tree matching the semantics of the original document. In doing so, we may move a node and its siblings together to make up for one parent-child relationship incorrectly identified. This is counted as one logical error since it corresponds to the same parent-child relationship.

The Document Converter takes 50 résumé documents. The XML documents computed by the Document Converter are compared to the logical layout of the information content of the résumés based on the author's perception. The XML documents are restructured to match their logical layout. The number of logical errors gives the number of errors in the document conversion process.

The result is shown in Figure 3.3. The average number of errors in each HTML document is 3.9. The average number of concept nodes in a HTML document is 53.7. The average percentage of error nodes in a HTML document with respect to the total number of concept nodes is 9.2%. In other words, our heuristics has an accuracy of 90.8%.

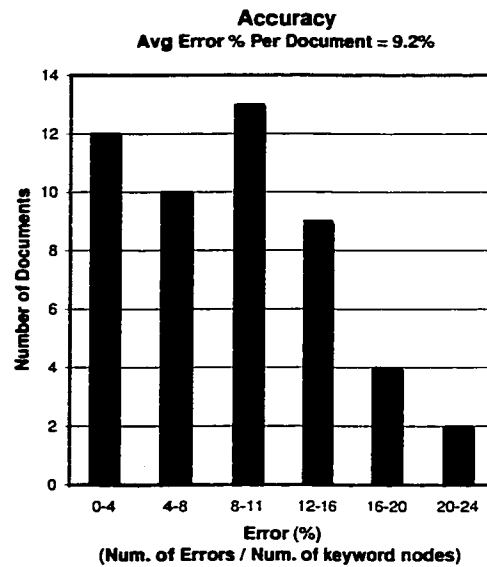


Figure 3.3: Accuracy - Percentage of Errors

## 3.4 Related Work

In this section, we first give an overview of related work in the area of extracting information from documents. Then, we compare the Document Converter with these approaches.

### 3.4.1 Classification

Our work is related to middleware systems which consist of heterogeneous distributed data sources. Each data source may have its own schema in its own data model. A

wrapper for a data source translates its data into a common data model. Interaction with the data sources is transparent to users who only have a global view of the data sources. A mediator takes user queries in the global view, decomposes them into subqueries to relevant wrappers, and merges the subquery results from the wrappers before presenting them to users. Common middleware systems include Information Manifold [LRO96], TSIMMIS [HGMI<sup>+</sup>95], Garlic [CHS<sup>+</sup>95], Ozone [LAW99] and MIX [PV99].

Topic specific HTML documents can be viewed as local data sources with their own schema. Translating the HTML documents into XML documents marked up in homogeneous topic specific XML tags is the "wrapping" phase.

There are quite a number of wrappers methods proposed in research literature that extract data from HTML documents ([PGMU96, HGMC<sup>+</sup>97, SA99, CDSS98, Ade98, AK97, DYKR00, ECJ<sup>+</sup>98, Kus00]).

These wrappers can be classified according to several criteria. One dimension is the degree of automation of wrapper generation. Manual wrappers require users to specify how to extract data from HTML documents. Their goal is to provide a language to facilitate the specification process. These include MedMaker [PGMU96], [HGMC<sup>+</sup>97], Editor [AM97], W4F [SA99] and YAT [CDSS98]. Some wrappers can be generated automatically, such as [Ade98, ECJ<sup>+</sup>98, Kus00, DYKR00]. Users give examples on the data extraction process or specify the structure of the data sources, and then the system learns the extraction rules.

Manual wrappers can be compared by the expressiveness of their wrapping language. [HGMC<sup>+</sup>97] uses regular expressions for pattern matching to specify extraction rules. In Editor [AM97], extraction rules are implemented as a procedural program using operators that can match strings in the document using regular expressions. W4F [SA99] allows path expressions in its wrapper language to facilitate navigation in HTML trees. MedMaker proposes a declarative Datalog-like rule-based language on OEM data [NUWC97] which can consider the tree structures of HTML documents. Similar to MedMaker, YAT proposes a declarative rule-based language on SGML data, with additional support for collections, order and grouping.

Automatic wrappers can be compared by the domain knowledge they consider. All of these wrappers assume that the data sources are relatively homogeneous, i.e. all the records in an HTML document or all HTML documents follow a particular structure. NoDoSE [Ade98] requires users to specify a schema for the records in form of templates and to give examples on how to extract data to fill these templates. The system can learn the extraction rules based on the examples. [AK97] uses heuristics to locate records based on the format and tree structures of HTML documents. [ECJ<sup>+</sup>98] assumes there is an ontology for the application which gives a schema for a database and specifies how entities in the ontology can be identified. They populate the database according to the ontology. Heuristics (based on HTML tags, fan-out of nodes in HTML trees, record sizes, repeating markup sequences and ontology) are used to locate record boundaries in HTML documents. [Kus00] assumes the records are in a form described by 6 wrapper classes. It learns how to extract records for each

type of wrapper class. From the examples given by users, [DYKR00] learns regular expressions that can extract data correctly from documents of similar format.

### 3.4.2 Comparison

The Document Converter is an automatic wrapper employing a wide spectrum of domain-independent heuristics with an explicit, optional mechanism to capture domain knowledge.

The Document Converter offers several advantages over manual wrappers [HGMC<sup>+</sup>97, SA99, PGMU96, CDSS98]. The problems of manual wrappers are: (1) They assume the data are in a known format (or a couple of variants of the same format). They are applicable for data generated by the same data source. Data gathered from the Web, however, are too diverse to follow the same or even similar format. (2) They cannot handle dynamics of the data. The format of the data from a given source may change over time. Every change of format would require a new handcrafted wrapper requiring too much human resources to generate. Unlike these manual wrappers, the data extraction process in the Document Converter is independent of the actual format of the data. The heuristics of visual clues used by the Document Converter are general enough to be applicable to diverse sources of data. The samples required to train the Bayes classifier as to how to associate text with label are domain-specific. Such information is independent of the data sources and is relatively stable. For instance, there are only 12 months in a year for the label DATE.

Automatic wrappers ([Ade98, AK97, Kus00, DYKR00]) assume that records in a document or the documents follow the same structure. NoDoSE requires users to specify this structure as a class template. [Kus00] supplies six system-defined classes of wrappers. The Document Converter does not assume that the records in different documents following the same structure. It only assumes that records within a document follow some regular pattern. This is a reasonable assumption because usually there is only one person authoring an HTML document and the information content is presented in a consistent way.

A design issue with automatic wrappers is the balance between being independent of the domain and the capability to capture domain knowledge. To strike a balance between these two, the Document Converter uses the strategy of incorporating a broad-spectrum of domain-independent heuristics, and provides an explicit mechanism for users to specify domain knowledge. Domain knowledge is optional but not required.

The Document Converter is selective in its choice of heuristics so that they are general enough to be applicable to a wide range of applications. It considers the format of the HTML documents, the format clues of HTML markup tags and the tree structure of the documents. All of these are relatively domain-independent. Although the heuristics are not new to the literature, we are not aware of any system that has incorporated all these domain-independent features.

Unlike [ECJ<sup>+</sup>98] which requires a known ontology of an application for the extraction process, domain knowledge is optional but not required for the Document Converter. It provides an explicit constraint mechanism to allow users to specify domain knowledge. The constraint mechanism is simple, flexible and easily extensible. We do not adopt an elaborate mechanism (such as ontology or natural language parser) to specify domain knowledge because data from diverse sources are too heterogeneous to bear complicated relationships. Even if they do, it is unlikely that users are aware of such relationships.

The quality of the converted XML documents depends on the number of concepts specified by the user and the examples or keywords associated with the concepts. If the Bayes classifier is used to recognize concepts, it is important that there are enough training data to train the Bayes classifier.

Reference	Input	Output	Automatic?	Wrapper Language/Domain Knowledge
[HGMC <sup>+</sup> 97]	HTML	OEM	N	regular expression
[AM97]	HTML	HTML	N	regular expression
[SA99]	HTML	Nested String List, XML	N	path expression
[CDSS98]	HTML	HTML	N	declarative rule-based language on SGML data
	/ODMG	/ODMG		
[PGMU96]	OEM data	OEM data	N	declarative rule-based language on OEM data
[Ade98]	text	hierarchy of class instances	Y	class template definition
[AK97]	HTML	tree-structure spec	Y	lang. in reg. exp. & procedures considers HTML tree & format
[ECJ <sup>+</sup> 98]	text	database instance	Y	ontology, HTML tags, fan-out of nodes, record sizes
[Kus00]	HTML	(label, value) pairs	Y	6 wrapper classes
[DYKR00]	HTML	regular expressions	Y	none
Quixote	HTML	XML	Y	document format, HTML tags document tree structure, constraints

Figure 3.4: Comparison of Wrapper Methods

## Chapter 4

# Schema Mining

Chapter 3 described how the Document Converter converts a collection of topic specific HTML documents into XML documents using topic specific tags. This chapter describes how a global schema for this collection of XML documents is discovered. We propose the notion of *majority schema* which describes prevalent structures among the documents as their global schema [CS00]. Majority schemas give users unfamiliar with the documents a bird's eye view of the documents. They can be used to optimize storage. Nodes in a document that are close in proximity in the majority schema are likely to be accessed together and thus can be stored together. They can be used for indexing. Nodes in the majority schema with the high support are most commonly found and can be indexed for more efficient retrieval.



In Section 4.1, we motivate the notion of majority schema. The formal definition of a majority schema and the schema discovery problem is then given. In Section 4.2, we describe how the Schema Miner discovers a majority schema for a collection of topic specific XML documents. By using the simple formalism of a tree schema, an initial majority schema is discovered efficiently. This initial majority schema is then further refined to derive its corresponding DTD. In Section 4.3, we present analytical and empirical results on the efficiency and feasibility of the Schema Miner. In Section 4.4, we describe related work and compare the Schema Miner to these approaches.

## 4.1 Problem Formulation

In Section 4.1.1, we introduce the notion of *majority schema* and motivate the use of a tree schema as its formalism. In Section 4.1.2, we give a formal definition of a majority schema and the schema discovery problem.

### 4.1.1 Majority Schema

Due to the heterogeneity among a collection of XML documents, an exact schema for the documents, i.e. a schema that describes all structures found in the documents and these structures only, may be too big in size. One way to reduce the size is to have an approximate schema which makes more general statements on the document

structures, i.e. it may cover structures not found in any of the documents or make no statement on structures of the elements if there are too many variants. This kind of approximate schema may convey little information for a heterogeneous collection of documents.

In view of this, we propose the notion of *majority schema* which describes only prevalent structures in the documents. Unlike an exact schema, a majority schema is small in size. It is an approximate schema which only covers prevalent structures in the documents. It covers the portion of the structures that are prevalent and hence representative for the collection of documents.

Since we are interested in the structures of the documents, we take the document-centric view on XML documents as opposed to the data-centric view (see Infoset versus the OEM data model in Section 2.1.1). Therefore, we choose DTD as the formalism for the majority schema. Since a majority schema describes only prevalent structures among the documents, we can first infer an initial majority schema with certain information deliberately ignored. Missing details are filled in later to convert the majority schema to a DTD. Hence, we can use a tree schema which is less expressive than a DTD as the formalism for the majority schema.

### 4.1.2 Problem Definition

We assume a set of element names (labels)  $E$ , a set of attribute names  $A$ , and a set of string values  $S$  which is the type of all atomic values.

The input is a collection of XML documents  $\mathcal{X} = \{X_1, \dots, X_n\}$  pertaining to a specific topic, i.e.  $\forall X_i, X_j \in \mathcal{X}: (Label(X_i.root) = Label(X_j.root))$ .

Similar to [WL99, WL98, NAM97], we use the occurrences, also called *support*, of a label path (Definition 2.4) among the documents to measure how prevalent it is. To avoid bias towards large XML documents and to normalize the measure for comparison, we refine the definition of support by the ratio of documents containing the label path.

**Definition 4.1 (Support).** *Given an XML document  $X$  and a label path  $p$ . The support of  $p$  in  $X$  is*

$$Support(p, X) := \frac{|\{v \mid v \in X \wedge LabelPath(v) = p\}|}{|\{v \mid v \in X\}|} \in \mathbf{R}[0, 1]$$

*Given a collection of XML documents  $\mathcal{X}$  and a label path  $p$ . The support of  $p$  in  $\mathcal{X}$  is*

$$Support(p, \mathcal{X}) := \frac{|\{X \mid p \in X, X \in \mathcal{X}\}|}{|\{X \mid X \in \mathcal{X}\}|} \in \mathbf{R}[0, 1]$$

*We may use the shorthand  $Support(p)$  if the collection of documents is clear from the context.* □

The higher the support, the more common the label path is in the collection of documents. It is used to select prevalent structures from the documents.

Since the support of a label path typically decreases with its length (i.e. if  $p$  is a subpath of  $q$ ,  $Support(p, \mathcal{X}) \geq Support(q, \mathcal{X})$ ), a more practical measure would be the ratio of the support of the label path and its 1-subpath, called its *support ratio*.

**Definition 4.2 (Support Ratio).** *Given a collection of XML documents  $\mathcal{X}$  and a label path  $p$ . The support ratio of  $p$  is defined as follows:*

$$SupportRatio(p, \mathcal{X}) := \begin{cases} \frac{Support(p, \mathcal{X})}{Support(Subpath(p,1), \mathcal{X})} & \text{if } |p| \geq 2 \\ 1 & \text{if } |p| = 1 \\ \perp & \text{otherwise} \end{cases}$$

□

A label path with high enough support and support ratio is called a *frequent label path*.

**Definition 4.3 (Frequent Label Path, Maximal Frequent Label Path).** *Given a collection of XML documents  $\mathcal{X}$ , and parameters  $supThreshold, ratioThreshold \in \mathbf{R}[0, 1]$ . A label path  $p$  is a frequent label path in  $\mathcal{X}$  at  $[supThreshold, ratioThreshold]$  iff the followings hold:*

- $p$  is a valid label path in  $\mathcal{X}$ , i.e.  $p \in \mathcal{X}$
- $p$  has high enough support:

$$\text{Support}(p, \mathcal{X}) > \text{supThreshold}$$

- All subpaths of  $p$  have a high enough support ratio:

$$\forall k \in \mathbf{Z}: (q = \text{Subpath}(p, k) \wedge \text{supportRatio}(q, \mathcal{X}) > \text{ratioThreshold})$$

Let  $\text{FreqLP}(\mathcal{X}, \text{supThreshold}, \text{ratioThreshold})$  denote the set of frequent label paths for  $\mathcal{X}$  at  $\text{supThreshold}$ ,  $\text{ratioThreshold}$ , or  $\text{FreqLP}(\mathcal{X})$  when the context for  $\text{supThreshold}$ ,  $\text{ratioThreshold}$  is clear. A frequent label path  $p$  is a maximal frequent label path in  $\mathcal{X}$  if there are no other frequent label paths in  $\mathcal{X}$  which are superpaths of  $p$ . The set  $\mathcal{P}$  of maximal frequent label paths for  $\mathcal{X}$  at  $[\text{supThreshold}, \text{ratioThreshold}]$  consists of all maximal frequent label paths, i.e.,

$$\mathcal{P} = \{p \mid (p \in \text{FreqLP}(\mathcal{X}, \text{supThreshold}, \text{ratioThreshold}) \wedge (\neg \exists q \in \mathcal{X}, k \in \mathbf{Z}^+ : (p = \text{Subpath}(q, k) \wedge q \in \text{FreqLP}(\mathcal{X}))))\}$$

□

The higher the values for parameters  $\text{supThreshold}$  and  $\text{ratioThreshold}$ , the more selective is the criteria of determining a path being a frequent label path. Empirical studies described in Section 4.3.2.2 show that  $\text{supThreshold} \geq 0.6$ ,  $\text{ratioThreshold} \geq 0.2$  are reasonable choices. The user can first take a small sample of the collection of documents to determine suitable values for these parameters.

Intuitively, a majority schema is a schema that represents the structures shared by the majority of the documents which are their frequent label paths. We represent a majority schema as a tree schema. A tree schema is an unordered tree with a one-to-one correspondence to a set of label paths. Thus, a majority schema is the tree schema built from the set of maximal frequent label paths of the documents. A tree schema can easily be converted to a DTD (Section 2.3.3).

**Definition 4.4 (Majority Schema).** *Given a collection of XML documents  $\mathcal{X}$  and parameters  $supThreshold$ ,  $ratioThreshold$ . Let  $\mathcal{P}$  denote the set of maximal frequent label paths of  $\mathcal{X}$  at  $[supThreshold, ratioThreshold]$ . A majority schema  $S$  for  $\mathcal{X}$  at  $[supThreshold, ratioThreshold]$  is a tree schema  $S$  built from  $\mathcal{P}$*

$$S = Paths2Tree(\mathcal{P})$$

□

The parameters  $supThreshold$  and  $ratioThreshold$  are used to adjust the degree of precision of the majority schema. In particular,  $supThreshold = 0$  and  $ratioThreshold = 0$  give a majority schema that encompasses structures found in *any* document in  $\mathcal{X}$ , i.e. a schema with maximum degree of precision.

**Schema Discovery Problem:**

Given a collection of XML documents  $\mathcal{X}$  and parameters  $supThreshold$ ,  $ratioThreshold \in \mathbf{R}[0, 1]$ . The schema discovery problem

$SDP[\mathcal{X}, supThreshold, ratioThreshold]$  is to compute the majority schema  $S$  of  $\mathcal{X}$  at  $[supThreshold, ratioThreshold]$  and its DTD representation.

## 4.2 Approach

In this section, we describe the approach taken by the Schema Miner. The schema mining process is guided by the intuition that imprecise data modeling reveals prevalent structures in the underlying data. We deliberately ignore certain information in the discovery process. After we infer an initial majority schema, we fill in the missing details to convert the majority schema to a DTD.

Section 4.2.1 describes the basic algorithm to compute an initial majority schema. Section 4.2.2 describes how domain knowledge (Section 3.1) can be utilized to optimize the discovery process. Since there may be a couple of typical ways of marking up the documents, the majority schema may contain structures that are slightly different but actually describe similar information content. Section 4.2.3 describes a method to homogenize the majority schema, if necessary. Finally, Section 4.2.4 describes how the content model of the elements in the DTD derived from the majority schema can be discovered.

### 4.2.1 Computing Frequent Label Paths

In this section, we give the basic algorithm to compute an initial majority schema for a collection of documents  $\mathcal{X}$ . The set of maximal frequent label paths for the documents can be obtained by exhaustively enumerating all label paths, as described

by the function *BasicCFP*<sup>1</sup>. In *BasicCFP*, all label paths of increasing lengths up to the length of the longest label path in  $\mathcal{X}$  are enumerated. The search space thus is

$$\mathcal{P} := \{p \mid p \in \mathcal{X}\}$$

It considers label paths in  $\mathcal{X}$  of increasing length by extending a label path  $p$  of length  $d$  in  $\mathcal{P}$  to  $q$ .

Similar to the Apriori algorithm in computing frequent itemsets [AS94], a label path cannot be a frequent label path if one of its subpath is not a frequent label path. If the support of a label path is below *supThreshold*, so would be the support of any of its superpaths. Any subpath of a frequent label path has to have a high enough support ratio. Therefore, in the **for** loops in *BasicCFP*, a label path  $q$  is added to  $\mathcal{P}$  only if it has high enough support. Since we only want maximal label paths, if a label path  $q$  is added to  $\mathcal{P}$ , its subpaths cannot be maximal and are pruned from  $\mathcal{P}$ .

Albeit the function *BasicCFP* is presented in a procedural language, query languages on XML data such as [CRF00] can be used to implement the function.

**Algorithm 4.12: BasicCFP**

*Function BasicCFP*( $\mathcal{X}$ , *supThreshold*, *ratioThreshold*) :  $\mathbf{X}$

**begin**

$$\mathcal{P} = \{x.root \mid x \in \mathcal{X}\}$$

$$maxPathLength = Max\{|p| \mid p \in \mathcal{X}\}$$

---

<sup>1</sup>It stands for Basic Compute Frequent Path



```

for  $d = 1$  to  $maxPathLength$  do
  for each  $p \in \mathcal{P} \wedge |p| = d$  do
    (Extend  $p$  by one label  $e$ )
     $extend = false$ 
    for each  $q = p \circ e \in \mathcal{X}$  do
      (Apriori condition: Only add a frequent label path)
      if  $Support(q, \mathcal{X}) > supThreshold \wedge$ 
         $SupportRatio(q, \mathcal{X}) > ratioThreshold$ 
        then
           $\mathcal{P} = \mathcal{P} \cup \{q\}$ 
           $extend = true$ 
        fi
      od
    if  $extend = true$ 
      then
         $\mathcal{P} = \mathcal{P} - \{p\}$ 
      fi
    od
  od
return  $Paths2Tree(\mathcal{P})$ 
end

```

### 4.2.2 Using Domain Knowledge

If users specify domain knowledge (Section 3.1) on the structures of concepts in a topic, the schema discovery process can be improved by utilizing such information to improve the quality of the schema discovered. To reiterate, domain knowledge is specified in a first order predicate logic language with the following predicates to constrain permissible structures in a tree schema:

- $MaxDepth(e, d)$  is true iff the depth of the label  $e$  cannot be larger than  $d$ .
- $MinDepth(e, d)$  is true iff the depth of the label  $e$  cannot be smaller than  $d$ .
- $NonAncestorDescendant(e_1, e_2, level)$  is true if the label  $e_2$  cannot be a descendant of the label  $e_1$  in a schema within  $level$  levels. In particular, if  $level = \infty$ ,  $e_2$  cannot be a descendant of  $e_1$ . If  $level = 1$ ,  $e_2$  cannot be an immediate child of  $e_1$ .
- $NonSiblings(e_1, e_2)$  is true if the labels  $e_1, e_2$  cannot be siblings in a tree schema.

The function  $BasicCFP$  is refined to  $DomKnowCFP$  to take into consideration the domain knowledge  $L$  which is a set of predicates. The function  $DomKnowCFP$  is similar to  $BasicCFP$  with additional procedures to check for violation of constraints in  $L$ . If a label path violates some constraint, it is not added to  $\mathcal{P}$ . In other words, if the candidate path  $q$  violates a constraint,  $DomKnowCFP$  continues the next iteration to look for another candidate path. The order of extending a label path

by labels is nondeterministic. Such an order has an impact on the majority schema discovered if there is a violation of sibling constraints. If two candidate label paths violate a sibling constraint, the first label path considered would be added to  $\mathcal{P}$ .

**Algorithm 4.13: DomKnowCFP**

*Function DomKnowCFP( $\mathcal{X}$ , supThreshold, ratioThreshold, L) :  $\mathbf{X}$*

**begin**

$\mathcal{P} = \{\mathbf{x}.root \mid \mathbf{x} \in \mathcal{X}\}$

$maxPathLength = Max\{|p| \mid p \in \mathbf{x} \in \mathcal{X}\}$

**for**  $d = 1$  **to**  $maxPathLength$  **do**

**for each**  $p = p_1 \circ \dots \circ p_d \in \mathcal{P}$  **do**

$extend = false$

(Extend  $p$  by one label  $e$ )

**for each**  $q = p \circ e \in \mathcal{X}$  **do**

(Check if  $e$  satisfies MaxDepth and MinDepth constraints)

**if**  $\exists md \in \mathbf{Z}: (MaxDepth(e, md) \wedge d > md)$

**then**

**continue**

**elseif**  $\exists md \in \mathbf{Z}: (MinDepth(e, md) \wedge d < md)$

**continue**

**fi**

(Check if  $q$  satisfies NonAncestorDescendant constraints)

**for**  $i = 1$  **to**  $d$  **do**

```

    if  $\exists level \in \mathbf{Z}: (\neg NonAncestorDescendant(p_i, e, level) \wedge$ 
       $d - i \leq level)$ 
      then
        continue
      fi
    od
    (Check if q satisfies NonSiblings constraints)
    for each  $p \circ e' \in \mathcal{P}$  do
      if  $\neg NonSiblings(e', e)$ 
        then
          continue
        fi
      od
      (Apriori condition: Only add a frequent label path)
      if  $Support(q, \mathcal{X}) > supThreshold$ 
         $\wedge SupportRatio(q, \mathcal{X}) > ratioThreshold$ 
          then
             $\mathcal{P} = \mathcal{P} \cup \{q\}$ 
             $extend = true$ 
          fi
        od
      if  $extend = true$ 
        then

```

```

         $\mathcal{P} = \mathcal{P} - \{p\}$ 
      fi
    od
  od
  return Paths2Tree( $\mathcal{P}$ )
end

```

### 4.2.3 Unification

A majority schema may contain of similar subtree structures. This arises because there may be a couple of typical ways to markup a document pertaining to a specific topic. For example, the document fragments shown below describe the educational background in similar but exactly the same way. Since we are interested in a schema giving a concise uniform view over the documents, it is desirable to unify the heterogeneity. The user should investigate the majority schema to decide if unification is necessary. The user can inspect if there are similar sibling nodes in the majority schema starting from the root. Presence of such nodes is indicative of the need for unification.

```

Resume 1
<EDUCATION val="Educational Background">
  <DEGREE val="M.Sci.(Comp.Sci.)">
    <DATE val="1999"/>
    <ORGANIZATION val="UC Davis"/>
    <THESIS val="Semistructured Data"/>
  </DEGREE>
</EDUCATION>

```

Resume 2

```
<EDUCATION val="Academic Background">  
  <DEGREE val="B.Sci.(Comp.Sci.)">  
    <ORGANIZATION val="Stanford University"/>  
    <GPA val="3.7/4.0"/>  
    <DATE val="1997"/>  
  </DEGREE>  
</EDUCATION>
```

The approach to unify similar subtrees is based on the tree edit problem which has a known solution. We first describe the tree edit problem in Section 4.2.3.1. Section 4.2.3.2 discusses how similar subtrees in a majority schema can be identified by making use of the tree edit algorithm. Section 4.2.3.3 describes how two similar subtrees can be merged into one. Finally, the overall unification process is presented in Section 4.2.3.4 in which similar subtrees in the majority schema are continuously merged.

#### 4.2.3.1 Tree Edit Problem

The *tree edit* algorithm describes how to transform a source tree into a target tree with minimum cost of modifications. The source tree can be transformed into the target tree using three edit operators: (1) *Insert* operator: a vertex can be inserted into the source tree, (2) *Delete* operator: a vertex can be deleted from the source tree, (3) *Rename* operator: a vertex in the source tree can be mapped to another vertex in the target tree. If the vertex in the source tree has a different label than the one in the target tree, the vertex in the source tree is renamed. Each edit operator is

associated with a cost which is described by a cost model. An *edit sequence* describes the sequence of edit operations to transform the source tree into the target tree. The cost of the transformation is the sum of the cost of the edit operators in the edit sequence. The tree edit problem is to find the edit sequence with the minimum cost, called the *tree edit distance* of the two trees.

There are several properties of an edit sequence. (1) The root of the source tree is always mapped to the root of the target tree. (2) The mapping is one-to-one. (3) The mapping preserves ancestor-descendant relationships, i.e. if the vertex  $u$  is an ancestor of the vertex  $v$  in the source tree, the vertex to which  $u$  is mapped cannot be a descendant of the vertex to which  $v$  is mapped (if there is any) in the target tree.

The tree edit problem has been well-studied in the literature [Tai79, Lu79, Zha96]. Let  $T_1, T_2$  be the source and target tree respectively. [Tai79] presents an algorithm on ordered trees with time complexity

$$O(|T_1| |T_2| \text{Depth}(T_1)^2 \text{Depth}(T_2)^2)$$

[Lu79] improves the algorithm to

$$O(|T_1| |T_2| \min\{\text{Depth}(T_1), \text{Leaves}(T_2)\} \min\{\text{Depth}(T_2), \text{Leaves}(T_1)\})$$

where  $\text{Leaves}(T)$  denotes the number of leaf vertices in  $T$ . [Zha96] considers the tree edit problem on unordered trees. The complexity of the algorithm is

$$O(|T_1| |T_2| (\text{Degree}(T_1) + \text{Degree}(T_2)) \log(\text{Degree}(T_1) + \text{Degree}(T_2)))$$

We can apply one of these algorithms to solve the tree edit problem. We view a tree edit algorithm as a black box. The input to a tree edit algorithm are the source tree  $T_1$ , the target tree  $T_2$ , and the cost model  $C$  of the edit operators. The algorithm outputs the mapping of vertices in  $T_1$  to those in  $T_2$  (called *tree edit mapping*), denoted as  $\nu : T_1.V \rightarrow T_2.V$ .  $\nu(v) = u$  means the vertex  $v$  in the source tree  $T_1$  is mapped to the vertex  $u$  in the target tree  $T_2$ . If the label of  $v$  is different than that of  $u$ ,  $v$  is renamed to  $u$ . If  $u = \perp$ , it means  $v$  is deleted and hence is not mapped to any vertex in  $T_2$ . A vertex in  $T_2$  to which no vertex in  $T_1$  has been mapped is an inserted node.

The tree edit algorithm  $TreeEdit(T_1, T_2, C)$  transforms the source tree  $T_1$  into the target tree  $T_2$  under the cost model  $C$ . It computes a mapping  $\nu$  between nodes in  $T_1$  and  $T_2$  that gives the minimum cost of transformation. Let  $TreeEditDist(T_1, T_2)$  be a function computed by the algorithm that gives the total cost of transforming  $T_1$  to  $T_2$ .

Based on the mapping computed by a tree edit algorithm, we can define a "boundary box" of a node in the source tree (Figure 4.1). Since the root of the source tree  $T_1$  is always mapped to the root of the target tree  $T_2$ , if we walk from any vertex  $v$  in  $T_1$  to its root, we will find at least one vertex that is mapped to  $T_2$ . Let the one with the greatest depth be the *reference ancestor* of  $v$ <sup>2</sup>, denoted as  $RefAncestor(v, \nu)$  (or simply  $RefAncestor(v)$ , if the mapping is clear from the context). Consider one of the child nodes of  $v$ . As we traverse from  $v$  along the path from this child node to

---

<sup>2</sup>It is the nearest ancestor of  $v$  with a vertex in the target tree that is mapped (referenced) to it by the tree edit algorithm.



the leaf in  $T_1$ , we may find vertices that are mapped. If there is any of such vertices, we denote the one with the smallest depth the *reference descendant* of  $v$ . Since there may be more than one path from  $v$  down to the leaves of the tree,  $v$  may have more than one reference descendant or none at all. Let  $RefDescendants(v)$  denote the set of reference descendants of  $v$ . Similarly, let  $RefSiblings(v)$  denote the siblings of  $v$  that are mapped. The reference ancestor, siblings and descendants of the node  $v$  make up the boundary box of  $v$ .

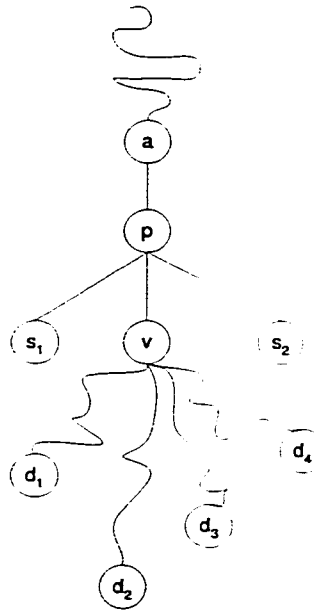


Figure 4.1: Boundary box of a node - The boundary box of the node  $v$  consists of its reference ancestor  $a$ , its reference siblings  $s_1, s_2$  and its reference descendants  $d_1, d_2, d_3, d_4$ .

### 4.2.3.2 Identify Similar Subtrees

To identify similar subtrees in a majority schema, we use the tree edit distance to define a quantitative measure, called *tree distance measure*, on the similarity between any two subtrees. The costs of the edit operators *Insert*, *Delete* are unit costs. Because of the Synonyms Property (Section 1.2.3), the cost of the *Replace* operator is infinite so that a vertex in the source tree can only be mapped to one with the same label in the target tree.

The tree distance measure between two trees is their tree edit distance, normalized by the sum of the sizes of the two trees (since one tree can always be obtained from the other by deleting all its nodes and inserting nodes of the other tree, which gives an upper bound on their edit distance).

**Definition 4.5 (Tree Distance Measure).** *Given two XML documents or document fragments  $X_1, X_2$  and a user-defined parameter  $distThreshold \in \mathbf{R}[0, 1]$ . The tree distance measure between  $X_1, X_2$  is*

$$TreeDist(X_1, X_2) := \frac{TreeEditDist(X_1, X_2)}{|X_1.V| + |X_2.V|} \in \mathbf{R}[0, 1]$$

where *TreeEditDist* is the tree edit distance to transform  $X_1$  to  $X_2$  with unit cost for *Insert* and *Delete* operators and infinite cost for the *Replace* operator.

$X_1, X_2$  are structurally similar at  $distThreshold$ , denoted as

*Similar*( $X_1, X_2, distThreshold$ ) if  $TreeDist(X_1, X_2) \leq distThreshold$ . □

In computing the tree edit distance, a technical nuance is that the root of one tree is always mapped to the root of the other. Since the roots may have different labels, it may not be desirable to map one to the other. This can be fixed by adding artificial roots of the same label to both trees before feeding them to a tree edit algorithm.

The tree distance measure lies within  $\mathbf{R}[0, 1]$ . The lower the tree distance measure, the more similar the two trees. The smaller the parameter *distThreshold*, the more selective is the criteria to determine if two trees are structurally similar. In particular, if *distThreshold* = 0, two trees are only structurally similar if they are exactly the same.

#### 4.2.3.3 Merging Similar Subtrees

If two subtrees in a majority schema are structurally similar, they need to be merged into one. Since we are interested in capturing one typical way of marking up the document in the schema, it is tempting to just pick one of the subtrees and discard the other. However, a concept may be missing from one way and is present in the other. It is desirable that we merge these structurally similar subtrees by superimposing them, i.e. concept nodes from both subtrees appear in the merged tree.

This is implemented by the function *Superimpose*( $X_1, X_2, \nu$ ). It takes two XML fragments  $X_1, X_2$  and the mapping from  $X_1$  to  $X_2$ , denoted as  $\nu$ , given by the tree edit distance algorithm. It uses  $X_2$  as the template for the merged tree. A vertex

$v \in X_1$  that is not mapped to any vertex in  $X_2$  (missing concept) should be added to the merged tree. Since the parent of  $v$  may not be mapped to any vertex in  $X_2$  either, we need to locate its reference ancestor,  $v_0$ , and append the whole path from  $v_0$  to  $v$  to the merged tree.

**Algorithm 4.14: Superimpose**

*Function Superimpose( $X_1, X_2, \nu$ ) :  $\mathbf{X}$*

**begin**

*(Add vertices of  $X_1$  to  $X_2$ )*

**for each**  $v \in X_1.V$  **do**

*(Check if  $v$  is mapped to any vertex in  $X_2$ )*

**if**  $\mu(v) = \perp$

**then**

*(Add vertices along the path from its reference ancestor to  $X_2$ )*

$v_0 = \text{RefAncestor}(v, \mu)$

$p = w \oplus \langle v_0, \dots, v_k \rangle \in \mathbf{X}, v_k = v$

**for**  $i = 1$  **to**  $k$  **do**

**if**  $\mu(v_i) = \perp$

**then**

$u_i = \text{Create}(X_2, \text{Label}(v_i))$

$u_{i-1} = \mu(v_{i-1})$

$\text{AppendChild}(u_{i-1}, u_i)$

$\mu(v_i) = u_i$

```

        fi
      od
    fi
  od
  return  $X_2$ 
end

```

#### Example 4.4: Superimpose

The following shows a majority schema. The two subtrees labeled with **DEGREE** are highly similar.

```

<EDUCATION>
  <DEGREE>
    <ORGANIZATION/>
    <DATE/>
    <THESIS>
      <ADVISOR/>
    </THESIS>
  </DEGREE>
  <DEGREE>
    <DATE/>
    <ORGANIZATION/>
    <GPA/>
  </DEGREE>
</EDUCATION>

```

Take the first **DEGREE** subtree as the source tree and the other as the target tree. Suppose the mapping computed by the tree edit distance algorithm on these two

subtrees maps the nodes labeled with `DEGREE`, `ORGANIZATION`, `DATE` in the first `DEGREE` subtree to those of the same label in the second `DEGREE` subtree. Then the merged subtree is:

```
<EDUCATION>
  <DEGREE>
    <DATE/>
    <ORGANIZATION/>
    <GPA/>
    <THESIS>
      <ADVISOR/>
    </THESIS>
  </DEGREE>
</EDUCATION>
```

#### 4.2.3.4 Unifying Subtree Structures

Based on the ideas described in the previous section, we now present the overall unification process. We adopt a hierarchical clustering approach ([Eve73]) for the unification process.

The process is described in the function *Unify(S, distThreshold)*. It takes the majority schema *S* and the parameter *distThreshold* as input. The variables for the clustering algorithm are subtrees in the majority schema. Each node *u* in *S* corresponds to one subtree *T(u)*. A distance matrix, *DistMatrix(u, v)*, stores the distance measure between the subtrees *T(u)* and *T(v)*. At each iteration, the function picks two subtrees, *T(u)* and *T(v)*, in the majority schema that are structurally similar and

their distance measure  $TreeEditDist(T(u), T(v))$  being the smallest. These two subtrees are superimposed by the function *Superimpose*. The merged subtree  $T(v)$  is kept in the schema. The other subtree  $T(u)$  is deleted from  $S$ , and the distance matrix is updated. The process is repeated until there are no more structurally similar subtrees.

**Algorithm 4.15: Unify**

*Function Unify(S, distThreshold) : X*

(Initialize distance matrix)

**for each**  $u, v \in S.V$  **do**

$DistMatrix(v, u) = DistMatrix(u, v) = TreeEditDist(T(u), T(v))$

**od**

(Find two subtrees  $T(u), T(v)$  that are structurally most similar.)

**while**  $\exists u, v \in S.V$ :

$((u \neq v) \wedge$

$(d = DistMatrix(u, v)) \wedge$

$(d \text{ is the minimum distance between any pair of subtrees})$

$(d = Min\{TreeEditDist(T(v_i), T(v_j)) \mid v_i, v_j \in S.V \wedge v_i \neq v_j\}) \wedge$

$(Similar(T(u), T(v), distThreshold))$ )

**do**

(Remove  $T(u)$ . Keep the superimposed subtree.)

$T(v) = Superimpose(T(u), T(v))$

$S.V = S.V - \{v \mid v \in T(u)\}$

```

    (Update the tree distance of any other subtree to  $T(v)$ )
    for each  $w \in S.V, w \neq v$  do
         $DistMatrix(w, v) = DistMatrix(v, w) = TreeDist(T(w), T(v))$ 
    od
od
return S
end

```

The function *Unify* chooses the two subtrees  $T(u)$  and  $T(v)$  that are structurally most similar to unify in each step, but the algorithm can be customized in a number of ways. The typical ways of marking up a document pertaining to the topic may differ only at a certain level. For example, people may have slightly different ways of marking up résumés within the educational background section, i.e. heterogeneity only arises among neighboring subtrees. In that case, the choices of pairs of subtrees to unify can be restrained to siblings instead of all vertices in the majority schema. Since there may be homonyms in the documents (the Homonyms Property in Section 1.2.3), two vertices with the same label in the majority schema are different elements in a DTD (Section 2.2). However, some of these labels may refer to the same element because of the heterogeneity of the documents. Therefore, it may be desirable to select a group of pairs of subtrees that are at similar depths.

As mentioned in Section 4.2.3.1, a tree edit mapping preserves ancestor-descendant relationships. Therefore, if one can swap the parent and child vertices in the subtree



$X_1$  to transform it to the subtree  $X_2$ , the mapping would suggest deleting one and inserting another instead. A more sophisticated algorithm is needed. The problem of merging two subtrees in the majority schema is similar to the problem of transforming a subtree into another subtree (Chapter 5). Ideas similar to those presented in Chapter 5 can be used for this purpose, but will not be explored here.

After the unification process, we derive a majority schema for the collection of XML documents. A majority schema ignores certain information, e.g., order, multiplicity. The next section describes how the content model of nodes in this initial majority schema can be refined.

#### 4.2.4 Enriching DTD Content Model

Content model information, such as order and multiplicity, is ignored in the initial majority schema. The following sections describe how order and multiplicity information of nodes in the initial majority schema can be determined.

##### 4.2.4.1 Order

Given a set of label paths with the same 1-subpath in a majority schema, the last labels of the paths are constituent elements of the second last label in the DTD derived from the majority schema. Instead of enumerating all orders of these labels found in the documents, we pick one sequence that is most typical. To determine

this order, we assume there is a function that gives a partial order on elements. An example of this function is one that computes the average positions of labels. As we compute the initial majority schema, we record the average positions of these labels in the documents. The labels are then sorted by their average positions in the final majority schema.

More precisely, given a tree schema  $S$  for documents  $\mathcal{X}$  and its set of label paths  $\{p \circ q_i\}$ ,  $p \in E^*$ ,  $q_i \in E$ . The *average position* of the label  $q_i$  in the label path  $p$  is

$$AvgPos(q_i, p, \mathcal{X}) := \frac{\sum_{v \in \mathcal{X}, \mathcal{X} \wedge LabelPath(Ancestors(v)) = p \circ q_i} Pos(v, \mathcal{X})}{|\{v \mid v \in \mathcal{X} \wedge LabelPath(Ancestors(v)) = p \circ q_i\}|}$$

where  $Pos(v, \mathcal{X})$  gives the position of vertex  $v$  among its siblings in the document  $\mathcal{X}$ .

The function *BasicCFP* is refined to collect additional information for  $q_i$  - the count of vertices in the documents conforming to  $q_i$  and the sum of their positions in the documents - which is stored as attributes of  $q_i$ . *AvgPos* is the ratio of these two numbers.

Let  $D$  be the DTD representation of  $S$ . Recall that elements in  $S.V$  may be renamed because of the presence of homonyms. Let  $e_i = q_i.rename$  and  $e = p.rename \in E'$  be elements of  $q_i$  and  $p$  in  $D$  respectively. The element  $e$  in  $D$  consists of constituent elements  $e_1, \dots, e_n$ . An initial content model of  $e$  is given by a permutation of  $e_1, \dots, e_n$  to  $e_{i_1}, \dots, e_{i_n}$ :

$$(e_{i_1}, \dots, e_{i_n})$$

where

$$AvgPos(q_{i_j}, p, \mathcal{X}) \leq AvgPos(q_{i_{j+1}}, p, \mathcal{X}), 1 \leq j \leq n - 1$$

The order of the constituent elements  $e_{i_j}$  in the content model of  $e$  is sorted by their average positions.

#### 4.2.4.2 Multiplicity

Regular expression primitives "?" (optional), "+" (required), "\*" (repetitive) can be used to describe the content model of an element. A majority schema describes prevalent structures among XML documents, not all structures in the documents. The support of nodes is used to determine which nodes are prevalent. In other words, frequent label paths describe what the constituent nodes in a majority schema are. The content model of an element can still be optional if its support ratio is too low.

Let  $p$  be a label path in a majority schema  $S$ . Let  $D$  be the DTD representation of  $S$ . Let  $e_i$  be the label of  $p_i$  and  $e$  the label of  $p$ .

The content model of an element  $e_n$  in  $D$  can be based upon the support ratio of  $p_n$ . If the support ratio is high, it occurs at least once within its parent ("+" ). If it is low, it is optional ("?"). This is adjusted by the user-defined parameter  $requiredThreshold \in \mathbf{R}[0, 1]$ .

In addition, we need to determine if the content model of  $e_n$  in  $e_{n-1}$  is "\*". Intuitively, if in the document a vertex conforming to  $p_{n-1}$  has more than a certain number of

children vertices conforming to  $p_n$ , the content model of  $e_n$  in  $e_{n-1}$  is "\*" <sup>3</sup>. Let the function  $IsRepPath(p, e)$ ,  $p = \langle v_1, \dots, v_n \rangle \in \mathcal{X}$  be a boolean function that determines if  $v_n$  has more than a certain number of child nodes labeled with  $e$ . The repetitive content model of  $e_n$  along the label path  $p_e = e_0 \circ \dots \circ e_n$  can be determined based on the proportion of such node paths in the document collection:

$$RepSupportRatio(p_e) := \frac{|\{p_v \mid p_v \oplus v_n \in \mathcal{X} \wedge IsRepPath(p_v, e_n) \wedge LabelPath(p_v \oplus v_n) = p_e\}|}{|\mathcal{X}|} \in \mathbf{R}[0, 1]$$

If  $RepSupportRatio(p_e, \mathcal{X})$  is greater than the user-defined parameter  $repeatThreshold$ , the content model of  $e_n$  in  $e_{n-1}$  is then "\*" .

The following rules are then used to determine the content model of elements in the DTD:

- if  $RepSupportRatio(p_n) > repeatThreshold$ , content model of  $e_n$  in  $e_{n-1}$  is \*
- else if  $SupportRatio(p_n) > requiredThreshold$ , content model of  $e_n$  in  $e_{n-1}$  is +
- else if  $SupportRatio(p_n) \leq optionalThreshold$ , content model of  $e_n$  in  $e_{n-1}$  is ?

In sum, we have discovered a global schema in form of a DTD that describes prevalent structures in the collection of XML documents. Order and multiplicity information about elements in the DTD is expressed in their content model. However, the grouping

---

<sup>3</sup>An empirical study shows that 3 is a good choice, which is also suggested by [GGR<sup>+</sup>00]. This is because in practice, if a concept occurs 3 or more times under another concept, it is intuitive for a user to model it as a repetitive element rather than repeating it 3 or more times.

of constituent elements is not modeled. For instance, the content model  $(p_1^*, p_2^*)$  can be discovered, but not  $(p_1, p_2)^*$ .

## 4.3 Evaluation

In this section, we evaluate the majority schema discovery approach with respect to three aspects. First, we analyze its computational complexity. We then evaluate the quality of a majority schema against the criteria of conciseness and coverage. Finally, we give a sample DTD discovered for over 1000 résumés to demonstrate the practicality of the whole approach for a particular setting.

### 4.3.1 Time Complexity

We analyze the time complexity of the functions *BasicCFP* and *Unify* with no domain knowledge specified. The time complexity of the function *DomKnowCFP* is not analyzed because its efficiency in real applications depends on the domain knowledge specified.

### 4.3.1.1 Function *BasicCFP*

Computing *maxPathLength* requires walking through all nodes in the document collection, which takes  $O(|\mathcal{X}|)$  time. As we walk through the nodes in the document collection, we can also count the support of the node paths from the root to the nodes. A hash table can be built to store the support of node paths. In the two outer **for** loops, we extend label paths of increasing length. Since each path  $p$  in  $\mathcal{X}$  is considered at most once, the total number of label paths considered is bound by  $|\mathcal{X}|$ . The number of times  $p$  is extended to  $p \circ e$  is bound by  $Deg(\mathcal{X})$ . Looking up the support of a label path from the hash table takes constant time. Hence, we have

**Theorem 4.1 (Time Complexity of BasicCFP).** *Given a collection of XML documents  $\mathcal{X}$ . The function *BasicCFP* takes time  $O(Deg(\mathcal{X})|\mathcal{X}|)$ , which is linear in the size of  $\mathcal{X}$ .* □

### 4.3.1.2 Function *Unify*

Let us first consider the function *Superimpose*. Initializing the root of  $X_1$  takes constant time. The outer **for** loop iterates over vertices in  $X_1$  which is bound by  $|X_1|$ . Locating the label path from its reference ancestor to each vertex is bound by the depth of  $X_1$ . Therefore, the function *Superimpose* takes  $O(Depth(X_1)|X_1|)$  time.

Let us now consider the function *Unify*. Let  $ted(T_1, T_2)$  denote the time to compute the minimum cost edit sequence to transform the source tree  $T_1$  into the target tree  $T_2$ . There are  $O(|X|^2)$  choices of pairs of subtrees in  $X$ . Therefore, initialization takes  $O(ted(X, X)|X|^2)$  time.

In the **while** loop, choosing the pair of subtrees with the minimum tree distance requires a scan of the distance matrix, which takes  $O(|X|^2)$  time. Superimposing the pair takes  $O(Depth(X)|X|)$  time. Updating the distance matrix takes  $O(ted(X, X)|X|)$  time. Each iteration in total takes  $O(|X|^2 + Depth(X)|X| + ted(X, X)|X|) \subseteq O(|X|^2 + ted(X, X)|X|)$  time.

The number of iterations of the **while** loop depends on the degree of heterogeneity of the majority schema. Empirical studies show that it usually takes several iterations because the documents are topic specific and fairly similar. Under this condition, we have

**Theorem 4.2 (Time Complexity of Unify).** *Given a majority schema  $X$ . The function *Unify* takes time  $O(|X|^2 + ted(X, X)|X|)$ .* □

### 4.3.2 Quality of Majority Schema

In practice, we would like to have a schema of high quality. Similar to the criteria proposed by [LPVV99], the schema should be exact, i.e. it describes all structures found in the documents and those structures only. The set of documents conforming

to an exact schema should be the collection of the documents. It should be concise, i.e. it should succinctly describe the documents. However, a concise schema can be too general in that it covers many cases not found in the documents (not relevant), or it ignores cases found in the documents (low coverage). This is illustrated in Figure 4.2.

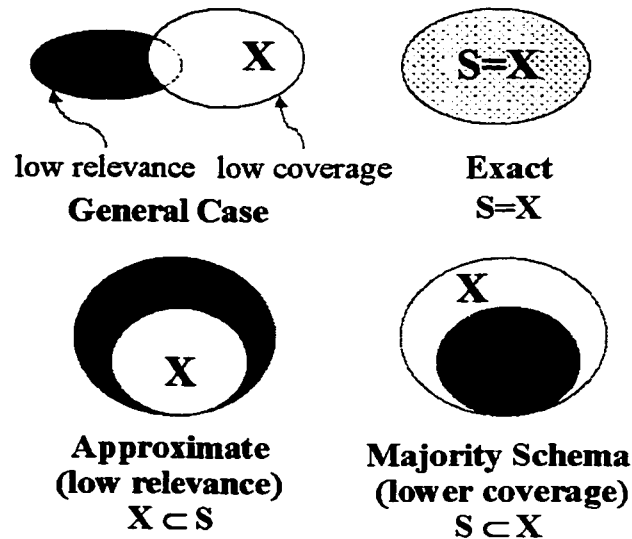


Figure 4.2: Schema Quality

The quality of a schema can be described by its relevance and coverage. An exact schema describes all input documents and those documents only. An approximate schema of low relevance but full coverage may describe documents not found in the input document collection. A majority schema is an approximate schema of low coverage but full relevance. It describes only a subset of the document structures that are representative in the input document collection.



Note that these criteria depend on the set of documents conforming to a schema. They are independent of the underlying formalism of the schema. The formalism of the schema determines the mechanism to check if a document conforms to it. For example, bisimulation can be used to check if a document conforms to a graph schema and schema subsumption to compare two graph schemas [BDFS97]. The type checking approach in [MSV00] can be used to check if an XML document is valid with respect to a DTD.

In the following sections, we first formulate the criteria to evaluate the quality of a majority schema. Then, we present empirical results on the evaluation of majority schemas against these criteria.

#### 4.3.2.1 Evaluation Criteria

The conciseness of a majority schema can be defined in terms of its size, i.e. its number of vertices.<sup>4</sup> It is normalized with respect to the size of the majority schema at  $supThreshold = 0$  and  $ratioThreshold = 0$ , which gives an upper bound on the sizes of majority schemas over all parameters. This majority schema is still an approximate schema because choice ("|") in a content model are not considered. Thus, its size is not larger than an exact schema. Hence, the conciseness measure is biased

---

<sup>4</sup>A more precise measurement may consider the number of bits to encode the majority schema, like the minimum length description principle described in [GGR<sup>+</sup>00]

towards the conservative which may underestimate how concise a majority schema actually is.

**Definition 4.6 (Conciseness).** *Given a collection of XML documents  $\mathcal{X}$  and parameters  $supThreshold$  and  $ratioThreshold$ . Let  $S$  be the majority schema for  $SDP[\mathcal{X}, supThreshold, ratioThreshold]$ . Let  $S_u$  be the majority schema for  $SDP[\mathcal{X}, 0, 0]$ . The conciseness of  $S$  is given by*

$$Conciseness(S) := 1 - \frac{|S.V|}{|S_u.V|} \in \mathbf{R}[0, 1]$$

□

The coverage of a schema can be defined in terms of the number of vertices in the documents conforming to the schema. It reflects the portion of the documents that are described by the schema. In Figure 4.2, it is  $\frac{|S \cap \mathcal{X}|}{|\mathcal{X}|}$  where  $S$  is the set of documents conforming to  $S$ .

**Definition 4.7 (Coverage).** *Given a collection of XML documents  $\mathcal{X}$  and a majority schema  $S$ . The coverage of  $S$  with respect to  $\mathcal{X}$  is given by*

$$Coverage(S) := \frac{|\{v \mid v \in \mathcal{X} \wedge u \in S \wedge LabelPath(u) = LabelPath(v)\}|}{|\mathcal{X}|} \in \mathbf{R}[0, 1]$$

□

The *relevance* of a schema concerns the document structures conforming to the schema, but not found in the input documents. In Figure 4.2, it is  $\frac{|S \cap \mathcal{X}|}{|S|}$  where  $S$  is the set of documents conforming to  $S$ . Since there are infinitely many documents

that can conform to a tree schema, this cannot be directly computed, but can be analyzed by schema subsumption [BDFS97]. Nevertheless, we do not need to consider this evaluation criteria for majority schemas. A majority schema is always relevant since there is at least one input document with a node path conforming to some node path in the schema. We mention this criteria only for the sake of completeness.

#### 4.3.2.2 Experiments

We conducted an empirical study to evaluate the majority schema obtained for résumé documents. We chose, at random, three datasets of increasing sizes of 40, 80 and 120 résumés, respectively. We set *supThreshold* = 0. Majority schemas at different parameters of *ratioThreshold* are discovered for each dataset. The conciseness and coverage of the schemas are shown in Figure 4.3.

Figure 4.3 shows that conciseness increases with *ratioThreshold* while coverage decreases. This is expected since a concise schema ignores structures shared by the minority of the documents and hence has lower coverage. Conciseness increases sharply at low *ratioThreshold* at a moderate cost of losing coverage. For example, compared to the most precise majority schemas (*ratioThreshold* = 0), the majority schemas at *ratioThreshold* = 0.1 boost the conciseness from 0 to 0.6-0.7 at 0.8 coverage. The majority schemas at *ratioThreshold* = 0.2 have a conciseness of 0.8-0.9 at 0.7 coverage. This demonstrates the usefulness of majority schemas in describing prevalent structures in topic specific documents without losing much coverage. Topic specific

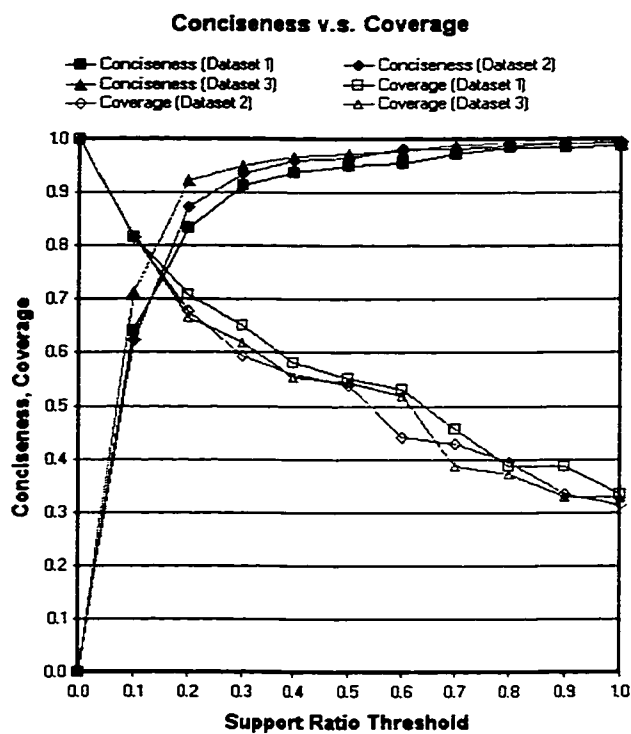


Figure 4.3: Evaluation of majority schema against conciseness and coverage

documents follow some typical logical layout. Minority patterns scatter in the search space and can be filtered out using a low *ratioThreshold*.

Figure 4.3 also shows the actual sizes of the majority schemas. At *ratioThreshold* = 0, the size of the schema increases with the size of the dataset. The larger the size of the dataset, the more heterogeneous it is. A larger schema is then required to describe them. However, the sizes of the majority schemas converge very quickly at *ratioThreshold* = 0.2. This implies that the underlying schema of the input

documents is fairly stable with respect to the heterogeneity, which further confirms our assumption that topic specific documents follow a similar style.

### 4.3.2.3 Sample DTD

We conclude this section by giving the DTD representation of a majority schema discovered for over 1000 résumés at *supThreshold* = 0 and *ratioThreshold* = 0.1. We set *repeatThreshold* = 3, *requiredThreshold* = 0.6 and *optionalThreshold* = 0.3. The attribute *val* has been discarded and #PCDATA has been associated with each element instead. The root of each document has been renamed to the topic name *resume*. Manual inspection of the DTD reveals that the schema discovered indeed agrees with common sense of how a résumé document should be structured.

```

<!ELEMENT resume      ((#PCDATA), contact+, objective, education+,
                        courses, experience+, awards, achievements+,
                        skills, activities+, reference)>
<!ELEMENT contact    (#PCDATA)>
<!ELEMENT objective  (#PCDATA)>
<!ELEMENT education  ((#PCDATA), (institute, date-entry))>
<!ELEMENT institute  (#PCDATA)>
<!ELEMENT date-entry ((#PCDATA), degree)>
<!ELEMENT degree     (#PCDATA)>
<!ELEMENT courses    ((#PCDATA), date+)>
<!ELEMENT date       (#PCDATA)>
<!ELEMENT experience ((#PCDATA), (date+, title+))>
<!ELEMENT title      ((#PCDATA), institute)>
<!ELEMENT awards     ((#PCDATA), date-entry3+)>
<!ELEMENT date-entry3 ((#PCDATA), (institute, award+))>
<!ELEMENT achievements ((#PCDATA), (date+, institute2+))>
<!ELEMENT institute2 ((#PCDATA), (date, title2))>

```

```
<!ELEMENT title2      (#PCDATA)>
<!ELEMENT skills      ((#PCDATA), (skill, date))>
<!ELEMENT skill        (#PCDATA)>
<!ELEMENT activities   ((#PCDATA), (date+, title))>
<!ELEMENT reference    ((#PCDATA), (date+, title))>
```

## 4.4 Related Work

In this section, we describe existing schema discovery approaches. We first classify these approaches according to different dimensions. Then we describe where our approach stands with respect to these dimensions, compare our approach to schema discovery to existing ones, and highlight the contributions of our work.

### 4.4.1 Classification

There have been a number of proposals to schema discovery on semistructured data in the literature ([BDFS97, NUWC97, GW99, WYW00, NAM98, NAM97, WL99, WL98, GGR<sup>+</sup>00, PV00]). These approaches can be classified according to several dimensions: data model underlying the input data, type of schema, precision of schema, and the type of structures considered.

[BDFS97, NUWC97, GW99, WYW00, NAM98, NAM97, WL99, WL98] take a data-centric view on a document collection in a data model similar to OEM (Section 2.1.1). The schema under the data-centric view describes incoming and outgoing labels from

objects, either as a graph schema [BDFS97] or similar to a Dataguide [NUWC97]. A graph schema specifies *permissible* structures and allows unary predicates on edges while a Dataguide specifies *required* structures. Whether a database conforms to a schema is checked by simulation [BDFS97]. Constraints on cardinality, order and grouping of nodes are not modeled in both schema formalisms. [GGR<sup>+</sup>00, PV00] take a document view on a document collection using a data model similar to Infoset (Section 2.1.1). They use DTDs to describe the structures of the documents. Graph schemas and Dataguides are not as expressive as DTDs. The content model in a DTD can describe multiplicity, variants and grouping of elements.

The schemas discovered by these approaches differ in their degree of precision. [NUWC97, PV00] discovers an exact schema which gives a precise description on the input documents. [GW99, WYW00, WL99, WL98, NAM98, NAM97, GGR<sup>+</sup>00] infer schemas which are approximate or abstracted representation of the documents. They consider different types of information. Data-centric approaches view the documents as a web of objects and hence consider cycles in the linkage structure. Most approaches (except [NAM98, NAM97]) consider multiplicity information of labels in the schema. [WL98] also considers wild cards in the schema.

There is a wide range of schemas inferred because different approaches have different goals. The primary goal of [BDFS97, NUWC97, GW99, WYW00, WL99, WL98, NAM98, NAM97] is to answer path queries on a web of hypertext documents. Hence they focus on the possible incoming and outgoing labels of objects. [GGR<sup>+</sup>00, PV00]

aim to describe the document structures of a collection of related XML documents. Therefore, they use DTDs as the formalism of their schema.

## 4.4.2 Comparison

Our goal is to describe the structures of a collection of related documents. Due to the heterogeneity among the documents, we prefer a schema with a balance between precision and size of the schema.

### 4.4.2.1 Data Model

Since we are interested in the document structures of the XML documents, we take a document-centric view on the collection. We thus model XML documents in a way similar to [W3C99a, GGR<sup>+</sup>00, PV00].

We prefer a schema formalism similar to [BDFS97, NUWC97] that can be directly compared to the documents to facilitate the document transformation process (Chapter 5). However, [BDFS97, NUWC97] do not differentiate where a label path originates. This information is semantically significant in a document structure. In view of this, we propose to use a tree schema as the formalism for the majority schema. It has the same formalism as XML documents and considers their document tree structures. Furthermore, a tree schema can always be converted to a DTD.



#### 4.4.2.2 Precision

Due to the heterogeneity of the documents, exact schemas as proposed in [NUWC97, PV00] are inappropriate for our purpose because of their large size. Therefore, we propose the notion of a majority schema. Users can adjust the degree of precision of the majority schema by the parameters *supThreshold* and *ratioThreshold*. This offers more flexibility for different applications.

A subtlety is that the approximation of majority schemas is different than that proposed in [GW99, WYW00, NAM98, WL98]. In these approaches, all structures found in the documents are still permissible in the schema, i.e. there is no loss in coverage. They are less *relevant* because they also allow structures not found in the documents. On the other hand, majority schemas do not describe all structures in the documents, i.e. they lose coverage. It nevertheless covers structures that are prevalent and hence representative in the documents. Since they only describe structures found in the documents, there is no loss in relevance.

The differences arise because of different design goals. The goal of [WYW00, NAM98, NUWC97, GW99, WL98] is to infer a schema that can be used in optimizing path queries. Therefore, the schema should not lose coverage. But it can be less relevant, which may result in less relevant results to queries. Our goal is to infer a schema that abstracts the heterogeneity of topic specific documents which can be used as a basis for integrating these documents. Therefore, losing the coverage of less prevalent structures is desirable in presenting a more succinct view on the documents. Het-

erogeneity among the documents may not be desirable from the point of view of end users, which makes visual browsing to filter out information difficult. Heterogeneity also complicates the task of managing these documents. Moreover, maintaining coverage is not crucial since the structures in the documents that are not found in the majority schema will be transformed to conform to it (Chapter 5).

An application of majority schemas is to give users unfamiliar with the documents a bird's eye view of the documents much like a table of content that leaves out the details. Majority schemas can also be used to optimize storage (Chapter 6). Nodes in a document that are close in proximity in the majority schema are likely to be accessed together and thus can be stored together. Tools like [DFS99] can be used to store a repository of XML documents in a relational database for efficient data management purpose based on a majority schema. Another application of majority schemas is indexing. Nodes in the majority schema with the high support are most commonly found. Assuming this implies that structures in the documents conforming to these nodes are most commonly accessed, they can be indexed for more efficient retrieval.

#### 4.4.2.3 Others

With respect to the type of structures considered, a tree schema is similar to tree expressions in [WL99, WL98]. Unlike [WL99, WL98, GGR<sup>+</sup>00], we take the opposite approach of ignoring certain information in the beginning. There are a number

of technical complications in [WL99, WL98, GGR<sup>+</sup>00] because [WL99, WL98] model the input data very precisely, and [GGR<sup>+</sup>00] takes everything into account in defining their problem space. We ignore certain information in the discovery process and focus our attention on discovering an initial schema with no information on the content model of the nodes. Information on the content model (order, multiplicity, grouping) is filled in later. This greatly simplifies the discovery process because the Apriori condition holds. This approach is justified because the input documents are too heterogeneous to be described precisely. Our approach uses the intuition that imprecise data modeling reveals prevalent patterns among data, which leads to efficient data mining processes. This is a useful guiding principle in data mining if the underlying patterns to be discovered are simple.

Most of the approaches aforementioned do not consider domain knowledge which is usually available for topic specific documents. [GGR<sup>+</sup>00] has some built-in heuristics to reduce the search space of the schema which would be intractable otherwise. However, they neither discuss nor justify the heuristics they choose. One of our contribution is the explicit incorporation of domain knowledge in inferring a majority schema. We provide a mechanism to allow users to explicitly specify application-specific domain knowledge on the input documents which is easily extensible and understandable. An empirical study demonstrates the feasibility of using simple and general domain knowledge in deriving a majority schema for résumés.

Reference	Data Model	Schema Model	Schema Type	Remarks
[NUWC97] *GW97*	OEM	Dataguide	exact	
[GW99]	OEM	Dataguide	approximate (relevance)	heuristics to group similar objects
[WYW00]	OEM	Dataguide	approximate (relevance)	clustering approach to group similar objects
[WL98, WL99]	OEM	Dataguide	approximate (relevance, coverage)	consider wild cards
[NAM98, NAM97]	OEM	Dataguide	approximate (relevance)	typing approach
[GGR <sup>+</sup> 00]	Infoset	DTD	approximate (relevance)	
[PV00]	Infoset	DTD	exact	Enrich DTD by CFG, stratified DTD
Quixote	Infoset	DTD	approximate (coverage)	stratified DTD

Figure 4.4: Comparison on Schema Discovery Approaches

## Chapter 5

# Document Transformation

After running the Document Converter and the Schema Miner, the collection of topic specific HTML documents has been converted to XML documents, and a majority schema that describes prevalent structures among these documents has been inferred. Since the majority schema covers prevalent structures only, some of the structures in the documents may not conform to the majority schema. The goal of the document transformation process, realized by the Document Transformer component of Quixote, is to transform the documents so that they all conform to their majority schema.

In Section 5.1, we describe how a transformation should preserve the semantics of a document and define the document transformation problem accordingly. In Section 5.2, we present the approach to the document transformation problem. In Section 5.3, we prove the correctness of the approach and analyze its computational complexity. Related work is compared and discussed in Section 5.4.

## 5.1 Problem Formulation

The input to the Document Transformer is a set of topic specific XML documents and a majority schema for them. The majority schema is in form of a tree schema without repetitive label paths. The goal of the Document Transformer is to transform an XML document from the collection, if necessary, so that (1) it conforms to the majority schema, and (2) the context of concept nodes in the XML document is stored in the transformed tree. The former concerns the issue of transforming the heterogeneous XML documents into a homogeneous collection. The latter concerns the issue that semantics are preserved during the transformation. Section 5.1.1 describes the issue of semantic preservation in a transformation. Section 5.1.2 formally defines the document transformation problem.

### 5.1.1 Semantic Preservation

We first describe the issues involved in preserving semantics in a transformation. For example, consider the following XML document and majority schema:

```
<EDUCATION>
  <ORGANIZATION val="UC Davis">
    <DEGREE val="M.Sci.(Comp.Sci.)"/>
  </ORGANIZATION>
  <ORGANIZATION val="Stanford University">
    <DEGREE val="B.Sci.(Comp.Sci.)"/>
  </ORGANIZATION>
</EDUCATION>
```

```
<EXPERIENCE>
  <TITLE val="Summer Intern">
    <ORGANIZATION val="IBM Almaden Research Center"/>
    <DATE val="1994"/>
  </TITLE>
</EXPERIENCE>
```

A majority schema

```
<EDUCATION>
  <DEGREE>
    <ORGANIZATION>
    <DATE/>
  </DEGREE>
</EDUCATION>
<EXPERIENCE>
  <TITLE>
    <ORGANIZATION/>
  </TITLE>
</EXPERIENCE>
```

Our goal is to store the context of concept nodes in the document tree obtained by the transformation. Hereafter, we will call this tree the transformed tree. The context of a concept node includes: (1) its concept name, (2) its textual content (in its attribute *val*), and (3) its neighborhood in the tree. Based on the context, we conclude that the concept nodes `ORGANIZATION` and `DEGREE` under `EDUCATION` in the document tree are *semantically related* to those of the same name in the majority schema, so are `TITLE` and `ORGANIZATION` under `EXPERIENCE`. The `DATE` concept node under `EXPERIENCE` in the document tree, however, is not semantically related to `DATE` under `EDUCATION` in the majority schema because their neighborhoods in the trees are different.

It is not always possible to find a semantically related node in the DTD for a node in the document tree. These nodes represent information objects not found in the majority of the document collection. For instance, `DATE` in `EXPERIENCE` in the document tree does not have any semantically related node in the majority schema. The textual contents of these concept nodes can be kept in parent nodes they detail information about.

Conversely, there may be nodes in the majority schema that are not semantically related to any node in a given document tree. These are concepts found in the majority of the document collection but which are missing in the document. For example, the `DATE` information object under `EDUCATION` based on the majority schema is missing in the document. Since the information is missing in the document, the textual contents of such nodes are unknown in the transformed tree.

A transformation that preserves the semantics of the document is shown below. `ORGANIZATION` and `DEGREE` in the document are re-arranged to conform to the schema. `DATE` in `ORGANIZATION` is deleted because there is no `DATE` in the majority schema. The textual content of `DATE` under `EXPERIENCE` in the document tree is stored in the *val* attribute in `TITLE` as `(DATE:1994)`. Semantics are preserved in the sense that the textual content of all nodes in the XML document are stored in some related nodes in the transformed document. These nodes in the transformed document are labeled with the same concept name or are in close proximity in the neighborhood of the nodes in the XML document.



Transformed XML Document

```
<EDUCATION val="(Date:1993)">
  <DEGREE val="M.Sci.(Comp.Sci.)">
    <ORGANIZATION val="UC Davis">
      <DATE/>
    </DEGREE>
  <DEGREE val="B.Sci.(Comp.Sci.)">
    <ORGANIZATION val="Stanford University">
      <DATE/>
    </DEGREE>
</EDUCATION>
<EXPERIENCE>
  <TITLE val="Summer Intern (DATE:1994)">
    <ORGANIZATION val="IBM Almaden Research Center"/>
  </TITLE>
</EXPERIENCE>
```

### 5.1.2 Problem Definition

Based on the discussion in Section 5.1.1, we give a formal definition of the document transformation problem.

Essential to the idea of semantic preservation is identifying semantically related nodes in the document tree and the majority schema. We capture this aspect by a *semantic mapping*, that is, a many-to-one total function that maps a node in the document tree to a node in the majority schema to which it is semantically related. Formally,

**Definition 5.1 (Semantic Mapping).** *Given an XML document  $X$  and a majority schema  $S$ . A semantic mapping  $\nu : X.V \rightarrow S.V$  is the following total function:*

$$\nu(x) = \begin{cases} S.root & \text{if } x = X.root \\ s \in S & \text{iff } x \text{ and } s \text{ are semantically related} \\ \perp & \text{iff } x \text{ is not semantically related to any node in } S \end{cases}$$

□

However, semantic mappings may not be readily available to Quixote without further input from the user. In practice, having the user manually label semantic relationship between each XML document and a majority schema is infeasible due to the heterogeneity among the documents and the number of documents. In order to build a useful tool, we have to derive the semantic mappings automatically.

The document transformation problem is described as follows:

**Document Transformation Problem:**

Given a collection  $\mathcal{X}$  of XML documents, a majority schema  $S$  for  $\mathcal{X}$ , and a document  $X \in \mathcal{X}$ . The Document Transformation Problem  $DTP[X, S]$  is divided into two tasks.

The first task is to compute a semantic mapping  $\nu : X.V \rightarrow S.V$ .

The second task is to compute an XML document  $M$  with a mapping  $\theta : X.V \rightarrow M.V$  such that  $\theta$  has the following properties:

- *Conformance Property*:  $\mathbf{M}$  conforms to  $\mathbf{S}$  under the mapping  $\mu : \mathbf{M.V} \rightarrow \mathbf{S.V}$ .
- *Content Preservation Property*: Consider a node  $x$  in  $\mathbf{X}$ . Suppose it is semantically related to  $s = \nu(x) \in \mathbf{S}$ . The textual content of  $x$ ,  $x.val$ , is stored in some node  $m$  in  $\mathbf{M}$  conforming to  $s$ :

$$x \in \mathbf{X.V} \wedge s = \nu(x) \Rightarrow$$

$$\exists m \in \mathbf{M.V}: (s = \mu(m) \wedge x.val \in m.val)$$

- *No Information Loss Property*: Consider a node  $x$  in  $\mathbf{X}$ . Suppose it is not semantically related to any node in  $\mathbf{S}$ . The textual content of  $x$  is stored in some higher level concept node  $m$  in  $\mathbf{M}$  it details information about.

$$x \in \mathbf{X.V} \wedge \nu(x) = \perp \wedge a = RefAncestor(x, \nu) \Rightarrow$$

$$\exists m \in \mathbf{M.V}: (m = \mu(a) \wedge x.val \in a.val)$$

$a$  is the referencing ancestor of  $x$  by  $\nu$  whereas  $m$  is the node in  $\mathbf{M}$  corresponding to  $a$ . The textual content of  $x$  is stored in  $m$ .

$\mathbf{M}$  is the document transformed from  $\mathbf{X}$  so that  $\mathbf{M}$  conforms to the majority schema  $\mathbf{S}$ .

Let us make a remark on the functions  $\nu$ ,  $\mu$  and  $\theta$ .  $\nu$  is the semantic mapping from  $\mathbf{X}$  to  $\mathbf{S}$ .  $\mu$  is the conformance mapping from  $\mathbf{M}$  to  $\mathbf{S}$ .  $\theta$  is the transformation mapping from  $\mathbf{X}$  to  $\mathbf{M}$ . Therefore, we have  $\theta = \nu \cdot \mu^{-1}$ .

## 5.2 Approach

In this section, we present our solution to the document transformation problem. Section 5.2.1 describes how a semantic mapping from an XML document to a majority schema is derived. Section 5.2.2 describes how the semantic mapping is used to transform the document to conform to the majority schema with semantics preserved.

### 5.2.1 Derivation of Semantic Mapping

As mentioned in Section 5.1.2, a semantic mapping between a document tree and the majority schema may not be readily available in practice. Therefore, we present an algorithm to derive such semantic mapping automatically based on the domain knowledge we have on topic specific documents and their majority schema. Section 4.2.3.1 described how deriving a semantic mapping can be related to the tree edit problem. Section 5.2.1.2 describes how domain knowledge can be used to customize the cost model of the tree edit problem to derive a semantic mapping. This semantic mapping is further refined in Section 5.2.1.3.

An overview on how to derive a semantic mapping is outlined in the function *DeriveSemanticMapping*. The majority schema  $S$  does not contain repetitive label paths. We thus first convert the document  $X$  to  $X'$  in a "bisimulated" form comparable

to  $S$  by the function *Bisimulate*.<sup>1</sup> Based on a customized cost model *CostModel*, the tree edit distance algorithm *TreeEdit* computes the mapping  $\nu$  between the document  $X'$  and the majority schema  $S$ . The semantic mapping computed is further refined by *RefineSemanticMapping*.

**Algorithm 5.16: DeriveSemanticMapping**

*Function DeriveSemanticMapping*( $X, S$ ) :  $V \rightarrow V$

**begin**

$X' = \text{Bisimulate}(X)$

$\nu = \text{TreeEdit}(X', S, \text{CostModel})$

$\nu = \text{RefineSemanticMapping}(X', S, \nu)$

**return**  $\nu$

**end**

### 5.2.1.1 Tree Edit Problem

Although we cannot assume that the user manually inspects each XML document, we make the following observations on an XML document and the majority schema for a collection of documents:

---

<sup>1</sup>Let  $\sigma$  be the mapping from  $X$  to  $X'$ . Taking this technical detail into account, the relationship between the functions  $\nu$ ,  $\mu$  and  $\theta$  is  $\theta = \sigma \cdot \nu \cdot \mu^{-1}$ . Hence, a semantic mapping gives us a transformation from  $X'$  to  $S$ .

1. A concept node in the document tree and its semantically related node in the majority schema are labeled with the same concept name. If concepts are identified from the HTML text in the Document Converter based on a Bayes classifier, we have statistical information on the relative probability of associating other concept names with a concept node. This gives us information of how to associate concept nodes in the document tree and the majority schema labeled with different concept names.
2. The majority schema is derived from the input collection of topic specific documents. Since the input documents are topic specific, they share some similarities. The majority schema describes the most prevalent structures among these documents. Hence, one can expect that an arbitrary document from the input collection shares some similarity in structures with the majority schema. Hence, the neighborhood of the nodes can be used to guide the identification of semantically related nodes.

Therefore, semantically related nodes in the document tree and the majority schema can be identified based on their concept names and their relationship to other nodes in the tree. This is an instance of a tree edit distance problem. Since the majority schema does not contain repetitive label paths, the document tree is first converted to another tree with all repetitive label paths removed and nodes sharing the same ancestor label path merged, similar to the concept of bisimulation presented in [BDFS97]. Let us call this tree the "bisimulated" tree of  $X$ . We may refer to the bisimulated tree as the document tree in this section.

The bisimulated tree of  $X$  is then compared against the majority schema. Taking the bisimulated tree as the source tree and the majority schema as the target tree, a tree edit distance algorithm computes a mapping from the XML document to the majority schema which is the semantic mapping we seek to obtain.

It should be noted that information is not lost in the bisimulated tree by merging nodes with the same ancestor label path. Based on the Regular Intradocument Format Property (Section 1.2.3), nodes with the same ancestor path have the same context. They all correspond to one concept node in the majority schema and hence their structures are described by the same element in the DTD representation of the majority schema. The following gives an example of an XML document and its bisimulated tree. The two `DEGREE` nodes in the XML document share the same ancestor label path and hence are merged into one `DEGREE` node in the bisimulated tree.

```
<EDUCATION val="Academic Background">
  <DEGREE val="M.Sci. (Comp. Sci.)">
    <DATE val="1999"/>
    <ORGANIZATION val="Stanford University"/>
    <GPA val="4.0/4.0"/>
  </DEGREE>
  <DEGREE val="B.Sci. (Comp. Sci.)">
    <DATE val="1998"/>
    <ORGANIZATION val="UC Davis"/>
    <GPA val="3.7/4.0"/>
  </DEGREE>
</EDUCATION>
```

Its bisimulated document

```
<EDUCATION>
  <DEGREE>
```

```
<DATE/>
<ORGANIZATION/>
<GPA/>
</DEGREE>
</EDUCATION>
```

The tree edit distance algorithm computes a minimum cost sequence of edit operators (*Insert*, *Delete*, *Rename*) on the document tree to transform it to match the majority schema. A node in the document tree for which the algorithm cannot find a close match in the majority schema may be marked "deleted". These nodes are called *deleted nodes*. Similarly, the algorithm may "insert" a node into the document tree to make it look like the majority schema. This is a node that is found in the majority schema with no similar node in the document. Such nodes are called *inserted nodes*. Deleted and inserted nodes are not really deleted from or inserted into the document tree. Rather, it means that the algorithm cannot find a node in the majority schema or the document tree that is similar to the node. The algorithm may also rename a node in the document tree so that it matches the majority schema. A node in the document tree that is not deleted or inserted is mapped to some node in the majority schema. This mapping gives us the semantic mapping from the document tree to the majority schema.



### 5.2.1.2 Cost Model

A tree edit distance algorithm computes a mapping so that the sequence of edit operations in transforming the document tree is of minimum cost. The cost of the edit operators - *Insert*, *Delete* and *Rename* - is specified by a cost model. A simple cost model is to assign unit cost to each edit operator. Such a cost model is not likely to compute a good semantic mapping. In this section, we describe how domain knowledge can be used to customize the cost model so that we can derive a better semantic mapping. This is based on the following observations:

- *Observation 1*: Since semantically related information objects are associated with the same concept name and there are no synonyms among the concept names, we can assume two concept nodes in the document tree and the majority schema can only be semantically related if they share the same label. In this simple cost model, the cost of the *Insert* and *Delete* operators can be unit cost. The cost of the *Rename* operator on a node can be infinity. If two nodes share the same label, there is no cost of renaming, i.e. the cost of *Rename* is zero.
- *Observation 2*: The cost model can be defined not only on concept names but also extended to concept nodes. The support ratio of a concept node in its tree gives a measure of the importance of that node. The tree edit distance algorithm may delete a concept node from the document tree that is not found in the majority schema. The cost of deleting a concept node with higher support ratio should be higher than that of one with lower support ratio. Similarly, the

tree edit distance algorithm may insert a node to the document tree because that node is found in the majority schema, but not in the document tree. Inserting a node with higher support ratio in the majority schema should have a lower cost than one with smaller support ratio.

- *Observation 3:* If concepts are identified based on Bayes classifier by the Document Converter, we have statistical information on associating other concept names with a concept node. The Document Converter chooses the concept name  $e$  with the highest relative probability as the label of the HTML text involved. A concept name  $e'$  with a slightly lower relative probability can as well be associated with the HTML text. The cost of renaming the label of the concept node in the document tree from  $e$  to  $e'$  in the majority schema then can be defined based on the relative probability of associating the HTML text with  $e$  and  $e'$ .
- *Observation 4:* The context of a concept node is not only its textual content, but also its relationships with other concept nodes in the document tree. The cost of the *Rename* operator depends not only on the concept names of nodes but also on comparing the neighborhood of nodes in the document and in the majority schema.

Based on the above observations, we refine the cost model for the tree edit problem to compute the semantic mapping from the document tree to the majority schema. Let  $Cost(insert, p, v)$ ,  $Cost(delete, v)$  and  $Cost(rename, v, e)$  denote the cost of inserting

the node  $v$  as a child of node  $p$ , deleting the node  $v$ , and renaming the node  $v$  to label  $e$  respectively. Let  $\mathbf{X}$  be an XML document and  $\mathbf{S}$  a majority schema.

**Insert Operator:** Consider the *Insert* operator. A node  $x$  can be inserted into the document tree because there is a node  $s$  of the same label path in the majority schema. Based on Observation 2, the cost of inserting  $x$  should be lower if the support of  $s$  is high in the majority schema. Given node paths  $p_x = \langle x_0, \dots, x_n \rangle \in \mathbf{X}$ , and  $p_s = \langle s_0, \dots, s_n \rangle \in \mathbf{S}$  where  $p_s \oplus s \in \mathbf{S}$ , we have

$$\text{Cost}(\text{Insert}, x_n, x) := 1 - \text{SupportRatio}(s, \mathbf{S}) \in \mathbf{R}[0, 1]$$

**Delete Operator:** Similarly, the cost of deleting a node  $x$  from the document tree  $\mathbf{X}$  is high if the node's support is high in the tree. That is,

$$\text{Cost}(\text{Delete}, x) := \text{SupportRatio}(x, \mathbf{X}) \in \mathbf{R}[0, 1]$$

**Rename Operator:** By Observation 3, the cost for renaming a node  $x$  in  $\mathbf{X}$  to another label  $e$  can be based on the relative probability of classifying  $x.val$  to  $e$  instead of  $\text{Label}(x)$ . A formulation of the cost for renaming  $x$  to  $e$  can be

$$\text{BayesRename}(x, e) := \frac{\text{Prob}(\text{Label}(x) \mid x.val)}{\text{Prob}(e \mid x.val)} \geq 1$$

$\text{BayesRename}(x, e)$  is greater than 1 if  $e \neq \text{Label}(x)$ . Otherwise, it is 1.

Now consider Observation 4. A node  $x \in \mathbf{X}$  is renamed to  $e$  so that the resulting label path would conform to the majority schema  $\mathbf{S}$ . Let  $p_x = \langle x_0, \dots, x \rangle \in \mathbf{X}$ , and

$p_s = \langle s_0, \dots, s \rangle \in \mathbf{S}$  with  $Label(s) = e$  be node paths in  $\mathbf{X}$  and  $\mathbf{S}$  respectively. By Observation 4, the cost of renaming  $x$  to  $e$  can be based on the similarity of the context of  $x$  and  $s$  in the trees. Intuitively, if the neighborhood of  $x$  in  $\mathbf{X}$  and that of  $s$  in  $\mathbf{S}$  is similar, it is more likely that  $x$  and  $s$  are semantically related and hence the lower the cost of renaming should be. The neighborhood of a node consists of its siblings, its descendants and its ancestors.

The sibling neighborhood of  $x$  and  $s$  is compared by the degree of overlap of the concept names of their siblings. Let  $x_1, \dots, x_p$ , and  $s_1, \dots, s_q$  be the siblings of  $x$  and  $s$  respectively. Among them, we have  $Label(x_{i_k}) = Label(s_{j_k})$ ,  $1 \leq k \leq m$ ,  $m \leq Min\{p, q\}$ . A simple formulation for the difference of the sibling neighborhood of  $x, s$  can be:

$$\Delta SiblingNeighbor(x, s) := \frac{1}{4} \left( 2 - \frac{2m}{Min\{p, q\}} + \frac{p-m}{p} + \frac{q-m}{q} \right) \in \mathbf{R}[0, 1]$$

$m$  is the number of sibling nodes in common between  $x$  and  $s$ . The component  $1 - \frac{m}{p}$  ( $1 - \frac{m}{q}$ ) then measures the similarity in sibling neighborhood based on  $x$  ( $s$ ). The component  $\frac{p-m}{p}$  ( $\frac{q-m}{q}$ ) measures the difference in sibling neighborhood based on  $x$  ( $s$ ). Since each of the 4 components ranges from  $\mathbf{R}[0,1]$ , we divide the sum of these components by 4.

The more similar the sibling neighborhood of  $x$  and  $s$  is, the smaller is

$\Delta SiblingNeighbor(x, s)$ . The cost of renaming  $x$  to  $Label(s)$  should be smaller.

The descendant neighborhood of  $x$  and  $s$  is compared by the degree of overlap of the concept names of their descendants. Let  $x_1, \dots, x_p$  be the child nodes of  $x$  and  $s_1, \dots, s_q$  those of  $s$ .  $x_{i_k}$  and  $s_{j_k}$  share the same label,  $1 \leq k \leq m$ . A simple formula-

tion for the difference between the descendant neighborhood of  $x$  and  $s$ , denoted by  $\Delta DescendantNeighbor(x, s)$  can be

$$\Delta DescendantNeighbor(x, s) := \begin{cases} 0 & \text{if } IsLeaf(x) \wedge IsLeaf(s) \wedge (Label(x) = Label(s)) \\ 1 & \text{if } IsLeaf(x) \wedge IsLeaf(s) \wedge (Label(x) \neq Label(s)) \\ \frac{1}{4} \left( \frac{2 \sum_k \Delta DescendantNeighbor(x_{i_k}, s_{j_k})}{Min\{p, q\}} + \frac{p-m}{p} + \frac{q-m}{q} \right) & \text{otherwise} \end{cases}$$

$\Delta DescendantNeighbor$  measures the overlap of the concept names of the children of  $x$  and  $s$ . If two leaf nodes share the same label, their  $\Delta DescendantNeighbor$  is 1. Otherwise, it is zero.  $\Delta DescendantNeighbor$  for internal nodes are defined recursively on  $\Delta DescendantNeighbor$  of their descendants, in a way similar to  $\Delta SiblingNeighbor$ .

The ancestor neighborhood of a node is based on the path from the root of the tree to the node. Based on the Concept Hierarchy Property (Section 1.2.3), higher level concept nodes are refined by lower concept nodes. A higher level concept node is typically more important in describing document structures than a lower level concept node. Thus, the order of the nodes along the path is relevant and a higher weight is assigned to a higher level concept node. Let  $\langle v_1, \dots, v_n \rangle$  be a node path in a tree. The weight of a node  $v_i, 0 \leq i \leq n$  then is

$$PathWeight(v_i) := Depth(v_i)^{-1}$$

Let  $p_x = \langle x_0, \dots, x \rangle \in \mathbf{X}$  and  $p_s = \langle s_0, \dots, s \rangle \in \mathbf{S}$  be node paths from the root to  $x$  and  $s$  in  $\mathbf{X}$  and  $\mathbf{S}$  respectively. Let the longest common subsequence of  $LabelPath(p_x)$

and  $LabelPath(p_s)$  be  $lcs_x = x_{i_1} \dots x_{i_m}$  in  $\mathbf{X}$  and  $lcs_s = s_{j_1} \dots s_{j_m}$  in  $\mathbf{S}$  where  $LabelPath(lcs_x) = LabelPath(lcs_s)$ .

A formulation for the difference of the ancestor neighborhood of  $x$  and  $s$  can be:

$$\begin{aligned} \Delta AncestorNeighbor(x, s) := & \frac{1}{4} \left( 2 - \frac{\sum_{k \in lcs_x} PathWeight(x_k)}{\sum_{k \in p_x} PathWeight(x_k)} - \frac{\sum_{k \in lcs_s} PathWeight(s_k)}{\sum_{k \in p_s} PathWeight(s_k)} \right. \\ & \left. + \frac{\sum_{k \notin lcs_x} PathWeight(x_k)}{\sum_{k \in p_x} PathWeight(x_k)} + \frac{\sum_{k \notin lcs_s} PathWeight(s_k)}{\sum_{k \in p_s} PathWeight(s_k)} \right) \in \mathbf{R}[0, 1] \end{aligned}$$

The formulation of  $\Delta AncestorNeighbor(x, s)$  is similar to  $\Delta SiblingNeighbor$ .

$\sum_{k \in lcs_x} PathWeight(x_k)$  and  $\sum_{k \in lcs_s} PathWeight(s_k)$  play the role of  $m$  in  $\Delta SiblingNeighbor$  while  $\sum_{k \in p_x} PathWeight(x_k)$  and  $\sum_{k \in p_s} PathWeight(s_k)$  play the roles of  $p$  and  $q$  in  $\Delta SiblingNeighbor$  respectively.

Taking both Observation 3 and 4 into account, let us consider a node  $x$  in  $\mathbf{X}$  and the cost of renaming  $x$  to  $e$  so that it conforms to  $\langle root, \dots, s \rangle$  in  $\mathbf{S}$ . The cost of the *Rename* operator can be defined as:

$$\begin{aligned} Cost(Rename, x, e) := & BayesRename(x, e) \left( 1 + \frac{1}{3} (\Delta SiblingNeighbor(x, s) + \right. \\ & \Delta DescendantNeighbor(x, s) + \\ & \left. \Delta AncestorNeighbor(x, s)) \right) \geq 1 \end{aligned}$$

The neighborhood factor ranges from 0 to 1. Since a concept node in  $\mathbf{X}$  is more likely to be semantically related to a node with the same concept name in  $\mathbf{S}$ , the *Rename* operator can be assigned a higher cost than that of the *Insert* and the *Delete* operators, which do not change the label of the node. Therefore, we add some base cost to the neighborhood factor, e.g., a base cost of 1.

We define the cost of the *Rename* operator as a linear formulation because of its simplicity. Other non-linear formulations can also be explored.

The following example illustrates this idea. Consider the `LOCALE` node  $x$  in the XML document and the `ORGANIZATION` node  $s$  in the majority schema.

<p>An XML document</p> <pre>&lt;RESUME&gt;   &lt;EDUCATION&gt;     &lt;LOCALE&gt;       &lt;DATE/&gt;       &lt;DEGREE/&gt;     &lt;/LOCALE&gt;     &lt;COURSES/&gt;   &lt;/EDUCATION&gt; &lt;/RESUME&gt;</pre>	<p>A majority schema</p> <pre>&lt;RESUME&gt;   &lt;EDUCATION&gt;     &lt;ORGANIZATION&gt;       &lt;DATE/&gt;       &lt;DEGREE/&gt;       &lt;GPA/&gt;     &lt;/ORGANIZATION&gt;   &lt;/EDUCATION&gt; &lt;/RESUME&gt;</pre>
---	---

Let us consider the *Rename* operator. Suppose the textual content of  $x$  can be associated with the concept name `LOCALE` with a relative probability 0.2 which is the highest among all concept names. The concept name with the next highest probability is `ORGANIZATION` with a relative probability 0.22. Then, we have

$$\text{BayesRename}(x, \text{ORGANIZATION}) = \frac{0.22}{0.2} = 1.1$$

Now consider the neighborhood of  $x$  and  $s$ . The sibling of  $x$  is the `COURSES` node.  $s$  does not have any sibling. Thus,

$$\Delta \text{SiblingNeighbor}(x, s) = \frac{1}{4} \left( 2 - 2 \times \frac{1}{1} + \frac{1}{2} + \frac{0}{1} \right) = \frac{1}{8}$$

The descendants **DATE** and **DEGREE** of  $s$  share the same label with those of  $x$ , but not the descendant **GPA**. Hence,

$$\Delta_{DescendantNeighbor}(x, s) = \frac{1}{4}(2 \times \frac{0}{2} + \frac{0}{2} + \frac{1}{3}) = \frac{1}{12}$$

The ancestor label paths of  $x$  and  $s$  are  $\langle \text{RESUME EDUCATION LOCALE} \rangle$  and  $\langle \text{RESUME EDUCATION ORGANIZATION} \rangle$  respectively. Their longest common subsequence is  $\langle \text{RESUME EDUCATION} \rangle$ . Then, we have

$$\Delta_{AncestorNeighbor}(x, s) = \frac{1}{4}(2 - 2 \times \frac{9}{11} + \frac{2}{11} + \frac{2}{11}) = \frac{2}{11}$$

The cost of renaming  $x$  to  $s$  is given by

$$Rename(x, s) = 1.1 \times (1 + \frac{1}{3}(\frac{1}{8} + \frac{1}{12} + \frac{2}{11})) = 1.24$$

Let us consider the *Insert* and *Delete* operators. Suppose the support ratio of  $x$  and  $s$  are 0.4 and 0.7 respectively. The cost of deleting  $x$  is thus 0.4. The cost of inserting a node labeled with **ORGANIZATION** conforming to  $s$  is  $1-0.7=0.3$ .

To transform the XML document to the majority schema, we either (1) delete  $x$  and insert  $s$ , or (2) rename  $x$  to  $s$ . The total cost of the former is  $0.4+0.3=0.7$  while the cost of the latter is 1.24.

### 5.2.1.3 Refining Semantic Mapping

In the previous section, we described how to use knowledge on the documents and the majority schema to customize the cost model of the tree edit distance algorithm



to derive a semantic mapping. However, the semantic mapping computed requires further refinement. This is because the tree edit distance algorithm aims to minimize the cost of the edit sequence rather than to maximize the semantic mapping between nodes in the document tree and the majority schema. Consider the following trees:

**Example 5.5:**

A tree edit distance algorithm optimizes the cost of transformation, not the number of semantically related nodes identified.

Document tree	Tree schema
<A>	<A>
<B/>	<B/>
<C/>	<D/>
</A>	</A>

The tree edit distance algorithm may suggest mapping A and B in the document tree to those in the majority schema, deleting C in the document tree and inserting D into the majority schema. The edit sequence may be of the minimum cost among all others. The edit sequence of renaming C to D may be of higher cost. However, if the cost is not significantly higher, we prefer to rename C to D so that all nodes in the document tree have semantically related nodes in the majority schema. In this way, we maximize the number of nodes in the document tree with semantically related nodes in the majority schema given by the semantic mapping.

The function  $\nu$  then is:

$$\nu(A) = A$$

$$\nu(B) = B$$

$$\nu(C) = D$$

Moreover, the tree edit distance algorithm only considers mappings that preserve ancestor-descendant relationship. Suppose a concept node  $A$  is an ancestor of another node  $B$  in the document tree and  $A'$  and  $B'$  are the nodes in the majority schema that they are mapped to. Then,  $A'$  cannot be a descendant of  $B'$ . For example, in the following fragment, the tree edit distance would suggest deleting  $B$  from  $A$  in the document tree and inserting another node of label  $B$  as the parent of  $A$ , instead of mapping  $A$  and  $B$  in the document tree to those in the majority schema directly.

**Example 5.6:**

The ancestor-descendant relationship is always preserved in a tree edit distance mapping.

Document tree	Tree schema
<A>	<B>
<B/>	<A/>
</A>	</B>

Therefore, we further refine the semantic mapping computed by the tree edit distance algorithm to maximize the semantic mapping. On one hand, we would like to refine the semantic mapping of a deleted node  $x$  in the document tree to some node with the same label in the majority schema. On the other hand, it is not desirable to map

$x$  to a node  $s$  that is not similar in terms of the neighborhood. In the example in Section 5.1.1, **DATE** under **EXPERIENCE** in the document tree should not be mapped to **DATE** under **EDUCATION** in the majority schema.

The idea is to use a *boundary box* (Section 4.2.3.1) to define the extent to which we consider refining the semantic mapping. To recap, the reference ancestors, reference descendants and the reference siblings of a node in a document tree or majority schema define the boundary box of the node. We consider the boundary box of a deleted node in the document tree. By the definition of boundary box, the nodes in the boundary box are semantically related to some nodes in the majority schema. The neighborhood of those nodes in the majority schema gives us the domain to match an inserted node of the same concept name as the deleted node. If a match is found, the deleted node in the document tree is then semantically related to this inserted node in the majority schema.

The function *RefineSemanticMapping* realizes this idea and is illustrated by Figure 5.1. It takes a document tree  $X$ , a majority schema  $S$ , and a semantic mapping from  $X$  to  $S$  that is to be refined. It searches each deleted node  $x \in X$  to locate an inserted node  $s \in S$  that is semantically related to  $s$ . The boundary box of each reference sibling of  $x$  is searched for a possible match of an inserted node in  $S$  with the same label. The process is repeated for the reference descendants and the reference ancestor of  $x$ . The first match  $s$  is the semantically related node for  $x$ . The semantic mapping of  $x$  is updated to  $\nu(x) = s$ .

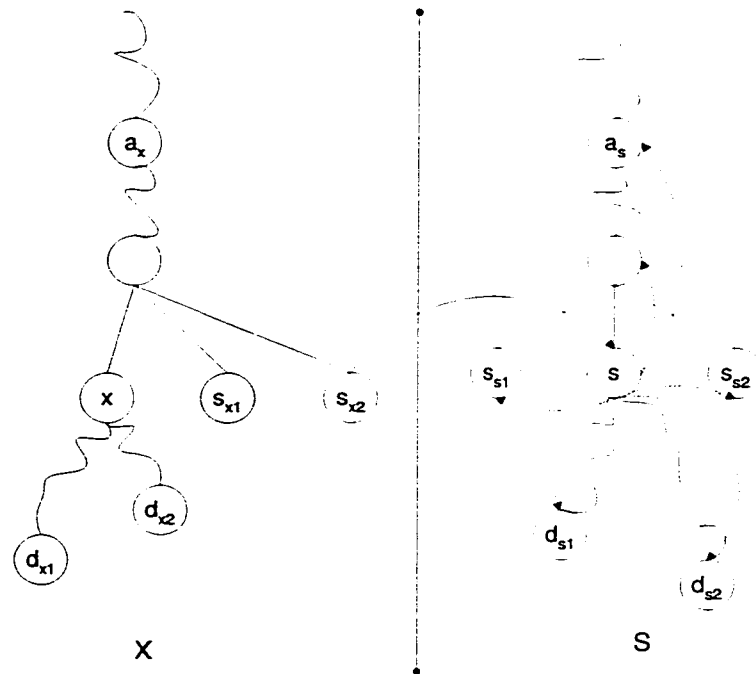


Figure 5.1: Refine semantic mapping

The boundary box of  $x$  is used to search for nodes in  $S$  that can be semantically related to  $x$ . This figure shows that the search is based on its reference sibling  $s_{x1}$ . The semantically related node of  $s_{x1}$  in  $S$  is  $s$ . The neighborhood of  $s$  - all nodes up to its reference ancestor  $a_s$ , its siblings  $s_{s1}, s_{s2}$ , all nodes along the paths to its reference descendants  $d_{s1}, d_{s2}$  - are searched for an inserted node labeled with  $Label(x)$ . If a match is located, the semantic mapping is refined to map  $x$  to this node.

**Algorithm 5.17: RefineSemanticMapping**

*Function*  $RefineSemanticMapping(X, S, \nu) : V \rightarrow V$

**begin**

**for each**  $x \in X$  s.t.  $\nu(x) = \perp$  **do**

*(Search based on the reference siblings of  $x$ )*

**for each**  $s_x \in RefSiblings(x, \nu)$  **do**

$s = MatchLabel(\nu(s_x), Label(x), S, \nu)$

**if**  $s \neq \perp$

**then**

$\nu(x) = s$

*(try next  $s_x$ )*

**continue**

**fi**

**od**

*(Search based on the descendants of  $x$ )*

**for each**  $d_x \in Descendants(x)$  in a breadth first search manner **do**

$s = MatchLabel(\nu(d_x), Label(x), S, \nu)$

**if**  $s \neq \perp$

**then**

$\nu(x) = s$

**continue**

**fi**

**od**

```

    (Search based on the reference ancestor of x)
     $a_x = \text{RefAncestor}(x, \nu)$ 
     $s = \text{MatchLabel}(\nu(a_x), \text{Label}(x), \mathbf{S}, \nu)$ 
    if  $s \neq \perp$ 
        then
             $\nu(x) = s$ 
            continue
        fi
    od
    return  $\nu$ 
end

```

Searching for a match of an inserted node of the same label is implemented by the function *MatchLabel*. It searches the neighborhood of  $s$  in  $\mathbf{S}$  for an inserted node with label  $e$ . It searches its siblings, its descendants (in a breadth first manner), and its ancestors up to its reference ancestor. The first match  $m$  is returned.

**Algorithm 5.18: MatchLabel**

*Function MatchLabel*( $s, e, \mathbf{S}, \nu$ ) :  $\mathbf{V}$

```

begin
     $m = \perp$ 
    (Search based on the siblings of s)
    for each  $s_i \in \text{Siblings}(s)$  do

```

```

if  $Label(s_i) = e \wedge \nu^{-1}(s) = \perp$ 
  then
     $m = s_i$ 
  return
fi
od
(Search based on the reference descendants of s)
for each  $s_n \in RefDescendants(s, \nu), s = s_0, \langle s_0, s_1, \dots, s_n \rangle \in S$  do
  for  $i = 0$  to  $n$  do
    if  $Label(s_i) = e \wedge \nu^{-1}(s) = \perp$ 
      then
         $m = s_i$ 
      return
    fi
  od
od
(Search based on the reference ancestor of s)
 $s_0 = RefAncestor(s, \nu), \langle s_0, s_1, \dots, s_n, s \rangle \in S$ 
for  $i = 1$  to  $n$  do
  if  $Label(s_i) = e \wedge \nu^{-1}(s) = \perp$ 
    then
       $m = s_i$ 
    return

```

```
fi  
od  
return m  
end
```

## 5.2.2 Transformation

In this section, we describe how the Document Transformer transforms a document tree to conform to the majority schema based on the semantic mapping derived in Section 5.2.1. An example of such semantic mapping is given in Section 5.2.1.3.

The idea is to construct a new document tree  $M$  (transformed tree) for the input document  $X$  using the majority schema as a template.<sup>2</sup> This is realized by the function *Transform*.

We use the notation  $x$ ,  $s$  and  $m$  to denote a node in  $X$ ,  $S$  and  $M$  respectively. The ancestor of a node  $n$  is denoted by  $a_n$ . For example,  $a_x$  is an ancestor of  $x$  and  $a_s$  is an ancestor of  $s$ . The parent of a node  $n$  is denoted by  $p_n$ , its sibling by  $s_n$ . We loosely use the notation  $\{n\}$  to denote a set of nodes and  $n$  a node in this set.

---

<sup>2</sup> $X$  is *not* the bisimulated tree in Sections 5.2.1, but the XML document itself. The bisimulated tree is compared against the majority schema to compute the semantic mapping. The document tree is used for the transformation process based on the semantic mapping computed.



In the function *Transform*, nodes in the majority schema  $S$  are considered in a top-down manner. Suppose sibling nodes in the majority schema are sorted according to their relative position in the associated DTD (Section 4.2.4). Nodes in  $S$  are considered according to this order. After initializing the transformed document  $M$ , the function *Transform* calls the function *Construct* which creates nodes for  $M$  using  $S$  as a template. After *Construct* terminates, the textual content of a node  $x$  in  $X$  without any semantically related node in  $S$  is preserved by propagating  $x.val$  to its ancestor node it details information about. This is described in the last **for** loop. Its reference ancestor  $a_x$  is located.  $a_m$  is the node in  $M$  which conforms to  $a_s$ .  $a_m$  keeps the textual content of  $x$  (Figure 5.2).

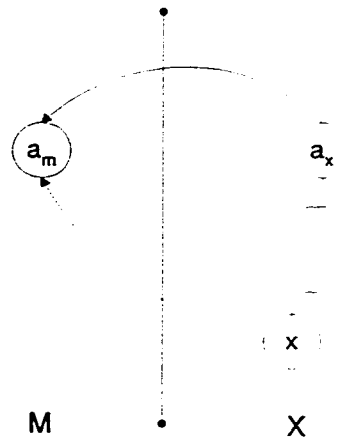


Figure 5.2: Textual content of "deleted" nodes is kept in its reference ancestor in the transformed document.

**Algorithm 5.19: Transform**

*Function Transform*( $\nu, X, S$ ) :  $X$

**begin**

*(Construct the transformed tree M)*

$M = \text{new XML document}$

$M.\text{root} = \text{Create}(\text{Label}(S.\text{root}))$

*Construct*( $\nu, X, S, M.\text{root}$ )

*(Textual content of nodes in X without semantically related nodes is propagated to their parents)*

**for each**  $x \in X, \nu(x) = \perp$  **do**

$a_x = \text{RefAncestor}(x, \nu)$

$a_m = \theta(a_x)$

$a_m.\text{val} = a_m.\text{val} \circ x.\text{val}$

**od**

**return**  $M$

**end**

In the function *Construct*, a node  $s$  in the tree schema has a set of nodes  $\{x\}$  in the document tree  $X$  to which these nodes are semantically related. We would like to create a new node  $m$  in  $M$  for  $x$ . In doing so, we need to locate the node in  $M$  which should be the parent of  $m$ . This is achieved by locating the reference ancestor,  $a_s$ , of  $s$ . There are nodes in  $\{a_x\}$  that are semantically related to  $a_s$ . Since *Transform* is applied in a top-down manner, new nodes are already created for  $\{a_x\}$  in  $M$ . We

pick the node in  $\{a_x\}$  that the node  $x$  details information about (as implemented by the function *PickNearestAncestor*). The node in  $M$  for  $a_x$  is  $a_m$ . Since there may be nodes in  $S$  with no semantically related nodes in  $X$  (inserted nodes),  $a_s$  may not be a parent of  $s$ .  $\langle a_s, a_{s_0}, \dots, a_{s_n}, s \rangle$  is the node path from  $a_s$  to  $s$  with  $a_{s_i}$  being the inserted nodes in  $S$ . Since  $M$  has to conform to  $S$ , we have to append  $m$  to  $a_m$  according to the node path  $\langle a_{s_0}, \dots, a_{s_n} \rangle$ . New nodes  $\langle a_{m_0}, \dots, a_{m_n} \rangle$  are created for this node path in  $M$ . Care is taken to ensure  $a_{m_i}$  is not created twice. If there are inserted nodes from  $a_s$  to  $s$ ,  $m$  is appended as a child of  $a_{m_n}$ , otherwise, as a child to  $a_m$ . Let  $p_m$  be the parent of  $m$  in  $M$ ,  $p_s$  the node in  $S$  to which  $p_m$  conforms.  $m$  is inserted into  $M$  based on the content model of  $p_s$ . If the content model of  $p_s$  is not repetitive ("\*") and  $p_m$  already has a child with the same label as  $m$ , called  $s_m$ , the textual content of  $m$  is kept in  $s_m$  by appending  $m.val$  to  $s_m.val$ . Otherwise,  $m$  is appended as a new child of  $p_m$ . We maintain a mapping  $\theta : X.V \rightarrow M.V$  to record which node in  $M$  is created for each node in  $X$ .

**Algorithm 5.20: Construct**

*Function Construct*( $\nu, X, S, u$ ) :  $X$

**begin**

**for each**  $s \in Children(u)$  **in order do**

$\{x\} = \nu^{-1}(s)$

$a_s = RefAncestor(s, \nu), \langle a_s, a_{s_1}, \dots, a_{s_n}, s \rangle \in S$

$\{a_x\} = RefAncestor(a_s, \nu)$

**for each**  $x$  **do**

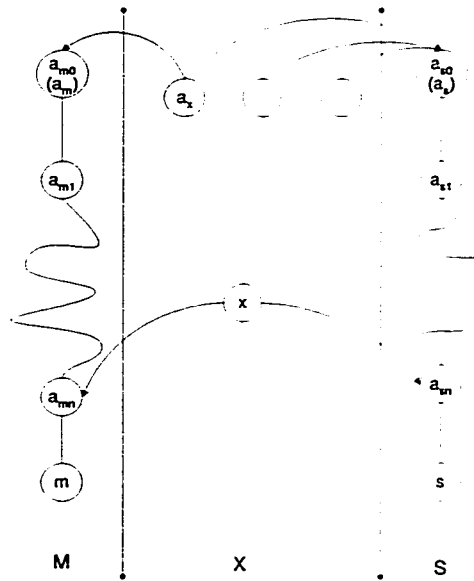


Figure 5.3: Construct - This figure illustrates how to construct a node  $m$  in the transformed document  $M$  for the node  $x$  in  $X$ . To do so, we need to locate the parent of  $m$  in  $M$ . The semantically related node for  $x$  in  $S$  is  $s$ . The reference ancestor of  $s$  is  $a_s$ . The set of nodes that are semantically related to  $a_s$  are  $\{a_x\}$ , one of which is picked whose corresponding node  $a_m$  in  $M$  is the new ancestor (parent) of  $m$ . The path  $\langle a_{s0}, \dots, a_{sn} \rangle$  in  $S$  are nodes without semantically related nodes in  $X$ . The same path is inserted into  $M$  as  $m$  is appended as a child of  $a_m$ .

*(Pick one among  $\{a_x\}$  that is nearest to  $x$ )*  
 $a_x = \text{PickNearestAncestor}(x, \{a_x\}, X)$   
 $a_m = \theta(a_x), a_{m_0} = a_m$   
*(Append path  $p_m = \langle a_{m_1}, \dots, a_{m_n} \rangle$  to  $a_m$*   
*in  $M$  according to  $\langle a_{s_1}, \dots, a_{s_n} \rangle$ ,*  
*if  $p_m$  does not exists)*  
**for  $i$  to  $n$  do**  
     **if  $\neg \exists a_{m_i} \in \text{Children}: (a_{m_{i-1}}) \wedge \text{Label}(a_{m_i}) = \text{Label}(a_{s_i})$**   
         **then**  
              $a_{m_i} = \text{Create}(\text{Label}(a_{s_i}))$   
              $\text{AppendChild}(a_{m_{i-1}}, a_{m_i})$   
         **fi**  
     **od**  
     *(Create a new node  $m$  for  $x$ .*  
     *Add  $m$  to  $M$  according to the content model of  $p_s$ .)*  
     *(Check if there are nodes between  $a_m$  and  $m$  that are inserted nodes.*  
     *If so, the last node along this path  $a_{m_n}$  is the parent of  $m$ .)*  
     **if  $a_{s_n} \neq \perp$**   
         **then**  
              $p_m = a_{m_n}$   
              $p_s = a_{s_n}$   
         **else**  
              $p_m = a_m$

$$p_s = a_s$$
**fi**

**if**  $p_s.content \neq "*" \wedge s_m \in Children(p_m) \wedge (Label(s_m) = Label(m))$

**then**

*(There is a node  $s_m$  with label  $Label(x)$  and the content model of  $p_s$  does not allow more than one concept node labeled with  $Label(x)$ .*

*Do not create new node  $m$  for  $x$ .*

*Store the textual content of  $x$  in  $s_m$ .*

$$s_m.val = s_m.val \circ x.val$$

$$\theta(x) = s_m$$
**else**

*(Create a new node  $m$  for  $x$  in  $M$ .)*

$$m = Create(Label(s))$$

$$m.val = x.val$$

$$AppendChild(p_m, m)$$

$$\theta(x) = m$$
**fi****od****od**

*(Recursively consider each child node of  $u$  in a breadth first search manner according to the order among siblings)*

**for each**  $s \in Children(u)$  **in order** **do**

```

    M = Construct( $\nu$ , X, S, s)
  od
  return M
end

```

The function *PickNearestAncestor* implements the function to choose the node in  $\{a_1, \dots, a_n\}$  that a node  $x$  is likely to detail information about. This is based on the Concept Hierarchy Property in Section 1.2.3. Therefore, *PickNearestAncestor* picks the node from  $\{a_1, \dots, a_n\}$  that is closest to  $x$ , i.e. their least common ancestor has the largest depth.

**Algorithm 5.21: PickNearestAncestor**

*Function* *PickNearestAncestor*( $x, \{a_1, \dots, a_n\}, X$ ) : V

```

begin
  a = a1
  md = 0
  for i = 1 to n do
    lcs = LeastCommonAncestor(x, ai)
    if Depth(lcs) > md
      then
        a = ai
        md = Depth(lcs)
    fi

```

**od**  
**return  $a$**   
**end**

Note that there may be trailing label paths in  $S$  without any semantically related node paths in  $X$ , e.g.,  $\langle p_1, \dots, p_m, q_1, \dots, q_n \rangle \in S$ ,  $\nu(s_i) = \perp$ . The empty node path  $\langle q_1, \dots, q_n \rangle$  is not appended to  $M$  since there is no textual content along the path. These nodes are ignored.

## 5.3 Evaluation

In Section 5.3.1, we prove the correctness of the transformation process. In Section 5.3.2, we analyze the time complexity of the transformation process, and consider some practical situations.

### 5.3.1 Correctness

We prove that the algorithm *Transform* correctly computes  $\theta$  for  $DTP[X, S]$  (Section 5.1.2).



Let us use the following notations:

$p = \langle root, \dots, a_m, a_{m_1}, \dots, a_{m_n}, m \rangle$ : path from the root to  $m$

$q = \langle root, \dots, s_m, a_{s_1}, \dots, a_{s_n}, s \rangle$ : path from the root to  $s$

$ip = \langle a_{m_1}, \dots, a_{m_n}, m \rangle$ : path from the root to  $m$  excluding the root

$iq = \langle s_{m_1}, \dots, a_{s_n}, s \rangle$ : path from the root to  $s$  excluding the root

$ap = \langle root, \dots, a_m \rangle$ : path from the root to  $m$  excluding  $m$

$aq = \langle root, \dots, a_s \rangle$ : path from the root to  $s$  excluding  $s$

**Lemma 5.1 (Conformance Property).** *Given an XML document  $X$ , a majority schema  $S$ , and a semantic mapping  $\nu$  from  $X$  to  $S$ . The mapping  $\theta$  computed by  $Transform(X, S, \nu)$  satisfies the Conformance Property.*

**Proof:** We need to show that the transformed tree  $M$  conforms to  $S$  under the mapping  $\mu$  that is constructively defined. There are only two occasions where a new node is inserted into  $M$  in *Construct*. The first is at appending  $a_{m_i}$  as a child to  $a_{m_{i-1}}$ . Define  $\mu$  as  $\mu(a_{m_i}) = s_{m_i}$ . The second place is at appending  $m$  as a child to  $p_m$ . Define  $\mu(m) = p_s$ .

We prove by induction based on the recursion level of *Construct* on  $S$ . Initialization in *Transform* sets  $\mu(M.root) = S.root$ . Suppose  $M$  conforms to  $S$  up to some recursion of *Construct*.

Consider the next recursion. We need to show that  $M$  still conforms to  $S$  after appending  $ip$  to  $a_m$ .  $a_m$  is appended to  $M$  in some previous recursion. By induction hypothesis, the node path  $ap$  conforms to some node path  $aq$  in  $S$ .

Consider a new node path  $p$  in  $M$ . We have to show that  $p$  conforms to some node path in  $S$ . Consider  $q$ . By the definition of  $\mu$ ,  $LabelPath(p) = LabelPath(q)$ . Care is taken not to create a node along  $\langle a_{m_0}, \dots, a_{m_n} \rangle$  again if a node of the same label has been created in some previous recursion. Hence, there is no sibling node of  $a_{m_i}$  with the same label. Since the content model of any node in the associated DTD of a majority schema is either "\*" or "?" or  $\epsilon$ ,  $a_{m_i}$  satisfies the content model of  $s_{m_i}$ . Therefore,  $p$  conforms to  $q$  in  $S$ .  $\square$

**Lemma 5.2 (Content Preservation Property).** *Given an XML document  $X$ , a majority schema  $S$ , and a semantic mapping  $\nu$  from  $X$  to  $S$ . The mapping  $\theta$  computed by  $Transform(X, S, \nu)$  satisfies the Content Preservation Property in  $DTP[X, S]$ .*

**Proof:** We show that the textual content of a node  $x$  in  $X$  that is semantically related to some node  $s$  in  $S$  is stored in some node  $m$  in  $M$  conforming to  $s$ .

First, we show that all the variable nodes in *Construct* actually exist. Suppose  $x$  is semantically related to  $s$ . Consider the recursion of *Construct* on  $s$ . Let us follow the program line by line. Since the root of a document tree is always mapped to the root of the majority schema,  $a_s$  and  $\{a_x\}$  exist. Since *PickNearestAncestor* always returns one element from  $\{a_x\}$ ,  $a_x$  exists. *Construct* is applied to  $S$  in a top-down

manner, so at least one node is created for  $a_s$  in  $\mathbb{M}$  in some previous recursion. Hence  $a_m$  exists.

Next, we show that the textual content  $x.val$  of the node  $x$  is stored in some node  $m$  in  $\mathbb{M}$  conforming to  $s$ . The textual content of  $x$  is either appended to  $ap \oplus s_m$  or  $ap \oplus m$ . By Lemma 5.1,  $ap \oplus m$  conforms to  $S$ . Since  $Label(s_m) = Label(m)$ ,  $ap \oplus s_m$  also conforms to  $S$ . In both cases, both node paths conform to  $S$ .  $\square$

**Lemma 5.3 (No Information Loss Property).** *Given an XML document  $X$ , a majority schema  $S$ , and a semantic mapping  $\nu$  from  $X$  to  $S$ . The mapping  $\theta$  computed by  $Transform(X, S, \nu)$  satisfies the No Information Loss Property in  $DTP[X, S]$ .*

**Proof:** We need to show that the textual contents of deleted nodes in  $X$  are stored in higher level information objects they detail information about. After *Construct* terminates, the textual content of a node  $x$  in  $X$  without semantically related node in  $S$  is propagated to  $a_m$  where  $a_x = RefAncestor(x) \wedge a_m = \theta(a_x)$ . We show  $a_m$  and  $a_x$  exist. Since the root of the document tree is always semantically related to that of the majority schema,  $a_x$  exists. Let  $a_s$  be  $\nu(a_x)$ . Since  $a_s$  has some semantically related nodes in  $X$ , at least one node is created accordingly in  $\mathbb{M}$  by *Construct* on  $a_s$ . Therefore,  $a_m$  exists.  $\square$

By Lemma 5.1, 5.2 and 5.3, we have

**Theorem 5.1 (Correctness).** *Given an XML document  $X$ , a majority schema  $S$ , and a semantic mapping  $\nu$  from  $X$  to  $S$ .  $\text{Transform}(X, S, \nu)$  correctly computes a transformation mapping  $\theta$  to  $\mathcal{DTP}[X, S]$ .  $\square$*

Let us comment on the relationship of the transformed document to the DTD derived from the majority schema (Section 4.2.4). The difference between a majority schema and its DTD lies in that the latter encodes content model of an element. Order and multiplicity are considered whereas choice and grouping are not modeled in the DTD. Since nodes are created in the transformed tree according to the relative order of the corresponding nodes in the majority schema, the order content model of the DTD is observed in the transformed document. The multiplicity information is considered in conformance, hence the multiplicity content model in the DTD is also observed. Therefore, the transformed tree is valid with respect to the DTD.

### 5.3.2 Computational Complexity

In this section, we give an analytical study on the computational complexity of the document transformation process. Section 5.3.2.1 analyzes the time complexity of deriving a semantic mapping (computing the tree edit distance, customizing the cost model and refining the semantic mapping) while Section 5.3.2.2 analyzes the time complexity of the transformation process.

### 5.3.2.1 Deriving Semantic Mapping

**Computing the Tree Edit Distance:** As discussed in Chapter 4, the time complexity of the tree edit distance algorithm presented in [Zha96] for transforming  $X$  to  $S$  is

$$O(|X| |S| (Degree(X) + Degree(S)) \log(Degree(X) + Degree(S)))$$

**Customizing Cost Model:** The cost model for the tree edit distance algorithm proposed in Section 5.2.1.2 is not on concept names, but on concept nodes. We can first compute the costs of the edit operators on the concept nodes and store the costs in a hash table. Running the tree edit distance algorithm would then still take constant lookup time of the cost of the operations.

Building such a hash table, however, can be costly. Computing the cost of *Insert* and *Delete* requires looking up the support of each node. Suppose such information is stored with each node (e.g., storing the support of a node in an attribute), then computing the costs takes constant time. This amounts to  $O(|X| + |S|)$  time in total. Computing *BayesRename* for all nodes takes  $O(|E| |X|)$  time. The most costly expense is computing  $\Delta SiblingNeighbor$ ,  $\Delta DescendantNeighbor$  and  $\Delta AncestorNeighbor$ . Computing  $\Delta SiblingNeighbor$  takes  $O((Deg(X) + Deg(S)) |S| |X|)$  time for all pair of subtrees in  $X$  and  $S$  while  $\Delta AncestorNeighbor$  takes  $O((Depth(X) + Depth(S)) |S| |X|)$  time. Computing  $\Delta DescendantNeighbor$  is most expensive. A naïve implementation takes time exponential to the depth of  $X$  and  $S$ . However, we notice that  $\Delta DescendantNeighbor(x, s)$

depends on  $\Delta DescendantNeighbor(d_x, d_s)$  where  $d_x$  and  $d_s$  are some descendant nodes of  $x$  and  $s$ . We can therefore speed up the computation by using dynamic programming. The idea is to store the cost of  $\Delta DescendantNeighbor(x, s)$  in a two-dimensional matrix and to compute  $\Delta DescendantNeighbor$  from nodes bottom-up. In computing  $\Delta DescendantNeighbor(x, s)$ , we can then lookup the cost of  $\Delta DescendantNeighbor(d_x, d_s)$  from the matrix. Computing the overlap of their child nodes takes  $O(Deg(X) + Deg(S))$  time. There are  $|X| |S|$  entries in the matrix. In total, it takes  $O((Deg(X) + Deg(S)) |X| |S|)$  time. Summing up, the time it takes to compute the customized cost model is

$$O(|S| + |E| |X| + (Deg(X) + Deg(S) + Depth(X) + Depth(S)) |S| |X|)$$

**Refining Semantic Mapping:** Let us consider the time complexity of the function *RefineSemanticMapping*. Let  $ml$  be the time it takes for each call to the function *MatchLabel*. For each node  $x$ , *RefineSemanticMapping* searches its siblings, descendants and ancestors. This is bound by  $Deg(X)$ ,  $Depth(X)$  and  $Deg(X)^{k_x}$  respectively where  $k_x$  is the number of levels of the descendants of  $x$  searched which is bound by  $Depth(X)$ . The total time then is  $O(ml(Deg(X) + Depth(X) + Deg(X)^{k_x}) |X|)$ .

Consider the function *MatchLabel*. Similarly, it searches the siblings, descendants and ancestors of  $s$ . The time for each call is again  $O(Deg(S) + Depth(S) + Deg(S)^{k_s})$  where  $k_s$  the number of levels of the descendants of  $s$  searched.

The time complexity of *RefineSemanticMapping* is thus:

$$O((Deg(S) + Depth(S) + Deg(S)^{k_s})(Deg(X) + Depth(X) + Deg(X)^{k_x}) | X |)$$

This is exponential to the depth of the document tree and the majority schema (due to searching the descendants of  $x$  and  $s$ ), which is not desirable in practice. We can cut off the search of descendants up to certain constant level or even limit the search to the immediate children of  $x$  and  $s$  in practice. The justification is that nodes that are several levels away are probably not semantically related. Under this simplification, *RefineSemanticMapping* takes time:

$$O((Deg(S) + Depth(S))(Deg(X) + Depth(X)) | X |)$$

**Theorem 5.2 (Time Complexity of Deriving Semantic Mapping).** *Given an XML document  $X$  and a majority schema  $S$ . The time complexity to derive a semantic mapping from  $X$  to  $S$  is:*

$$O(| S | + (| E | + (Deg(S) + Depth(S))(Deg(X) + Depth(X))) | X | +$$

$$((Deg(X) + Deg(S)) \log((Deg(X) + Deg(S))) + Deg(X) + Deg(S) + Depth(X) + Depth(S)) | S || X |)$$

which is proportional to  $| S || X |$ . □

### 5.3.2.2 Transformation

Consider the complexity of the function *Transform*. As in Chapter 3, we assume a unit cost model for operations on an XML document. The mappings  $\nu$ ,  $\mu$  and  $\theta$  can be hashed so that lookups take constant time.

Consider the function *Construct*. It is applied to each node  $s$  in  $S$  once. Let us consider the cost over all the recursions. While the number of elements in  $\{x\}$  differs at each recursion, over all the recursions, we have the total number of  $x$  being bound by  $|X|$  since  $\nu$  is a function ( $x$  can be mapped to at most one node  $s$ ). Therefore, looking up  $\{x\}$  costs at most  $|X|$ . Locating the reference ancestor of  $s$  takes  $O(\text{Depth}(S))$  time which amounts to  $O(\text{Depth}(S)|S|)$  time over all recursions. Similarly, looking up  $\{a_x\}$  costs at most  $|X|$  over all recursions.

Now consider the inner loop on  $x$ . Let  $pna$  be the time it takes for each call of *PickNearestAncestor*. Appending  $\langle a_m, \dots, a_{m_n} \rangle$  takes  $n\text{Deg}(X)$  time which is bound by  $\text{Depth}(S)\text{Deg}(X)$ . Appending  $m$  to  $p_m$  takes time  $O(\text{Deg}(X))$ . Since there are at most  $|X|$  such  $x$ , we have the total time of the inner loop being bound by  $O(|X|(pna + \text{Depth}(S)\text{Deg}(X)))$ . In sum, *Construct* takes a total of  $O(\text{Depth}(S)S + |X|(pna + \text{Depth}(S)\text{Deg}(X)))$  time.

Next, consider the time for preserving the textual content of nodes in  $X$  with no semantically related node in  $S$ . For each such node  $x$ , we locate its reference ancestor which takes  $O(\text{Depth}(X))$  time. Appending  $x.val$  accordingly takes constant time. Since there are at most  $|X|$  such nodes, the total time is  $O(\text{Depth}(X)|X|)$ .

Consider the function *PickNearestAncestor*. Locating the least common ancestor of  $x$  and  $a_i$  takes  $O(\text{Depth}(X))$  time, which amounts to  $O(n\text{Depth}(X))$  time in total. A pessimistic estimate of the upper bound of  $n$  is  $|X|$ . This is the case when all nodes in  $X$  are semantically related to one node in  $S$ .



Summing up, we have

**Theorem 5.3 (Time Complexity of Transform).** *Given an XML document  $X$ , a majority schema  $S$  and a semantic mapping  $\nu$ . Let  $pna$  be the time complexity of the function  $PickNearestAncestor$ . The time complexity of  $Transform(\nu, M, S)$  is*

$$O(\text{Depth}(S) | S | + (pna + \text{Deg}(X)\text{Depth}(S) + \text{Depth}(X)) | X |)$$

*Taking the upper bound of the time complexity of  $pna$ , this becomes:*

$$O(\text{Depth}(S) | S | + (\text{Deg}(X)\text{Depth}(S) + \text{Depth}(X)) | X | + \text{Depth}(X) | X |^2)$$

*which is proportional to  $| X |^2$ .*

□

### 5.3.2.3 Practical Situations

We consider the time complexity of the document transformation process under some practical scenarios.

In practice, we make the following simplifications:

1. Since the majority schema  $S$  describes an XML document  $X$ , we can expect the depth of  $X$  to be fairly similar to that of  $S$ , i.e. we approximate  $\text{Depth}(X)$  by  $\text{Depth}(S)$ .
2. Since there are no repetitive label paths in  $S$ , we assume  $\text{Deg}(S)$  to be a constant.

3. Since a majority schema describes the prevalent structures found among topic specific documents, we assume it is smaller in size than the documents themselves, i.e.  $|S| \leq |X|$ .
4. The set of concept names for the topic is fairly constant. Hence, we assume  $|E|$  to be a constant.

**Corollary 5.1 (Time Complexity Under Practical Situation).** *Under simplifications made for practical situations, the time complexity of deriving a semantic mapping from  $X$  to  $S$  is reduced to:*

$$O(|S| + Depth(S)^2 |X| + (Deg(X) \log Deg(X) + Depth(S) + Deg(X)) |S| |X|)$$

*The time complexity of the transformation process is:*

$$O(Depth(S) |S| + Deg(X) Depth(S) |X| + Depth(X) |X|^2)$$

□

Now, let us consider two extreme scenarios of the properties on  $X$ . At one extreme,  $X$  is very deep in its tree structure. We can assume its fan-out to be relatively small, i.e.  $Deg(X)$  is a constant. In this case, the time it takes to derive a semantic mapping is:

$$O(Depth(S)^2 |X| + Depth(S) |S| |X|)$$

The time to transform  $X$  is:

$$O(Depth(S) |X|^2)$$

At the other extreme,  $X$  is very flat in its tree structure. We can assume its depth to be fairly small, i.e.  $Depth(X)$  is a constant. In this case, the time it takes to derive a semantic mapping is:

$$O((1 + \log Deg(X)) Deg(X) | S || X |)$$

The time to transform  $X$  is:

$$O(Deg(X) | X |^2)$$

## 5.4 Related Work

The Document Transformer is related to the research area of schema integration. In this section, we first describe the schema integration problem and give a taxonomy of existing schema integration approaches. Then we describe where our approach stands with respect to these dimensions, and compare our approach with existing ones.

### 5.4.1 Existing Approaches

The schema integration problem is to integrate local schemas, possibly in different data models, into an integrated schema. There are four major tasks: (1) Schema translation: local schemas are translated into a common data model. (2) Interschema relationship generation: related objects of the local schemas are identified and their relationship classified. (3) Integrated schema generation: an integrated schema is

generated based on the relationship between related objects. (4) Schema mapping generation: objects in the local schemas are mapped to the integrated schema.

There are two major abstraction levels of existing schema integration approaches. The majority of these approaches ([LNEM89, NEML86, SPD92, HR90, Ber91, KDN90, GMP<sup>+</sup>92]) integrate at the conceptual level, i.e. they consider the schemas of local databases. They identify conflicts of the schemas and propose methods to resolve these conflicts. Another class of approaches ([DeM89, PRSL93, Pu91, CS91, RR95, PRSL93]) integrate at the data level. They identify related objects and conflicts by looking at the values of the objects. [DeM89, PRSL93, Pu91, CS91] use a probabilistic model to infer tuples that refer to the same real-world object by looking at their attribute values. [PRSL93] identify equivalent tuples by instance-level functional dependencies. [RR95] infer integrity constraints in the integrated schema by investigating if the constraints that hold on a local schema also hold in another local schema.

These approaches differ in the type of heterogeneity and the semantics of the schema they consider. Some approaches deal with name conflicts while some assume there are no synonyms. Most approaches deal with structural conflicts. They identify related objects using different semantic information. For example, the domain and cardinality of attributes and integrity constraints for identifying related entities, names of participating entities for relationships, methods and typing of classes for class objects, keys, attributes and functional dependencies for tuples.

Schema integration approaches can also be classified according to the underlying common data model of the schemas. The most popular data model is the entity-relationship model ([LNEM89, EMN84, NEML86, SP94]). The relational model ([DeM89, CS91, PRSL93]) and object-oriented models ([TS93, GMP<sup>+</sup>92], [Ber91, KDN90]) have been also considered. More recently, approaches for XML data are proposed ([MZ98, Mur97])

In entity-relationship based approaches ([LNEM89, EMN84, NEML86, SP94]), related attributes are identified based on their domain, their cardinality, their names, uniqueness property, and allowable operations on them. Entities are compared based on their names, similarity of their attributes, and integrity constraints on them. In turn, relationships are compared on their names, their cardinality and the similarity of their participating entities. Based on the comparison, the semantic relationship between objects is classified. For example, [LNEM89] classifies the equivalence between attributes into EQUAL, CONTAINS, CONTAINED-IN and OVERLAP. Relationships between entities are classified based on the equivalence of their attributes accordingly. Relationships are categorized according to the similarities of their participating entities. Rules are used to integrate the entities and relationships based on class of their relationship, typically by generalization/specialization operations.

Object-oriented approaches integrate class hierarchies. Classes are compared by their types and their methods. To integrate methods of classes, mechanisms are proposed to resolve their heterogeneity (names, parameters). New methods may be defined for the integrated schema.

Ozone [LAW99] is a mediator system for XML data. It does not derive an automatic mapping to integrate XML data. [MZ98] considers data with very similar schema being represented in different data models, e.g., as XML DTD or an object-oriented schema. Using knowledge on the data types in different data models, rules are proposed to identify related objects in the two databases and to integrate them. [Mur97] proposes a language to allow users to specify how one DTD can be transformed into another. The system then automatically transforms the XML data into the new DTD based on the specification.

### 5.4.2 Comparison

The Document Transformer integrates a collection of XML documents according to their majority schema. Since a majority schema is in the form of an XML document, translating them into the same data model is not necessary. The derivation of the semantic mapping is the task of interschema relationship generation whereas the transformation process is the schema mapping generation task in the schema integration problem.

The Document Transformer integrates primarily at the conceptual level in the sense that related nodes in the XML trees are identified based on their names and their neighborhood in the trees. The textual content of the nodes is not considered in the integration (except in customizing the cost of the *Rename* operator in deriving a semantic mapping). The semantics considered are the concept names and the

neighborhood of the nodes in the document tree and the majority schema. There are no synonyms among the concept names, but homonyms are considered. Only the textual content of the documents is relevant. Other attributes of nodes are ignored.

Deriving the semantic mapping is in spirit similar to the interschema relationship generation task in entity-relationship approaches. Related objects are identified in the entity-relationship model based on attributes, entities and relationships. The Document Transformer also identifies related concept nodes based on their similarities in terms of their names and their neighboring concept nodes.

Identifying related classes in object-oriented based approaches is reminiscent to deriving the semantic mapping. However, the context of classes in an OODB is different from the context of nodes in XML documents. Classes are compared based on their inheritance, their methods, and their types. Nodes in XML documents are compared based on their names and their neighborhood, e.g., siblings, ancestors and descendants.

Although [MZ98, Mur97] both consider XML data, the problem tackled by the Document Transformer is different from theirs. [MZ98] considers the integration of data under a schema in different data models. The Document Transformer considers the integration of data having different schemas but are formulated in the same data model. [Mur97] provides a mechanism to allow the user to specify how to integrate the documents whereas the Document Transformer aims to derive automatically how integration can be achieved. Viewing each XML document having its own DTD,

the Document Transformer may use [Mur97] to transform an XML document to the majority schema based on the semantic mapping computed.

There are two major contributions by the Document Transformer. While many of the existing approaches ([DeS86, SLCN88, HR90, SM92, RR95, LNEM89]) are manual or semi-automatic, the Document Transformer automates the process of integrating topic specific XML documents. This is possible because relatively few semantics are considered in the integration process and we have knowledge on the topic specific documents, whereas existing approaches deal with many different types of structural conflicts on data with no known characteristics.

Second, existing schema integration approaches developed for relational are not directly applicable to XML data. The data model of XML is distinctly different from the data model of the entity-relationship or relational model. Although the hierarchical structures of classes in object-oriented data shares some similarities with the tree structures of XML documents, existing integration approaches on object-oriented data focus on types, inheritance, methods and dynamic behavior of classes, not on ancestor-descendant relationship. We proposed novel integration techniques that are reminiscent to those in the relational and object-oriented data to XML data. The insights offered by this work can be used for future research in object-oriented data.



## Chapter 6

### Conclusion

In this dissertation, we have presented a system called Quixote [CGS01] to solve the problem of integrating topic specific HTML documents into a repository of homogeneous XML documents. The three components of Quixote to realize this goal, the Document Converter, the Schema Miner and the Document Transformer, were described in detail in Chapter 3, 4 and 5 respectively. Detailed analytical studies and empirical results were presented, and comparisons with existing approaches were discussed. In this chapter, we summarize and scope future work. In Section 6.1, we summarize our contributions. In Section 6.2, we discuss some of the applications of the integrated XML repository. Finally, we highlight possible extensions to various components of Quixote in Section 6.3.

## 6.1 Contributions

Existing state-of-the-art information retrieval techniques cannot locate information in topic specific HTML documents effectively because (1) the information is buried in the HTML text, (2) no semantics are associated with the textual information since HTML is for visual rendering purpose, not for describing the information content of the documents, and (3) the HTML documents are heterogeneous in structure, which makes it difficult to automate the process of locating relevant information.

We have presented a flexible system, Quixote, that addresses these problems. Quixote extracts semantic information from topic specific HTML documents, encodes the information content of the HTML documents in XML documents, and integrates these XML documents based on a majority schema inferred. To achieve these goals, Quixote utilizes several novel techniques. The major contributions of each component are described below.

**Document Converter:** (1) It is automatic demanding minimal intervention from a user. (2) The Document Converter realizes several restructuring rules that convert the HTML documents to XML documents to match the logical information content of the HTML documents. The restructuring rules are based on the format clues of HTML markup tags and the semantics of XML tags, and are therefore domain-independent and are insensitive to changes in the formats of the data sources. (3) We assume minimal domain knowledge from the user. No schema on the documents

is assumed. But we provide a simple and extensible constraint mechanism to allow a user to specify knowledge about the structures of concepts pertaining to the topic. Such knowledge can improve the accuracy of data extraction and is optional.

**Schema Miner:** (1) A classification of the quality of a schema based on relevance and coverage is presented. (2) We introduce the notion of majority schema, an approximate schema that is always relevant and describes only the prevalent structures among the documents. Existing schema proposals - exact schema or approximate schema of low relevance - are inappropriate for HTML documents gathered from the Web which are heterogeneous in terms of structure. (3) The schema mining process is guided by the intuition that imprecise data modeling helps to reveal prevalent patterns. Unlike existing approaches, this principle maps the schema mining problem to a data mining problem with well known efficient mining algorithms. This reduces the complexity of the problem and leads to an efficient computation of a majority schema.

**Document Transformer:** (1) It automates the process of integrating XML documents with different structures. This problem is not addressed by existing literature. (2) The Document Transformer uses novel techniques for the identification of interschema relationship. Existing integration approaches on relational data and object-oriented data are not directly applicable to XML documents.

## 6.2 Applications

Information buried in the HTML text that is not readily accessible to users and applications is now made available to them by Quixote. Quixote transforms a heterogeneous collection of topic specific HTML documents into a homogeneous collection of XML documents conforming to a global schema. This integrated XML repository enables many applications. In this section, we highlight some of these applications with respect to information retrieval and data management.

### 6.2.1 Structured Queries

State-of-the-art information retrieval technology relies on keyword-based search engines which do not support structured queries on HTML documents. For example, in résumé HTML documents, one may ask "show me all résumés of people with Java programming skills". Keyword-based search engines may return résumés of people from Java, Indonesia. Encoding the information in the HTML documents in XML documents allows one to issue structured queries. Various query languages on XML data ([CRF00, DFF<sup>+</sup>98, AQM<sup>+</sup>97, JR98, W3C00b]) can be used to process such queries.

### 6.2.2 Summarization

A user unfamiliar with the information content of topic specific documents can gain a bird's eye view on the structures of the documents from the majority schema. A majority schema abstracts the heterogeneity among the documents and presents only prevalent structures to the user. In the résumé documents, there may be many different ways of describing a person's educational background. Some people may describe their marital status on their résumés while the majority of the people do not. Presenting all these structures to the user obscures the understanding of the major information content of the topic. On the other hand, a majority schema summarizes the information content and presents a succinct view to the user.

### 6.2.3 Data Presentation

From the point of view of an administrator, it is often useful to present HTML documents to the users under a uniform view and format. There is no tool that can automate this process on heterogeneous HTML documents. Automating this process on the integrated XML repository is possible by tools like XSL and XSLT [W3C00b, W3C99b], CSS [W3C96, W3C98a] and Strudel [FFK<sup>+</sup>97]. For example, a career site may want to present the educational background information on résumés from people's personal homepages to employers. The following shows an XSL stylesheet that locates relevant data from the XML documents (`CONTACT-INFO`, `DEGREE`, `DATE`,

ORGANIZATION) and presents them to the user in a tabular form. As illustrated, writing such stylesheet is simple and can easily be modified for different purposes.

```
<?XML version='1.0' encoding='Utf-8'?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/xsl/transform"
  xmlns:date="http://www.jclark.com/xt/java/java.util.date">
<xsl:output Method="html"/>
<xsl:template match="/html">
  <title> Educational Background Of Candidates </title>
  <body><h2> Educational Background Of Candidates </h2>
  <table border="1">
    <tr> <td colspan="2"> <b> Contact Information </b> </td></tr>
    <tr> <td> <b> Degree </b> </td>
      <td> <b> Organization </b> </td></tr>
    <xsl:apply-templates select="html"/>
  </table>
  </body>
</xsl:template>
<xsl:template match="html">
  <tr><td colspan="2">
    <b> <xsl:apply-templates select="CONTACT-INFO"/> </b></td></tr>
    <xsl:apply-templates select="EDUCATION"/>
  </xsl:template>
<xsl:template match="CONTACT-INFO">
  <xsl:value-of select="@val"/>
</xsl:template>
<xsl:template match="EDUCATION">
  <xsl:apply-templates select="DATE"/>
</xsl:template>
<xsl:template match="DATE">
  <tr><td><xsl:apply-templates select="DEGREE"/></td>
    <td><xsl:apply-templates select="ORGANIZATION"/></td></tr>
  </xsl:template>
<xsl:template match="DEGREE">
  <xsl:value-of select="@val"/>
```

```
</xsl:template>
<xsl:template match="ORGANIZATION">
  <xsl:value-of select="@val"/>
</xsl:template>
</xsl:stylesheet>
```

### 6.2.4 Indexing

A majority schema for XML documents can be used to build index structures on them to facilitate processing path queries. [FS98] proposes "state extents" as an index structure which stores the object identifiers of database objects in a graph schema. [MS99] proposes an index structure to answer path queries involving regular pattern matching. A similar technique can be used to build an index structure on the integrated XML documents based on a majority schema.

A majority schema is like a Dataguide for XML documents ([NUWC97]). Unlike an exact Dataguide, a majority schema is more concise. This results in an index requiring less space. An approximate Dataguide is smaller than the exact Dataguide from which it is derived. Unlike an approximate Dataguide which may contain irrelevant structures, a majority schema is always relevant.

### 6.2.5 Storage

A majority schema can be used to optimize the storage of XML documents. Assuming nodes in an XML document that are in close proximity in the majority schema are likely to relate to certain information content, these nodes are likely to be retrieved together in answering a user query. A user interested in a node in the document may also gather information about its children nodes (zoom in), sibling nodes and parent node (zoom out). The neighborhood of a node can thus be stored together on secondary storage devices to better facilitate information retrieval. Since the XML documents conform to the majority schema, tools like STORED [DFS99] can be used to store them in a relational database management system for efficient data management.

## 6.3 Future Work

The approach taken by Quixote gives a basic framework in integrating heterogeneous HTML documents. In this section, we describe some of the possible extensions to Quixote.

At present, the HTML documents are assumed to be static. In case they are changed constantly, it is desirable to update only the relevant portions in the corresponding XML document and to reflect the changes in the majority schema. Currently, changes made to an HTML document require an XML document to be derived from scratch.



One can keep track of the correspondence between sections in the HTML and XML document. Changes made to a section in the HTML document only require changes to the relevant section(s) in the XML document.

Information about the output of each component in Quixote can be used to improve other components. For example, a majority schema can be compared to an XML document to give a hint on the accuracy in the extraction process. A node in an XML document whose semantically related node in the majority schema has significantly low support may indicate an unusual structure in the document or it may indicate that a concept name is incorrectly identified or placed at the wrong place. Those nodes can be highlighted and presented to the user for inspection. Similarly, if the semantic mapping between the document and the majority schema identifies too few semantically related nodes in the document and the majority schema, it drops hints on possible errors in the document conversion phase. If the documents are gathered from the Web by keyword-based search engines, the keywords to the search engines can be refined by inspecting the textual content of high level concept nodes in the XML documents because high level concepts are more important in a topic than low level concepts.

It would be useful if Quixote has an interactive graphical user interface. Such interface can facilitate the user in labeling HTML text with concept names that are used by the Bayes classifier in concept identification in the document conversion process. By presenting an HTML document and its converted XML document side by side, the user can give feedback to the system on the accuracy of identifying concept names

from HTML text. This information, in turn, can be used to build up a repository of examples to the Bayes classifier as the system evolves. The tool can be used to visualize the relative importance of nodes in the documents based on their support in the majority schema (e.g., by intensity of the colors of the nodes), and in correlating structures in the documents with those in the majority schema (e.g. by colors of the nodes). Semantic integration is not a trivial task. Although the Document Transformer is automated, some user interaction is needed to ensure the transformation is performed correctly. The graphical user interface can be used to visualize the semantic mapping proposed to the user, so that she can correct a document before transformation is performed.

We now describe extensions specific to the components of Quixote. With respect to the Document Converter, the current implementation of the Document Converter encodes restructuring rules in the algorithms to realize the various steps in the data extraction process. A rule-based language would give a more flexible system for the user to modify rules which are currently hard coded in the process. One can investigate the formalism of the rule-based language and how rules can be specified under such formalism.

The current implementation does not consider the visual clues given by HTML markup tags like `font` or spacing `nbsp`. Elements in a table can be organized in a row major / column major manner. One can take such order into consideration in deriving subtree structures of elements in the table. Another extension is to view a sequence of markup tags as a unit. Since sections are usually marked up in the same

way, repeating sequences of markup tags can be used to detect boundaries between records. The Document Converter recognizes tokens from sentences by punctuation. Grammar-based parsing techniques and more sophisticated techniques based on spacing can be used to tokenize the texts in the document.

Concept identification requires examples on how to associate text with keywords. One obstacle of putting the system to use is that a large amount of data is required to obtain high classification accuracy. This problem can be alleviated by feeding the system with a database of known facts. For example, popular college guides would have the names of universities in the states. Suppose a user is interested in giving examples on a concept related to universities. The names of these universities can be used as instances for this concept. Similarly, databases of the names of fortune 500 companies, common proper names, geographical locations can be used for concepts related to companies, names and locales. The Bayes classifier can be fine-tuned in a number of ways, such as considering only the top  $k$  words, using more sophisticated parameter smoothing methods, or computing the discriminating power of words by their mutual occurrences. Other machine learning techniques can also be explored to identify concepts. A meta-classifier can be built to combine the results from various techniques.

With respect to the Schema Miner, one can enrich the formalism for tree schemas. A tree schema does not model choices of its constituent elements in the content model of an element. Considering these sequences substantially increases the search space. An extension would be to design a practical method to compute these sequences. We do

not consider IDREFs and recursive definitions of elements. Further investigation is needed to extend the schema discovery algorithm to take recursion into consideration. A tree schema can also be extended to model attributes in XML documents. DTD is the standard formalism of a schema for XML data. Nevertheless, a number of proposals [W3C00f, W3C00g] have been submitted to W3C Consortium to provide a richer formalism to describe XML data. For example, in [W3C00f, W3C00g], there are data types in addition to the string primitive data type in DTDs. There is also inheritance between complex types. Data can assume different ranges and there are mechanisms to specify primary and foreign key constraints. Further research is needed to extend the majority schema to consider these issues.

A majority schema can be used to build index structures to optimize path queries and to optimize storage of the documents. Concept nodes with the high support in a majority schema are most commonly found. Assuming this implies they are most commonly accessed, structures in the documents conforming to these nodes can be indexed for more efficient retrieval. Ideas similar to [NUWC97, MS99, DFS99] can also be investigated which index objects with high support.

With respect to the Document Transformer, one can investigate a more sophisticated cost model for the edit operators. The current cost model for the *Rename* operator for deriving a semantic mapping assumes a simple formulation. Weights can be associated with different components in the neighborhood component in the formulation. Different non-linear formulations can also be investigated. At present, the cost of the *Insert* and *Delete* operators range from 0 to 1 while that of the *Rename* operator

ranges from 1 to infinity. One can look into assignments of different ranges for these operators. The cost of transforming a source tree to a target is bounded by the sum of the cost of deleting all nodes in the source tree and of inserting all nodes from the target tree. The range of the *Rename* operator can be bounded by this cost. Assuming a weighted sum formulation for the cost model, it would be helpful if the system can automatically fit the best weights for the cost model. The user can give training examples detailing the best transformation from an XML document to a majority schema. Machine learning approaches can be used to search for the weights that best fit the examples given by the user.

The Document Transformer heuristically searches the siblings, followed by descendants and ancestors of a node. Further research is needed to investigate the effect of different search order to locate the best match in case there are more than one.

Quixote integrates topic specific HTML documents which satisfy the properties detailed in Section 1.2.3. While these properties are general assumptions on the information content of documents, it would be interesting to relax the Regular Intradocument Format Property. This implies that there may be choices of structures of constituent elements in the content model of an element in a DTD. This impacts the requirements of the schema formalism to model the majority schema, and the derivation and refinement of a semantic mapping in the transformation process.

At present, we consider single documents only. As HTML documents are linked together by hyperlinks in the Web, a useful extension is to consider the anchor text and

relationships between HTML documents based on hyperlinks. For example, sections of a person's résumé may be linked together by hyperlinks with the anchor text of a hyperlink describing the content of the section involved. Another example is a site map of a Web site. HTML pages of the Web site are related to each other by hyperlinks. The information content of the linked HTML documents not only concerns the information content of single documents, but also the relationships between HTML documents which may not be hierarchical in nature. There are several implications. XLink and XPointers [W3C00c] can be used to capture the hyperlinks. Since the documents may not be linked in a hierarchy, cycles may be present which need to be considered in the schema. Some HTML documents may relate to the same type of information and the types may be related by generalization/specialization relationship, e.g., a corporate Web site may have several pages on different products and some products are within the category of some other products. Detecting groups of and relationships between these HTML documents is necessary in deriving a concise and precise schema.

Another challenging extension is to investigate topic specific documents in other formats, e.g., plain text, postscript. The Document Converter exploits the format clues of HTML documents which are readily available from the HTML markup tags and the tree structures of the documents. However, these clues may not be present or may not be readily available from documents in other formats. It may be worthwhile to consider language parsing techniques in deriving the relationship between information objects in these documents.

# Bibliography

- [ABS99] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.
- [Ade98] B. Adelberg. NoDoSE: A tool for semi-automatically extracting semi-structured data from text documents. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 283–294, Washington, USA, June 2–4, 1998. ACM Press.
- [AES98] M. Rusinkiewicz A. Elmagarmid and A. Sheth, editors. *Management of Heterogeneous and Autonomous Database Systems*. Morgan Kaufmann Publishers, San Francisco, California, 1998. Chapter 5.
- [AK97] N. Ashish and C. A. Knoblock. Semi-automatic Wrapper Generation for Internet Information Sources. In *Proceedings of the Second IFCIS International Conference on Cooperative Information Systems (CoopIS'97)*, pages 160–169, Los Alamitos, CA, June 1997. IEEE.

- [AM97] P. Atzeni and G. Mecca. Cut & paste. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 144–153, Tucson, Arizona, May 12–14, 1997. ACM Press.
- [AQM<sup>+</sup>97] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The Lorel query language for Semistructured Data. In *International booktitle on Digital Libraries*, volume 1(1), pages 68–88. Springer, 1997.
- [AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 487–499, Santiago, Chile, 1994. Morgan Kaufmann Publishers.
- [BC00] A. Bonifati and S. Ceri. Comparative Analysis of Five XML Query Languages. In *Proceedings ACM SIGMOD International Conference on Management of Data*, volume 29(1) of *SIGMOD Record*, pages 68–79. ACM Press, March 2000.
- [BDFS97] P. Buneman, S. B. Davidson, M. F. Fernandez, and D. Suciu. Adding structure to unstructured data. In *Proceedings of International Conference on Database Theory (ICDT)*, volume 1186 of *Lecture Notes in Computer Science*, pages 336–350, Delphi, Greece, January 8–10, 1997. Springer.
- [BDHS96] P. Buneman, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimisation techniques for unstructured data. In *Proceed-*



- ings ACM SIGMOD International Conference on Management of Data*, pages 505–516, Quebec, Canada, June 4–6, 1996. ACM Press.
- [Ber91] E. Bertino. Integration of heterogeneous data repositories by using object-oriented views. In *Proceedings of the First International Workshop on Interoperability in Multidatabase Systems (IMS)*, pages 22–39, Kyoto, 1991. IEEE.
- [BLFD99] T. Berners-Lee, M. Fischetti, and M. Dertouzos. *Weaving the Web : The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. Harper San Francisco, October 1999.
- [BMR99] D. Beech, A. Malhotra, and M. Rys. A Formal Data Model and Algebra for XML. Submission to the World Wide Web Consortium - [elib.cs.berkeley.edu/seminar/2000/20000207.html](http://elib.cs.berkeley.edu/seminar/2000/20000207.html), 1999.
- [CDSS98] S. Cluet, C. D., J. Siméon, and Katarzyna Smaga. Your mediators need data conversion ! In *Proceedings ACM SIGMOD International Conference on Management of Data*, volume 27(2) of *ACM SIGMOD Record*, pages 177–188, New York, U.S.A., June 1–4, 1998. ACM Press.
- [CGS01] C.Y. Chung, M. Gertz, and N. Sundaresan. Quixote: Building XML Repositories from Topic Specific Web Documents. In *Fourth International Workshop on the Web and Databases (WebDB)*, Santa Barbara, CA, 2001.

- [Cha00] S. Chakrabarti. Data mining for hypertext: A tutorial survey. In *SIGKDD Explorations: Newsletter of the Special Interest Group (SIG) on Knowledge Discovery & Data Mining*, volume 1. ACM Press, 2000.
- [CHS+95] M. J. Carey, L. M. Haas, P. M. Schwarz, M. Arya, W. F. Cody, R. Fagin, M. Flickner, A. W. Luniewski, W. Niblack, D. Petkovic, J. Thomas, J. H. Williams, and E. L. Wimmers. Towards Heterogeneous Multimedia Information Systems: The Garlic Approach. In *Research Issues in Data Engineering (RIDE)*, pages 124–131, California, USA, March 1995. IEEE.
- [CRF00] D. D. Chamberlin, J. Robie, and D. Florescu. Quilt: An XML Query Language for Heterogeneous Data Sources. In *Proceedings of the Third International Workshop on the Web and Databases*, pages 53–62, Dallas, Texas, U.S.A., May 18–19, 2000.
- [CS91] A. Chatterjee and A. Segev. A probabilistic approach to information retrieval in heterogeneous databases. In *Proceedings of the First Workshop on Information Technology Systems*, pages 107–124, 1991.
- [CS99] C.Y. Chung and N. Sundaresan. System and Method for Discovering Schematic Structure in Hypertext Documents., 1999. IBM Patent Application AM9-99-0173. Filed.

- [CS00] C.Y. Chung and N. Sundaresan. Method and System for Discovering a Majority Schema in Semi-Structured Data, 2000. IBM Patent Application ARC9-2000-0117-US1. Filed.
- [DeM89] L. G. DeMichiel. Resolving database incompatibility: An approach to performing relational operations over mismatched domains. In *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, volume 1(4), pages 484–493. IEEE, 1989.
- [DeS86] J. M. DeSouza. Sis – a schema integration system. In *Proceedings of the Fifth British National Conference on Databases*, pages 167–185. Cambridge University Press, 1986.
- [DFE+98] A. Deutsch, M. F. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: A Query Language for XML. Submission to W3C - <http://www.w3.org/NOTE-xml-ql-19980819>, 1998.
- [DFS99] A. Deutsch, M. F. Fernandez, and D. Suciu. Storing semistructured data with STORED. In *Proceedings ACM SIGMOD International Conference on Management of Data*, volume 28(2), pages 431–442, New York, USA, June 1–3, 1999. ACM Press.
- [DYKR00] H. Davulcu, G. Yang, M. Kifer, and I. Ramakrishnan. Computational Aspects of Resilient Data Extraction from Semistructured Sources. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 136–144, New York, USA, 2000. ACM Press.

- [ECJ+98] D. Embley, D. Campbell, Y. Jiang, Y. Ng, R. Smith, S. Liddle, and D. Quass. A Conceptual-Modeling Approach to Extracting Data from the Web. In *Proceedings of International Conference on Conceptual Modeling (ER)*, volume 1507 of *Lecture Notes in Computer Science*, pages 78–91, Singapore, November 1998. Springer-Verlag.
- [ECSL98] D. W. Embley, D. M. Campbell, R. D. Smith, and S. W. Liddle. Ontology-based extraction and structuring of information from data-rich unstructured documents. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, pages 52–59, New York, USA, November 1998. ACM Press.
- [EMN84] R. El-Masri and S. Navathe. Object integration in logical database design. In *IEEE Transactions on Data Engineering*, pages 426–433. IEEE, 1984.
- [Eve73] Brian Everitt. *Cluster Analysis*. John Wiley & Sons - New York, 1973.
- [FFK+97] M. F. Fernandez, D. Florescu, J. Kang, A. Levy, and D. Suciu. STRUDEL: a Web site management system. In *Proceedings ACM SIGMOD International Conference on Management of Data*, volume 26(2), pages 549–552, New York, USA, May 13–15, 1997. ACM Press.
- [FMHA98] M. Fuketa, S. M., Y. H., and J. I. Aoe. A Fast Method of Determining Weighted Compound Keywords from Text Databases. In *Information*

- Processing and Management*, volume 34(4), pages 431–442. Elsevier, 1998.
- [FS98] M. F. Fernandez and D. Suciu. Optimizing regular path expressions using graph schemas. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 14–23, Florida, USA, February 23–27, 1998. IEEE.
- [FS00] W. Fan and J. Simeon. Integrity Constraints for XML. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 23–34, Texas, USA, May 15–17, 2000. ACM Press.
- [FSSW] M. Fernandez, J. Simeon, D. Suciu, and P. Wadler. A Data Model and Algebra for XML Query. [www.cs.bell-labs.com/wadler/topics/recent.html](http://www.cs.bell-labs.com/wadler/topics/recent.html).
- [GGR<sup>+</sup>00] M. N. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: A System for Extracting Document Type Descriptors from XML Documents. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 165–176, Texas, U.S.A., May 14–19, 2000. ACM Press.
- [GMP<sup>+</sup>92] J. Geller, A. Mehta, Y. Perl, E. Neuhold, and P. Sheth. Algorithms for structural schema integration. In *Proceedings of the Second International Conference on Systems Integration*, pages 604–614, 1992.

- [GMV99] N. Guarino, C. Masolo, and G. Vetere. Ontoseek: Content-based access to the web. In *IEEE Intelligent Systems*, volume 14(3), pages 70–80. IEEE, 1999.
- [GW99] R. Goldman and J. Widom. Approximate dataguides. In *Proceedings of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*, Jerusalem, Israel, January 1999.
- [HGMC<sup>+</sup>97] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. Extracting semistructured information from the Web. In *Proceedings of the Workshop on Management of Semi-Structured Data*, pages 18–25. ACM Press, 1997.
- [HGMI<sup>+</sup>95] J. Hammer, H. Garcia-Molina, K. Ireland, Y. Papakonstantinou, J. D. Ullman, and J. Widom. Information translation, mediation, and mosaic-based browsing in the tsimmis system. In *Proceedings ACM SIGMOD International Conference on Management of Data*, volume 24(2), page 483, San Jose, California, May 22–25. 1995. ACM Press.
- [HR90] S. Hayne and S. Ram. Multi-user view integration system (MUVIS): An expert system for view integration. In *Proceedings of the International Conference on Data Engineering (ICDE)*, Los Alamitos, CA, February 1990. IEEE.
- [IBM97] IBM Almaden Research Center. IBM: all searches start at Grand Central. In *Network World Front Page*, November 1997.

- [JR98] D. Schach, J. Robie, J. Lapp. XML Query Language (XQL). In *W3C QL98 Workshop*, 1998. <http://www.w3.org/TandS/QL/QL98/pp/xql.html>.
- [KDN90] M. Kaul, K. Drosten, and E. J. Neuhold. Viewsystem: Integrating heterogeneous information bases by object-oriented views. In *Proceedings of the Sixth International Conference on Data Engineering*, pages 2–10. IEEE, February 1990.
- [Kus00] N. Kushmerick. Wrapper induction: Efficiency and expressiveness. In *Artificial Intelligence*, volume 118(1–2), pages 15–68. Elsevier Science, 2000.
- [Lap95] P. S. Laplace. *Philosophical Essays on Probabilities*. Springer-Verlag, New York, 1995.
- [LAW99] T. Lahiri, S. Abiteboul, and J. Widom. Ozone: Integrating Structured and Semistructured Data. In *Proceedings of the Seventh International Workshop on Database Programming Languages*, Kinloch Rannoch, Scotland, September 1999.
- [LC00] D. Lee and W. Chu. Comparative Analysis of Six XML Schema Languages. In *Proceedings ACM SIGMOD International Conference on Management of Data*, volume 29(3) of *SIGMOD Record*. ACM Press, September 2000.

- [LNEM89] J. Larson, S. B. Navathe, and R. El-Masri. A theory of attribute equivalence and its applications to schema integration. In *IEEE Transactions on Software Engineering*, volume 15(4), pages 449–463. IEEE, April 1989.
- [LPVV99] B. Ludaescher, Y. Papakonstantinou, P. Velikhov, and V. Vianu. View Definition and DTD Inference for XML. In *Proceedings of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats*, 1999.
- [LRO96] A. Y. Levy, A. Rajaraman, and J. J. Ordille. The world wide web as a collection of views: Query processing in the information manifold. In *Workshop on Materialized Views: Techniques and Applications (VIEW 1996)*, pages 43–55, Montreal, Canada, June 1996.
- [Lu79] S. Y. Lu. A tree-to-tree distance and its application to cluster analysis. In *IEEE Transaction on Pattern Analysis and Machine Intelligence*, volume 1(2), pages 219–224. IEEE, 1979.
- [MS99] T. Milo and D. Suciu. Index structures for path expressions. In *Proceedings of International Conference on Database Theory (ICDT)*, volume 1540 of *Lecture Notes in Computer Science*, pages 277–295, Jerusalem, Israel, January 10–12, 1999. Springer.
- [MSV00] T. Milo, D. Suciu, and V. Vianu. Typechecking for XML Transformers. In *Proceedings of the ACM Symposium on Principles of Database*



- Systems (PODS)*, pages 11–22, Texas, USA, May 15–17, 2000. ACM Press.
- [Mur97] M. Murata. Transformation of documents and schemas by patterns and contextual conditions. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, volume 1293 of *Lecture Notes in Computer Science*, pages 153–169, Palo Alto, California, September 1997. Springer.
- [MZ98] T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 122–133, New York, USA, August 24–27, 1998. Morgan Kaufmann.
- [NAM97] S. Nestorov, S. Abiteboul, and R. Motwani. Inferring structure in semistructured data. In *Proceedings ACM SIGMOD International Conference on Management of Data*, volume 26(4) of *SIGMOD Record*, pages 39–43, 1997.
- [NAM98] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting schema from semistructured data. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 295–306, Washington, USA, June 2–4, 1998. ACM Press.

- [NEML86] S. Navathe, R. El-Masri, and J. Larson. Integrating user views in database design. In *IEEE Computer*, volume 19(1), pages 50–62. IEEE, Jan 1986.
- [NUWC97] S. Nestorov, J. D. Ullman, J. L. Wiener, and S. S. Chawathe. Representative objects: Concise representations of semistructured, hierarchial data. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 79–90, Birmingham U.K., April 7–11, 1997. ACM Press.
- [PGMU96] Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. Medmaker: A mediation system based on declarative specifications. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 132–141, Washington - Brussels - Tokyo, February 1996. IEEE.
- [PRSL93] S. Prabhakar, J. Richardson, J. Srivastava, and E. P. Lim. Instance-level integration in federated autonomous databases. In *Proceedings of the 26th Annual Hawaii International Conference on System Sciences*, volume III, pages 62–69. IEEE, 1993.
- [Pu91] C. Pu. Key equivalence in heterogeneous databases. In *Proceedings the First International Workshop on Interoperability in Multidatabase Systems (IMS)*, pages 314–317, 1991.
- [PV99] Y. Papakonstantinou and P. Velikhov. Enhancing Semistructured Data Mediators with Document Type Definitions. In *Proceedings of the In-*

- ternational Conference on Data Engineering (ICDE)*, pages 136–145, Sydney, Australia, March 1999. IEEE.
- [PV00] Y. Papakonstantinou and V. Vianu. DTD Inference for Views of XML Data. In *Proceedings of the ACM Symposium on Principles of Database Systems (PODS)*, pages 35–46, Texas, USA, May 15–17, 2000. ACM Press.
- [Rag98] D. Raggett. Clean up your Web pages with HP’s HTML Tidy. In *Computer Networks and ISDN Systems*, volume 30(1–7), pages 730–732. Elsevier Science, April 1998.
- [RR95] V. Ramesh and S. Ram. A methodology for interschema relationship identification in heterogeneous databases. In *Proceedings of the 28th Annual Hawaii International Conference on System Sciences. Volume 3: Information Systems - Decision Support and Knowledge-Based Systems*, pages 263–272, Los Alamitos, CA, USA, January 1995. IEEE.
- [SA99] A. Sahuguet and F. Azavant. Looking at the Web through XML Glasses. In *Proceedings of the Fourth IFCIS International Conference on Cooperative Information Systems*, pages 148–159, Edinburgh, Scotland, September 2–4, 1999. IEEE.
- [SC99] M. Sanderson and W. B. Croft. Deriving Concept Hierarchies from Text. In *Proceedings of the Annual International ACM SIGIR Confer-*

- ence on Research and Development in Information Retrieval (SIGIR)*, pages 206–213, Berkeley, CA, August 15–19, 1999. ACM Press.
- [SLCN88] A. Sheth, J. Larson, A. Cornelio, and S. B. Navathe. A tool for integrating conceptual schemata and user views. In *Proceedings of the Fourth International Conference on Data Engineering*, pages 176–183, Los Alamitos, CA, February 1988. IEEE.
- [SM92] A. P. Sheth and H. Marcus. Schema analysis and integration: Methodology, techniques and prototype toolkit. Technical Report TM-ST-019981/1, Bellcore, 1992.
- [SP94] S. Spaccapietra and C. Parent. View integration: A step forward in solving structural conflicts. In *IEEE Transactions on Knowledge and Data Engineering*, volume 6(2). IEEE, April 1994.
- [SPD92] S. Spaccapietra, C. Parent, and Y. Dupont. Model independent assertions for integration of heterogeneous schemas. In *The VLDB Journal*, volume 1(1). Morgan Kaufmann, July 1992.
- [Tai79] K. C. Tai. The tree-to-tree correction problem. In *booktitle of the ACM*, volume 26(3), pages 422–433. ACM Press, July 1979.
- [TS93] C. Thieme and A. Seibes. Schema integration in object-oriented databases. In *Proceedings of the Fifth International Symposium on Advanced Information Systems Engineering (CAiSE)*, pages 54–70. Springer, 1993.

- [W3C96] W3C Working Group. Cascading style sheets, level 1. W3C Recommendation - <http://www.w3.org/pub/WWW/TR/REC-CSS1>, December 1996.
- [W3C98a] W3C Working Group. Cascading style sheets, level 2 css2 specification. W3C Recommendation - <http://www.w3.org/TR/REC-CSS2>, May 1998.
- [W3C98b] W3C Working Group. Extensible Markup Language (XML) 1.0. W3C Recommendation - <http://www.w3.org/TR/REC-xml>, February 1998.
- [W3C99a] W3C Working Group. XML Information Set. W3C Recommendation - <http://www.w3.org/TR/xml-info>, December 1999.
- [W3C99b] W3C Working Group. Xsl transformations (xslt) version 1.0. W3C Recommendation - <http://www.w3.org/TR/1999/REC-xslt>, 1999.
- [W3C00a] W3C Working Group. Document Object Model (DOM) Level 1 Specification ( Second Edition). W3C Recommendation - <http://www.w3.org/TR/REC-DOM-Level-1/>, September 2000.
- [W3C00b] W3C Working Group. Extensible Stylesheet Language (XSL) Version 1.0. W3C Recommendation - <http://www.w3.org/TR/2000/WD-xsl>, October 2000.
- [W3C00c] W3C Working Group. XML linking language (XLink) version 1.0. W3C Recommendation - <http://www.w3.org/TR/xlink/>, 2000.

- [W3C00d] W3C Working Group. XML Query Data Model. W3C Recommendation - <http://www.w3.org/TR/query-datamodel>, May 2000.
- [W3C00e] W3C Working Group. XML Query Requirements. W3C Recommendation - <http://www.w3.org/TR/2000/WD-xmlquery-req>, January 2000.
- [W3C00f] W3C Working Group. XML Schema Part 1: Structures. W3C Recommendation - <http://www.w3.org/TR/2000/WD-xmlschema-1>, April 7, 2000.
- [W3C00g] W3C Working Group. XML Schema Part 2: Datatypes. W3C Recommendation - <http://www.w3.org/TR/2000/WD-xmlschema-2>, April 7, 2000.
- [WL98] K. Wang and H. Liu. Discovering typical structures of documents: A road map approach. In *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 146–154, Melbourne, Australia, August 1998. ACM Press.
- [WL99] K. Wang and H. Liu. Discovering association of structure from semistructured objects. In *IEEE Transactions on Knowledge and Data Engineering, (TKDE)*, volume 12(2). IEEE, 1999.
- [WYW00] Q. Y. Wang, J. X. Yu, and K. F. Wong. Approximate graph schema extraction for semi-structured data. In *Proceedings of International Conference on Extending Database Technology (EDBT)*, volume 1777

of *Lecture Notes in Computer Science*, pages 302–316, Konstanz, Germany, March 27–31, 2000. Springer.

- [Zha96] K. Z. Zhang. A constrained edit distance between unordered labeled trees. In *Algorithmica*, volume 15(3), pages 205–222. Springer, March 1996.





# Symbols

<b>elem</b>	domain of <i>elem</i>
$\Sigma$	domain of alphabets
<b>S</b>	domain of strings (concatenation of alphabets)
<b>Z</b>	domain of integers
<b>R</b>	domain of real numbers
<b>Z<sup>+</sup></b>	domain of natural numbers
<b>boolean</b>	domain of boolean
$\epsilon$	empty
$\perp$	null or undefined
$\sqcup$	space character
$\infty$	infinity
<b>S<sup>*</sup></b>	sequence built over elements from set <i>S</i>
<i>Set(S)</i>	set built over elements from set <i>S</i>
<i>PowerSet(S)</i>	power set built over elements from set <i>S</i>
$\circ$	concatenation operator on strings
$\circ$	concatenation operator on label paths
$\oplus$	concatenation operator on node paths

$V$	domain of vertices
$E, E'$	domain of labels in an XML document (concept names)
$A$	domain of attributes names
$X$	domain of XML documents
$X, X'$	XML document
$\mathcal{X}$	collection of XML documents
$S$	tree schema or majority schema
$D$	DTD (from a majority schema)
$M$	XML document transformed from $X$
$H$	XML document in HTML markup tags
$Deg(X)$	degree of the XML document $X$
$Depth(X)$	maximum depth of vertices in $X$
$ X $	number of vertices of XML document $X$
$T(v)$	subtree rooted at vertex $v$

$\mathcal{C}$	language elements for the content model of an element in a DTD
$\mathcal{L}$	language for concept constraints
$\mathcal{C}$	concept instances, a set of $(label, keyword/text)$ pairs
$\mathcal{P}$	a set of maximal frequent label paths
$\mu$	conformance mapping of an XML document to tree schema
$\nu$	semantic mapping from an XML document to tree schema
$\theta$	transformation mapping to convert an XML document to another
$\sigma$	mapping from an XML document to one with repetitive label paths removed
<b>punc</b>	punctuation delimiters
<b>group</b>	HTML tags that are group tags
<b>list</b>	HTML tags that are list tags
<b>supThreshold</b>	threshold of support for frequent label paths
<b>ratioThreshold</b>	threshold of support ratio for frequent label paths
<b>maxPathLength</b>	maximum length of label paths considered in schema discovery
<b>distThreshold</b>	threshold to check if two trees are structurally equivalent